SIMULATION OF A MULTIPROCESSOR

COMPUTER SYSTEM

A.M. SALIH

Submitted in partial fulfilment of the requirements for the Doctor of Philosophy degree.

Sponsoring Establishment: Plymouth Polytechnic,
Drake Circus,
Plymouth

Collaborating Establishment: The General Electric Company Ltd
GEC Hirst Research Centre
Wembley.

MAY, 1981
ABSTRACT

A.M. SALIH

SIMULATION OF A MULTIPROCESSOR COMPUTER SYSTEM

The introduction of computers and software engineering in telephone switching systems has dictated the need for powerful design aids for such complex systems. Among these design aids simulators - real-time environment simulators and flat-level simulators - have been found particularly useful in stored program controlled switching systems design and evaluation. However, both types of simulators suffer from certain disadvantages.

An alternative methodology to the simulation of stored program controlled switching systems is proposed in this research. The methodology is based on the development of a process-based multi-level hierarchically structured software simulator. This methodology eliminates the disadvantages of environment and flat-level simulators. It enables the modelling of the system in a 1 to 1 transformation process retaining the sub-systems interfaces and, hence, making it easier to see the resemblance between the model and modelled system and to incorporate design modifications and/or additions in the simulator.

This methodology has been applied in building a simulation package for the System X family of exchanges. The Processor Utility Sub-system used to control the exchanges is first simulated, verified and validated. The application sub-systems models are then added one level higher, resulting in an open-ended simulator having sub-systems models at different levels of detail and capable of simulating any member of the System X family of exchanges. The viability of the methodology is demonstrated by conducting experiments to tune the real-time operating system and by simulating a particular exchange - The Digital Main Network Switching Centre - in order to determine its performance characteristics.
ADVANCE STUDIES UNDERTAKING IN CONNECTION WITH THE PROGRAMME OF RESEARCH IN PARTIAL FULFILMENT OF THE REQUIREMENTS OF THE DOCTOR OF PHILOSOPHY DEGREE

<table>
<thead>
<tr>
<th>No.</th>
<th>Title of Study/Conference</th>
<th>Institution/Location</th>
<th>Dates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>Mini-Computers and their Applications</td>
<td>Plymouth Polytechnic</td>
<td>7-18 June, 1976</td>
</tr>
<tr>
<td>3)</td>
<td>Stored Program Control of Telephone Switching Systems</td>
<td>IEE Vacation School, Essex University</td>
<td>11-16 Sept. 1977</td>
</tr>
<tr>
<td>4)</td>
<td>Simulation with SIMULA and SIMON</td>
<td>Exeter University</td>
<td>Oct-Dec. 1977</td>
</tr>
<tr>
<td>5)</td>
<td>Switching and Signalling in Telecommunications Networks</td>
<td>IEE Vacation School, University of Aston in Birmingham</td>
<td>10-15 Sept. 1978</td>
</tr>
<tr>
<td>6)</td>
<td>Association of SIMULA Users' Seminar</td>
<td>University of Bradford</td>
<td>19th Dec. 1978</td>
</tr>
<tr>
<td>7)</td>
<td>Queuing Theory</td>
<td>Imperial College</td>
<td>8th May - 5th June, 1980</td>
</tr>
<tr>
<td>8)</td>
<td>Data Networks</td>
<td>Essex University</td>
<td>13th Jan - 10th Feb. 1981</td>
</tr>
</tbody>
</table>

DECLARATION:

The author of this thesis hereby declares that:

(1) While registered as a candidate for the Ph.D. degree the author has not been a registered candidate for another award of the CNAA or a University during the research programme.

(2) No material contained in this thesis has been used in any other submission for an academic award.

........................................
A. M. SALIH
ACKNOWLEDGEMENTS

Acknowledgement is due to Plymouth Polytechnic whose financial support has made it possible to carry out this research and to GEC Telecommunications and GEC Hirst Research Centre whose interest in this work was motivating. My thanks and gratitude to the Director of Research, Dr. E. McQuade and to my Supervisors Mr. H. Gray and Mr. G.J. Owen for their continued encouragement, advice and interest which made it possible to complete this programme of research.

I would like to acknowledge the contribution made to my understanding of System X sub-systems and simulation approaches and languages by all those whom I met and have discussed these topics over the past four years. In particular, I would like to thank Messrs J. Fanstone, J. Nissen and G. Dent of GEC Telecommunications, F.M. Clayton, D. Bear and V.R. Barker of GEC Hirst Research Centre, and R.J. Cunningham of the Imperial College whose interest and provision of computing facilities made it possible to complete the last phase of this research.

My thanks are also conveyed to the other establishments, namely The Institute of Marine Environmental Research and Exeter University Computer Unit who generously provided us with the necessary computing facilities during the first three years.

Last but not least I would like to thank my wife Fatima for her patience and understanding during the course of this research.
CHAPTER 1
INTRODUCTION

CHAPTER 2
DISCRETE-EVENT SIMULATION APPROACHES & LANGUAGES

2.1 Discrete-Event Simulation Approaches & Techniques
   2.1.1 Systems
   2.1.2 Models
   2.1.3 System Simulation
   2.1.4 Discrete-Event Simulation Approaches
      2.1.4.1 The Approaches
      2.1.4.2 Simulation Executive Program
      2.1.4.3 An Example and a Summary

2.2 Discrete-Event Simulation Programming Languages (SPL)
   2.2.1 Introduction
   2.2.2 Static Representation
   2.2.3 Dynamic Representation
   2.2.4 Simulation Support

2.3 Criteria for Simulation Programming Language Selection

2.4 SIMULA 67

CHAPTER 3
THE TELECOMMUNICATION NETWORK & STORED PROGRAM CONTROLLED SWITCHING-SYSTEMS

3.1 The Telecommunication Network

3.2 Types of Telephone Exchanges
   3.2.1 Introduction
   3.2.2 Manual Exchanges
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.3 Direct Control Systems</td>
<td>67</td>
</tr>
<tr>
<td>3.2.4 Register Control Systems</td>
<td>69</td>
</tr>
<tr>
<td>3.2.5 Electronic Systems</td>
<td>73</td>
</tr>
<tr>
<td>3.3 Stored Program Controlled (SPC) Systems</td>
<td>75</td>
</tr>
<tr>
<td>3.4 Telecommunications Processors</td>
<td>76</td>
</tr>
<tr>
<td>3.4.1 Their Characteristics</td>
<td>76</td>
</tr>
<tr>
<td>3.4.2 Configurations of the Processor Utility</td>
<td>79</td>
</tr>
<tr>
<td>3.5 SPC Software Organisation</td>
<td>87</td>
</tr>
<tr>
<td>3.6 System X</td>
<td>91</td>
</tr>
<tr>
<td>3.6.1 System X Characteristics</td>
<td>91</td>
</tr>
<tr>
<td>3.6.2 System X Architecture</td>
<td>93</td>
</tr>
<tr>
<td>3.6.3 System X Subsystems</td>
<td>95</td>
</tr>
<tr>
<td>3.6.4 System X Family of Exchanges</td>
<td>98</td>
</tr>
<tr>
<td>3.7 Performance Evaluation of Computer and SPC Switching</td>
<td>100</td>
</tr>
<tr>
<td>Systems</td>
<td></td>
</tr>
<tr>
<td>3.7.1 The Objectives</td>
<td>100</td>
</tr>
<tr>
<td>3.7.2 Performance Evaluation Techniques</td>
<td>105</td>
</tr>
<tr>
<td>3.7.2.1 Introduction</td>
<td>105</td>
</tr>
<tr>
<td>3.7.2.2 Workload Characterisation and Modelling</td>
<td>106</td>
</tr>
<tr>
<td>3.7.2.3 Performance Evaluation Models</td>
<td>109</td>
</tr>
<tr>
<td>3.7.3 Performance Evaluation through Simulation Modelling</td>
<td>111</td>
</tr>
<tr>
<td>3.7.4 Simulation in the Field of SPC Switching Systems</td>
<td>113</td>
</tr>
<tr>
<td>3.7.5 Mark II BL Simulator Package in Relation to Other SPC Systems</td>
<td>118</td>
</tr>
</tbody>
</table>
CHAPTER 4

THE GEC MARK II BL MULTIPROCESSOR COMPUTER SYSTEM 121

4.1 Introduction 121

4.2 Mark II BL Hardware 122

4.3 Mark II BL Software 131

4.3.1 Introduction 131

4.3.2 The Real Time Operating System 133

4.3.2.1 The Process Allocator 133
4.3.2.2 The Interrupt and Timing Process (INTIM) 149
4.3.2.3 The Storage Allocator 150
4.3.2.4 Thrash & Crash Processes 152
4.3.2.5 Other Operating System Processes 153

CHAPTER 5

THE MARK II BL MULTIPROCESSOR SYSTEM SIMULATOR 154

5.1 Introduction 154

5.2 Design Philosophy of the Simulator 156

5.3 Problems in Simulating a Multiprocessor System 157

5.4 The Simulator Program of Mark II BL System 162

5.4.1 The Main Program 165
5.4.2 The Clock Interrupt 166
5.4.3 The CPU Process 167
5.4.4 Background Process 167
5.4.5 The Taskblock 169
5.4.6 Process AP 169
5.4.7 The Storage Allocator Process 170
5.4.8 INTIM Process 171
5.4.9 The Interrupt Triplicates Process 171
5.4.10 The Process Allocator Process 174
5.4.11 Activities Durations 189

5.5 Conclusion 191

(iii)
CHAPTER 6
THE VERIFICATION, VALIDATION & USE OF THE MARK II BL SYSTEM SIMULATOR

6.1 The Verification Problem

6.1.1 The Verification Experiment

6.1.1.1 The Hypothetical DMNSC
6.1.1.2 The Trace Program
6.1.1.3 The Verification

6.2 The Validation Problem

6.2.1 The Validation Experiment

6.2.1.1 RASH Process
6.2.1.2 The Experiment
6.2.2 Discussion of Validation Experiment Results

6.3 Experiments Conducted using the Mark II BL System Simulator

6.3.1 Investigation of a New Call "FBLOCK(P)" to the Process Allocator
6.3.2 Tuning of Interrupts Handling in the Mark II BL System

CHAPTER 7
SIMULATION OF THE DIGITAL MAIN NETWORK SWITCHING CENTRE

7.1 The Digital Main Network Switching Centre

7.1.1 Introduction
7.1.2 Basic Sequence of Events in the Call Processing through the DMNSC
7.1.3 Sequencing of Messages
7.1.4 Circuit Selection
7.1.5 The Virtual Circuit Concept
7.1.6 Communication between CPS and Other Sub-systems
CHAPTER 7 continued....

7.2 The Design of the DMNSC Simulation Model

7.2.1 Introduction 268
7.2.2 The SIS Simulation 273
7.2.3 The CPS Simulation 277
7.2.4 The DSS Simulation 281
7.2.5 Telephone Calls Generation Simulation 285
7.2.6 Quantification of the Delays in the DMNSC 296

CHAPTER 8

CONCLUSIONS

8.1 Achievements 299

8.2 Suggestions for Further Work 303

APPENDICES

Appendix A: MK II BL Multiprocessor System Simulator in CSL
Appendix B: SIMULA Listing of System X Simulator Package and Cross Reference
Appendix C: A Sample of the Trace Program Output
Appendix D: A Sample of Output Reports of FBLOCK(P) Experiment
Appendix E: Derivation of the Coefficient of Correlation

REFERENCES
FIGURES

FIGURES FOR CHAPTER 1

(1.1) SIMULA Definition of PROCESS CLASS AP 9
(1.2) Hierarchical Structure of the Simulator 11
(1.3) The Open-Ended Multi-level Process-Based System X Simulator 12

FIGURES FOR CHAPTER 2

(2.1) Model Types 17
(2.2) Basic Discrete Event Simulation Technique 23
(2.3) Relationship between Event, Activity and Process with respect to Cars Arriving at a Filling Station. 17
(2.4) Simulation Executive Routine - Event Approach 36
(2.5) Simulation Executive Routine - Activity Approach 37
(2.6) Simulation Executive Routine - Process Approach 38

FIGURES FOR CHAPTER 3

(3.1) U.K. Network Hierarchy 61
(3.2) A Basic Switching Machine 65
(3.3) A Strowger Step-by-Step System 68
(3.4) A Strowger Exchange with a Register Translator 70
(3.5) Trunking of LM Ericsson ARF Crossbar Exchange 72
(3.6) A General Model of a Computer-Controlled Switching System 74
(3.7.1) Main and Stand-by System 83
(3.7.2) Dual Synchronous System 83
(3.7.3) Dual Load Sharing System 83
(3.8) Characteristic Features of Existing and New Networks 94
(3.9) System X Trunk Exchange (DMNSC) 99
(3.10.1) SPC Switching System in Real State 115
(3.10.2) SPC Switching System where Network & Periphery Substituted by Environment Simulator

FIGURES FOR CHAPTER 4

(4.1) Example of MK II BL Configuration 123
(4.2) Process States 130
(4.3) MK II BL Tasking System 132
(4.4) A Task Block State Transition Diagram 134
(4.5) A Process State Transition Diagram 136
(4.6.1) Process Allocator State Transition Diagram (Level 0) 139
(4.6.2) Process Allocator State Transition Diagram (Level 1) 141
(4.7) MK II BL Software Structure 143

FIGURES FOR CHAPTER 5

(5.1) Process Allocator Model 163
(5.2) Procedures for the Process Allocator Model 164
(5.3) CLOCKINTERRUPT Model Flow Chart 168
(5.4) CPU Model Flow Chart 168
(5.5) Background Model Flow Chart 168
(5.6) TASKBLOCK Model Flow Chart 168
(5.7) AP Model Flow Chart 168
(5.8) The Storage Allocator Model Flow Chart 168
(5.9) INTIM Model Flow Chart 168
(5.10) Interrupt Triplicates Model Flow Chart 168

FIGURES FOR CHAPTER 6

(6.1) The Modelling Process 196
(6.2) Basic Structure of Hypothetical DMNSC 202
(6.3) Enhanced DMNSC Structure + PPTABLE 204
(6.4) Simulator Structure for Validation Experiment 214
(6.5) RASH Flow Chart 216
(6.6) RASH Process Model Flow Chart

(6.7) - (6.17) Hypothetical DMNSC Processes

(6.18) Verification and Validation Process

(6.19) FBLOCK(P) Call and FBLOCK(P) Procedure

(6.20) NOFBLCK Model Flow Chart

(6.21) WITHFBLCK Model Flow Chart

(6.22) INITIATOR Process Model

FIGURES FOR CHAPTER 7

(7.1) Message Sequence Chart of an Incoming (Junction) Decadic to an Outgoing (Junction) Decadic Call through the DMNSC

(7.2) Components of the DMNSC Model and Signals Inter-changed per Call

(7.3) Circuit Assignment to SIS and CPS Replications

(7.4) Signal Interworking Sub-system (SIS) Model Flow Chart

(7.5) Call Processing Sub-system (CPS) Model Flow Chart

(7.6) DSS S/W Model

(7.7) DSS H/W Model

(7.8) CALL GENERATOR Model Flow Chart

(7.9) CALL RECORD Model

(7.10) I/C SIS H/W Model

(7.11) O/G SIS H/W Model
### TABLES FOR CHAPTER 2

| (2.1) | Categorisation of Some of the Well-known SPLS According to Approach | 44 |
| (2.2) | Types of Object Classes Available in some SPLs | 44 |
| (2.3) | Relational Concept and Object Generation Method | 48 |
| (2.4) | Scheduling Properties of SIMULA, CSL, SIMSCRIPT and GPSS | 48 |

### TABLES FOR CHAPTER 3

| (3.1) | Telephone Statistics of 12 Countries with Highest Telephones/100 of Population | 62 |
| (3.2) | Telecommunication Processors Types and Characteristics | 80 |
| (3.3) | Percentage of Maintenance Software for some of the Well-known SPC Systems | 90 |
| (3.4) | System X Family of Exchanges | 90 |
| (3.5) | Issues vs. Objects | 104 |

### TABLES FOR CHAPTER 6

| (6.1) | Times RASH Processes Executed for a Configuration of 1, 2 and 3 CPUs with 1, 2 and 3 RASH Processes. | 230 |
| (6.2) | Values for Calculation of Regression of Y on X and Correlation Coefficient | 233 |
| (6.3) | Process Allocator Overhead (us) Due to FETCH (15) JNZ(C+Z), BLOCK(P) | 245 |
| (6.4) | Process Allocator Overhead (us) Due to FBLOCK(P) | 246 |
| (6.5) | Average Process Allocators Overhead and their Percentage Reductions | 247 |
| (6.6) | Summary of Statistics for the Interrupt Handling Mechanism Modifications | 251 |

### TABLES FOR CHAPTER 7

| (7.1) | TIT of SIS | 274 |
| (7.2) | H/W Message Table for SIS | 274 |
(7.3) Incoming Tasks to SIS Identified by $G(1)$

(7.4) TIT of CPS

(7.5) Incoming Tasks to CPS Identified by $G(1)$

(7.6) Quantification of the Delays in the DMNSC

GRAPHS FOR CHAPTER 6

(6.1) Regression Line of $Y$ (Simulator) on $X$ (Real System)

(6.2) Percentage Reduction in Using FBLOCK(P) with Periodic Processes

(x)
CHAPTER 1

INTRODUCTION

With the advent of digital computers, it was proposed back in 1955 that digital computer techniques could also be used to control telecommunication switching systems (KAWA 71). The work was initiated by the Bell Telephone Laboratories of the U.S.A. and resulted in the development of the Number 1 Electronic Switching System (No.1 ESS) which went into public service in 1965. There are now over twenty different designs of computer-controlled telephone exchanges and many more in the advanced development stage (HILL 76A). An increasing number of telecommunication administrations in various countries have been introducing or planning to introduce computer-controlled switching systems into their communication networks.

Although the present electromechanical telephone switching systems, such as the step-by-step and crossbar systems, are offering a reasonable and economical service, stored program controlled (SPC) switching systems possess more overall capability than conventional systems. This overall capability manifests itself in the following features: (Section 3.3).

1. A higher level of system security
2. The ability to interwork with the older existing network.
3. Labour saving as a result of easier administration and considerably reduced maintenance effort.
4. A range of new services to the customer.
5. Space saving, power saving and higher traffic capacity.
6. Flexibility to changes and ability to make use of advances in micro-electronics.
These features have been brought about by the application of software engineering in conjunction with the development of powerful telecommunication-oriented processors. The software qualities of richness of function and versatility have made it possible for example, to expand basic call processing with wide varieties of special features for business and individual subscribers. The telecommunication industry is in a position to make the most of the opportunities presented to it by software as the technologies of communications, computers and information management rapidly converge, bringing about what is often referred to as the 'information society'. However, the complexity of software and hence stored program controlled systems called for the provision of suitable design tools and performance evaluation aids. In the domain of performance evaluation aids modelling techniques such as empirical, analytical and simulation modelling assumed a special significance recently, particularly with the development of multiprogrammed, virtual memory multiprocessor systems (SVOB 76, UNG72). The performance of such complex systems is a function of several parameters, such as the system configuration, the resource management policies of the operating system and the efficiency of the application programs. Performance evaluation is required during the design stage as well as during its operational life to aid in the design decisions, in fine-tuning and assessing the system capacity.

The most potentially powerful and flexible of the performance evaluation techniques of computer and SPC systems is simulation (CALI 67, SVOB 76). The concept of simulation is both simple and intuitively appealing, allowing the user to experiment with systems (real and proposed) where it would be impossible or impractical otherwise. Yet the application of simulation for the
analysis and evaluation of computer system performance is an important and demanding field. It is important because performance is one of the prime considerations in evaluating a computer system and demanding because it requires a deep understanding of the inner system mechanisms, both hardware and software, knowledge of the processing requirements and the workload characteristics. As a result of the demand for more powerful and flexible computing systems and better performance per unit cost, computer systems have become increasingly complex, and system performance increasingly difficult to assess. To overcome this difficulty researchers resort to a combination of simulation and measurement techniques where the workload only is simulated (CASY 77), or using a combination of analytical modelling and simulation techniques (JOUB 78), or simulation of certain aspects of a system at a time (WEAT 73) or a coarse flat-level simulation of a class of computer systems that give an approximate indication of the overall performance characteristics (UNGE 72).

In this research we attempt to develop a simulation methodology that will allow us to develop a multi-level process-based simulation of the GEC Mark II BL microprogrammed telecommunication-oriented multiprocessor system both in hardware and software as well as the members of the family of System X SPC switching systems, which are controlled by Mark II BL System, in a flexible level of detail. The aim is to provide a computer-aided design package that will enable the software engineers to assess their designs and strategies in the areas of the real-time operating system and the telephony applications software.

This is in contrast with the previous simulation work in the SPC field where the simulators were either real-time environment simulators or concentrated on one aspect or sub-system only, such as
traffic capacity, networks and control sub-system studies (ANDE 72, JAME 78, POVE 65). Real-time environment simulators have been pioneered by a subsidiary of ITT, BTM of Belgium, since 1966. The technique relies on simulating the call types and the environment of the SPC switching system such as the cross-points of the switching network, the junctors, peripheral devices and subscribers and trunks in a separate processor. The simulator processor is then run in real-time with the system processor that houses the operating system and application software. The main use of environment simulators is for testing and debugging programs, though they have been used recently for traffic capacity studies (FONT 71, GRUS 76, DGWE 78). Environment simulators proved to be effective in reducing the costs of testing and debugging of application and diagnostic software and the technique has been adopted by some other SPC and computer systems manufacturers (BECK 73, GUIT 76, CHAR 78).

The major problem with environment simulators, however, is that they cannot be used for designing and evaluating the real-time operating system. They assume the presence of a proved design of one. Their use for testing and debugging applications and diagnostic software comes at a later stage of the design process and their development is costly, considering the cost of hardware, software and the synchronization circuitry involved. The alternative of an all-software flat simulator is often difficult to verify and validate and does not lend itself to a non-simulation specialist because of the lack of resemblance between the real and simulated system. This is particularly true if the simulation approach adopted is that of the event-scheduling approach (FISH 73). Examples of such simulations are the British Telecommunications simulations of the Mark II BL system, the Digital Main Network Switching Centre and the
Pre-processor Utility (SINC 80).

Here we are suggesting a different approach to both environment and flat simulations - an all-software multi-level process simulation. We will show how and why this alternative methodology of simulating SPC systems is attractive, cost-effective and capable of simulating both the real-time operating system and the applications software. The simulator developed in this research is a powerful tool for computer aided design and evaluation of a class of SPC systems, that is the System X family of exchanges. The real-time operating system level of the simulator can be used to evaluate the present system design, tune the system and assess any proposed modifications. The applications software level can be used to evaluate a particular SPC exchange from the family of System X of exchanges.

The immediate objectives of the simulator developed in this research are:

1. To provide a tool for the designers of the real-time operating system of the System X processor utility sub-system, that is the GEC Mark II BL multiprocessor system, which would permit
   a) tests of the performance of the prototype design
   b) evaluation of possible modifications and extensions.

2. To provide a tool for evaluating the performance of applications software for the System X Digital Main Network Switching Centre (DMNSC), which uses the processor utility sub-system and other sub-systems.

The second objective is concerned with a coarser level of detail than the first, suggesting that the use of a hierarchical multi-level process simulation may be appropriate. This approach has proved to be both straightforward to implement and adaptable in its application since:
1. The amount of detail at each level in the hierarchy is kept relatively low, reducing the tasks of verification and validation at each level.

2. The simulator is structured so that it corresponds closely with the structure of the system, making it easier to understand and adapt to system design changes.

3. It is possible to exploit the confidence in the simulator, once established for one hierarchical level by using the features of this level as a base for simulating applications.

Multi-level simulation has been suggested before (ZURC 68). However, Zurcher et al used the term multi-level modelling in a different context and meaning. What they suggested was an iterative method with the concurrent existence within a single model of several representations of the system being modelled, at different levels of detail using an activity-based simulation language (a collection of Fortran sub-routines). The methodology and philosophy we suggested in this research for SPC systems simulation is primarily concerned with the inexperienced user who can build up his model of a particular SPC system from a library of models, where no more than one representation of a module exists. The objective of the Zurcher et al methodology is to provide a facility to design a computer system from the outside inwards such that each level of abstraction consists of a simulation program, constructed of a hierarchy of procedures. The simulation program is controlled by the program on the level above, which is making more global decisions based on its own variables. These next-higher level variables are an abstraction of those on the current level and, hence, are continually updated when the values of variables on the current level change.

The goal of the Zurcher methodology is to use the lowest-level model to produce the actual system, by replacing the basic algorithms in the lowest level of abstraction and the facilities which are provided by
the simulation system, such as sequencing and list processing facilities, by the real-life mechanism from which the system is to be built. This is in contrast to the objective and methodology we adopted. The objective here is to provide an integrated computer-aided design package oriented towards the community of software engineers developing the system as the main users. The application programs can be tested and debugged individually using an existing emulator program. Then the performance of individual application programs when interacting in a model of a particular System X exchange can be assessed and modified using the simulator. The transformation of the real system modules into their corresponding models in a 1-to-1 transformation process preserving the interfaces in the real system is very valuable in increasing the understanding and confidence of the software engineers in the simulator. No attempt is made to use the simulator to produce designs of the application programs from their models in the manner followed by Zurcher et al. This is because of the great volume of software in SPC systems. For example, the call processing sub-system amounts to about 100K statements. The objective of the simulator is to assess and tune the present design and not to produce new designs altogether. Applying Zurcher et al methodology would produce a simulator approaching half a million of statements! However, the top-down design implied by Zurcher et al methodology has been used by some simulation analysts in the field of SPC systems (JAME 78). Others have used a variation of their methodology where the environment is simulated in a simulation language such as SIMULA as well as the system model. The simulation of the environment and the system is run on a big computer, tested and debugged. The system model then becomes the implementation (LOVD 77, BELS 78)
The design philosophy of System X simulator called for a careful selection of a simulation system to implement the simulator. Portability was very much in mind, and led initially to the use of an available FORTRAN-based simulation language, CSL (BUXT 62). The processor sub-system, including the real-time operating system kernel and periodic application processes were simulated in CSL, an activity-based simulation language, but the resulting model did not fulfil all of the above stated requirements. From this first attempt, it was apparent that out of the three discrete event simulation approaches (event-scheduling, activity scanning and process interaction), the last approach is the most appropriate one. The notion of a process and a process instance in a simulation sense is very close to the notion of a software process as used in SPC switching systems.

SIMULA (DAHL 70) is an ALGOL-based general-purpose language with a structural concept, called the CLASS which made it very convenient for developing self-contained sub-systems for special applications. Indeed, the CLASS concept is used within SIMULA itself to transform the general-purpose language into a process-based simulation language. Another useful concept is that of a VIRTUAL PROCEDURE. Together with the CLASS prefixing concept, it is possible to simulate a complex system using a multi-level structure with many levels of refinement.

After the first attempt at simulation using CSL, an attempt was made to implement a multi-level process simulator using SIMULA. This was most successful. Here we give some demonstrations of the reasons for this. It is the CLASS prefixing concept which is most useful in realising a multi-level structure. The real-time software processes which we are modelling have some identical data structures and must be able to call for the same services of the real-time operating system. In the simulator this can be achieved by
defining a simulation process class AP that contains these common data structures and the common interfacing procedures. A SIMULA definition of AP is given in Figure (1.1).

```
PROCESS CLASS AP (PROCESS INDEX); INTEGER PROCESS INDEX;
COMMENT ** AP FOR APPLICATION PROCESS **;
BEGIN
COMMENT ** NOW DEFINE COMMON DATA STRUCTURES **;
REAL TIMELEFT;
REF(CPU)MYCPU;
REF(PROCESS ALLOCATOR) PA;
REF(HEAD)INPUTQ;
INTEGER ARRAY PROCESS DESCRIPTOR(1:J); TASK INDEX TABLE(1:K,1:L);
:
:
COMMENT ** NOW DEFINE INTERFACING PROCEDURES **;
PROCEDURE HAND ;---;
PROCEDURE FETCH(N) ;---;
:
:
PROCEDURE BLOCK(N) ;---;
END ** OF AP DEFINITION **;

FIGURE (1.1)
```

These common data structures and interfacing procedures are automatically inherited by the process class which simulates a particular application sub-system, for example the call processing sub-system, merely by prefixing the new simulation process by the identifier AP. We arrived at the choice of SIMULA after a careful review of the available discrete-event simulation approaches and languages. These approaches and languages are thoroughly discussed in Chapter 2, together with the criteria for simulation programming language selection.
The levels in the simulator are shown in Figure (1.2). At the bottom is the SIMULA system, which provides the simulation concepts and the language constructs. The GEC Mark II BL multiprocessor system model, including a detailed model of the operating system is one level higher. On the next level reside the application software simulations. Each level assumes the services of the level below. The level of application software simulation is open-ended. Models of new application software at different levels of detail may be included to form a library of simulation sub-systems as illustrated in Figure (1.3). A user can easily assemble a model of a particular exchange configuration by initiating process instances of the relevant sub-system simulations from the level below.

Chapter 3 outlines the development of telephone switching systems from the manual exchange to the present stored program controlled switching systems. Since a major impetus of this research is the study of the processor utility sub-system, telecommunications processors are analysed in greater detail including their characteristics, configurations and reliability. SPC software organisation is then introduced. The System X family of SPC exchanges is outlined revealing the complexities of the designs which highlights the need for performance evaluation tools as design aids.

The processor utility sub-system, that is the GEC Mark II BL multiprocessor system, is explained in Chapter 4, both in hardware and software in some detail, to show how the system modules interact and function. The description of the processor utility model then follows in Chapter 5, revealing the 1-to-1 transformation process and, hence, the resemblance between the system's hardware and software modules and the corresponding simulation processes. The processor utility model developed uses the multi-level process approach. The detailed modelling
FIGURE (1.2): HIERARCHICAL STRUCTURE OF THE SIMULATOR
FIGURE (1.3) : THE OPEN-ENDED MULTI-LEVEL PROCESS-BASED SYSTEM X SIMULATOR
of the process allocator, which is the central and most important part of the real-time operation system, is explained. The other module models at a coarser level of detail are also outlined.

The processor utility level is first verified and validated. The logic of the simulation is checked using a detailed tracing facility and a model of a hypothetical digital exchange. In order to validate the processor utility model, a series of experiments are conducted in a controlled environment using different configurations of the real processor utility sub-system. Use is made of operating system process instances to represent workload resource demand, that is the system is self-driven, and relevant statistics are compiled. The same series of experiments are then duplicated using simulated configurations of the processor utility. The results demonstrate the credibility of the processor utility model and inspire confidence in its findings. These aspects of verification and validation are discussed in detail in Chapter 6.

After validation, the processor utility model is used to investigate several design aspects of the real-time operating system. One example is the interrupt handling mechanism of the multiprocessor system. It is found that a reduction in overhead can be achieved by slight modifications to the original interrupt handling mechanism. Another example is the study of the feasibility of introducing a new service call to the operating system for periodic processes in order to replace two existing calls. The feasibility of such a new call is demonstrated by designing and conducting a number of experiments on the simulator. The models of System X sub-systems which are the basic building blocks of the Digital Main Network Switching Centre (DMNSC) are developed on a level above that of the processor utility model. This application level is open-ended and sub-systems models may be added to form a library of sub-systems models. Using the detailed
trace output, the model of the DMNSC is verified against the message sequence chart of telephone calls through the exchange. This model is then used to obtain delay statistics necessary for the design validation of the call processing sub-system of the exchange. The DMNSC and its simulation are the subject matter for Chapter 7.
CHAPTER 2

DISCRETE-EVENT SIMULATION APPROACHES AND LANGUAGES

2.1 DISCRETE-EVENT SIMULATION APPROACHES AND TECHNIQUES

2.1.1 Systems

In the context of simulation, by a system we mean a collection of related objects or entities, each characterised by a set of attributes assuming numerical or logical values that may themselves be related (FISH 73). Generally, every system is characterised by three features; it has boundaries, exists in an environment and is made up of sub-systems. The environment constitutes the set of surroundings in which the system is embedded, whereas the boundaries distinguish the entities in a system from those that make up the environment. The system is influenced by the environment through the input it receives from the outside world. This input is transformed by the process operating in the system, resulting in the output of the system.

With regard to the dynamic behaviour of a system, the system progresses through different states characterised by the numerical or logical values of its attributes over time. The system is said to be in a steady-state if the probability of being in some state does not vary over time. The steady-state probabilities are independent of the state in which the system started and are the limiting distribution of the transient states probabilities.
In general, the objectives in studying a system are to learn about how change in system state occur., to predict change and to control change. Particular studies are usually a combination of these objectives of varying emphasis. The ultimate objective always remains to optimise performance in some sense.

2.1.2 Models

The most general definition of a model is that it is anything that represents something else. This general definition would include such things as statues as models of particular humans, plays representing historic events, etc. A more appropriate definition is that it is a formal representation of theory or a formal account of empirical observation (FISH 73). Kiviat, on the other hand, (KIVI 67) classifies models as iconic (physical), symbolic or analogous model. Gordon (GORD 78) investigates two general categories of models; physical and mathematical, with subsequent sub-division of the models into static or dynamic, numerical or analytical as shown in Figure (2.1). Furthermore, a model can be either deterministic or stochastic. Here, we are concerned with the class of symbolic, dynamic, numerical models implemented on a digital computer i.e. computer simulation models.

Models, including simulation models are built for a number of reasons, viz:

1) It may be more costly, dangerous, time-consuming or impractical to experiment with the actual system.

2) A real system may not be available e.g. hypothetical system.
FIG (2.3): RELATIONSHIP BETWEEN EVENT, ACTIVITY AND PROCESS WITH RESPECT TO CARS ARRIVING AT A FILLING STATION.

FIG (2.1): MODEL TYPES
3) It is impossible to manipulate or control variables of interest.

4) It leads to more insight and improved understanding of the system.

5) It provides a convenient framework for testing new modifications and proposals.

The problem at hand determines to a great extent the type of model and solution adopted. Analytical solutions are more attractive than simulations, if they can achieve the objective. This is because analytical models, once developed and verified, give answers to a variety of input parameters, with very little additional effort. However, except for the simplest cases, the derivation of an analytical model for a complex system proves intractable mathematically, except through a series of simplifying assumptions which may affect the credibility of the model itself or limit its use. One such common approximation is that of statistical independence of the inter-arrival and service times, and hence the use of a Poisson input process and exponential service times. The use of this 'memoryless' exponential distribution greatly simplifies the mathematics in analytical models using networks of queues. A further simplifying assumption is that the system is stationary (has reached steady state or statistical equilibrium) thus the time derivatives vanish.

In spite of the simplifying assumptions, queuing models have been used successfully to study computer and communication systems (GRAN 64, FRAN 74, KUCH 75). They are best suited to give a general insight into the system dynamic behaviour, to identify the bottle necks and in the study of sub-systems. Sometimes, it is more advantageous to use them in conjunction with other methods such as system simulation (JOUB 78, UNGU 75).
2.1.3 System Simulation

System Simulation may be broadly defined as the act of representing a system by a symbolic model which can easily be manipulated to produce numerical results. More specifically, Krasnow (KRAS 67) defines system simulation as the activity comprising the description of a system by constructing a model, the description of an experiment to be conducted with the model, the carrying out of the experiment and the analysis of the results.

The range of system simulation models is fairly broad; they range from using a prototype of the system as a model (identity simulation) to a computer model. In this study we are concerned with computer models only. These may be classified as analogue and digital computer models. In analogue simulation models, analogue computers are used to simulate a set of differential equations modelling the behavioural characteristics of a system. Digital simulation models may be implemented in hardware or in software (HART 75). Hardware simulators are composed of special-purpose computers (or equipment) and undetailed programming. They are characterised by having high speed and parallel operation. Software simulation models on the other hand, utilise detailed computer programs to model a system in a general-purpose computer. They are characterised by low speed and sequential operation. The simulated time advancement may follow either the epoch-by-epoch (or equal increment) approach or the event-by-event approach for both hardware and software simulators. In the former, the simulated time clock is incremented by a fixed amount, $\Delta T$, every time and the system updated at the epochs where events occur, whereas in the latter, emphasis lies on updating the overall
simulation only on the occurrence of an event. In this latter sense, simulation may be viewed as the activity concerned with the generation and cancellation of event notices or records, transformational rule selection and clock maintenance. Time intervals associated with events are normally drawn from appropriate statistical distributions. Due to their relative importance within the simulation methodology, the time advancement mechanisms will be considered in greater details in the next two sections. Thus, software digital simulation models are based on representational descriptions of entity interactions and state variable transformations that must be specified in the computer program (model) which, when executed, traces or mimics the dynamic behaviour of the modelled system. The program is then the realisation of the model. Furthermore, the specification of the program must be made within the confines of the abstractions or concepts supported by the computer language selected to implement the model. Thus the language choice greatly influences the way the modelled system is viewed.

The essential features of all types of computer simulation are the computers, operation rules, mathematical functions and probability distributions and it is specifically suited to systems where the relationships between the key variables cannot be expressed analytically, or where the major attributes of the system are characterised by probability distributions or stochastic processes (REIT 71). Computer simulation offers a scientific approach to system investigation. Although systems differ in their complexities and characteristics, the ingredients of this approach, namely, model building, computer science and statistical techniques are applicable in the study of any system employing this approach.
Many advantages accrue to computer simulation in comparison to other modelling techniques. It can compress time, so that many years of activity may be simulated in minutes or even seconds. This will enable a system analyst to compare long-term behavioural characteristics of alternative designs or operation rules. It can also 'dilute' or expand time. Here, the system is simulated at a finer level of detail, e.g. a time grain of a microsecond say, to enable the components interactions to be closely observed which cannot be done in real time. This research study is one such example, especially the study of the process allocator (Chapter 5). The simulation can also freely identify and control the sources of variation in the model by explicitly specifying the sources of variation and degree of variation due to each, which is not possible when experimenting with a real system. This is particularly important if the statistical analysis of the relationship between input and output factors in an experiment is to be performed.

If programmed appropriately, a computer model may be stopped to investigate the results of a run so far and re-start it again without loss of continuity. Provided that the same seeds for the pseudo-random number generators are used, the dynamic behaviour of a system can be re-produced again and again for purposes of debugging and fault diagnosis. This is difficult to achieve in a real situation. However, a price has to be paid in the form of increased computing cost and human effort. This is particularly true for detailed simulations. Another price is the necessity to apply statistical techniques to the analysis of the simulation output, since simulation is in essence a sampling experiment. In this respect, it should be emphasised that simulation is essentially an experimental technique. The immediate purpose
of an experiment is to observe the behaviour of a given model within a given environment. It is also recognised from the outset that digital computers are discrete devices and that a digital simulation model is a discrete approximation to a given system. Thus continuous changes in the real system are represented by a series of discrete changes in the model i.e. events, and such a model is called a 'discrete event model'.

In contrast to continuous simulation, where the system as a whole is represented by a set of differential equations, the individual events of a discrete-event model are often specified in great detail. The apparent realism of the resulting discrete-event model accounts for some of the charm and fascination of the 'art' of discrete event simulation.

The models developed in this research study are of a discrete-event type. The occurrence of an event is represented by a change in a component's attribute value. Since the components or entities states remain constant between events, there is no need to account for this inactivity in the model. Hence, all modern computer simulation programming languages use the next-event approach to time advance i.e. at a time corresponding to a particular event, all relevant state changes will be made. Simulated time is then advanced to the time of the next event and the above process repeated and so on. This cyclic process is depicted in Figure (2.2).

In this way a simulation is able to skip over the inactive time whose passage in the real world people are forced to endure. The efficiency of this event-by-event method over the epoch-by-epoch one is also evident, particularly where the epoch is smaller than the average length of event inter-arrival time, which is usually the case.
FIG (2.2) BASIC DISCRETE EVENT SIMULATION TECHNIQUE
One can identify two structures that play significant roles in discrete-event modelling. The mathematical relationships between model variables or entities' attributes comprise one structure. e.g. if \( L \) is a queue length, then \( L \) becomes equal to \( L + 1 \) or \( L - 1 \) according to whether a customer arrives or departs. For some systems, the specification of the mathematical relationships for a system serves to describe completely the way in which state changes occur, e.g. a model of a natural economy system (FISH 73). The second structure comprises the logical relationships. Here, logical operating rules are established, logical conditions are checked and actions are taken according to the established operating rules.

The concepts of simulated time passage and relational structures are central to any discrete-event modelling system. Different modelling approaches account for these concepts in varying ways. In the following section, we will try to identify those approaches and the characteristics pertinent to each.

2.1.4 Discrete-Event Simulation Approaches

2.1.4.1 The Approaches

Discrete-event modelling approaches are built around the concepts of event, activity and process. An event, as mentioned before, signals a change in the state of an entity. An activity may be defined as the collection of actions that transform the state of an entity. It starts with an event and ends with an event. Similar concepts are used in other areas of software. One such area is the specification and description languages (KAWA 71, GALE 75).
In particular, with reference to the specification and description language adopted by CCITT (Consultative Committee of International Telephony and Telegraphy) (SDL) for SPc switching systems specification and description (KAWA 71, GALE 75, GERR 74, CCIT 77 etc.), an event is an input and an activity is a transition between two stable states in a finite-state machine. On the other hand, a process may be looked at as a sequence of events ordered in time. Figure (2.3) depicts the relationship between an event, an activity and a process. More specifically, from a simulation point of view, a process may be defined as a dynamic entity, a singularly-occurring instance of execution of a set of logically-related activities (MACD 73A). Processes comprised of like sets of activities are considered to belong to the same class.

Based on the three concepts of event, activity and process, three alternative approaches to discrete-event modelling exist (KIVI 67); namely, the event-scheduling, the activity-scanning and the process-interaction approaches. The evolution of the three approaches is related to the development of computer simulation languages. For example, SIMSCRIPT (KIVI 69A et al) and GASP (PRIT 69) are based on the event-scheduling approach (a recent version of SIMSCRIPT II.5 supports the process approach), CSL (BUXT 66) uses the activity-scanning approach and SIMULA (DAHL 70), GPSS (GORD 78) and ASPOL (MACD 73A) the process-interaction approach.

In the event scheduling approach, the events that may occur in the system are identified first. Some of the events are conditional on the occurrence of other events. Future events records are listed in time-order sequence in a calendar of events. Each record contains the future simulated time for the occurrence
of the event and a reference to the code block representing that event. The actions initiated by the occurrence of an event are identified in the event flow chart. This includes scheduling of other events, in the form of filing and/or removal of event records from the calendar. Selection of the next event is cited as the last instruction. When this instruction is encountered, control passes to the simulation control program, which searches the list of filed records to find the record with the earliest scheduled time. Simulated time, kept by a monotonically-increasing global clock, is then advanced to this scheduled time and control is passed again to the code block representing the new event.

One disadvantage of this approach is that the inclusion of an event record in the calendar ordered by future time occurrence and the subsequent search for an appropriate event is a rather time consuming process, especially where the event density is high. To speed up this operation, some simulation systems sub-divide the calendar by the type of event; the actual event list scanned by the timing mechanism is only the 'best-of-show' of sub-calendars of each event. It is claimed that this event approach is more suited to single-resource and logically-simple simulations, where a one-to-one correspondence between events and activities exists (LASK 65). Single-resource activities are those requiring the availability of no more than one entity other than those already present. Here, the event approach is efficient in the sense that each activity is attempted only when it is logically possible for it to be performed.

However, in multi-resource and logically-complex situations, where the relationship between event and activity is many-one or many-many, the programming of such a situation in an event approach is rather complex.
An example of such a situation is the operation of a shipping port. Here, for example, the activity of berthing a ship will involve the presence of a pilot, tug, berth and a ship, each of which is an event in its own right. It is worthwhile noting that, in the event approach, activities are not explicitly recognised. A more convenient method for the above situation is to put into a sub-routine the programme corresponding to transforming the world-state (which constitutes the activity) and stating exactly the conditions of entry to the sub-routine.

Such a sub-routine is the essence of the activity scanning approach, where each activity is composed of a test part at the beginning and a body. The emphasis here is on the review of all the activities in a simulation model to determine by performing the tests on the test parts, which can be executed each time an event occurs. In this approach, events are only implicitly recognised. To keep track of various events and advance simulated time to that of the next event, the notion of 'time-cells' is introduced. These are storage locations associated with certain entities and holding the relative time at which the entity will or has become available to participate in activities. Relative time means with reference to 'now' or relative to 'now', e.g. a positive time-cell value indicates the time interval that is to elapse from now for the event to occur, whereas a negative time-cell value indicates how long ago the event associated with the entity has occurred. Here, time advancement is more involved than just advancing a master clock to the next event, a characteristic of the next event approach. Rather, all the time-cells have to be searched to determine which of them stores the least value, then subtracting that value from all the time-cells, a process which is relatively time consuming. However, this disadvantage with respect to the event approach is off-set by the time advantage when causing an event. To cause an event $T$ time units from now, say, will only require setting the appropriate time-cell to the value $T$ and does not involve a search through a calendar to insert an event record appropriately. Thus, from this point of
view, there is little to choose between the two approaches, except in cases where events are cancelled or have their times altered. Whereas, in the activity approach, the appropriate time-cell value only requires to be altered, the event approach would require a double search through the calendar.

Thus, one can say that the important advantage of the event approach lies in the execution efficiency it achieves by preserving the knowledge of which entities are involved in an event and using this information to ignore, in the subsequent event-activities phase, those activities whose entry tests do not include the presence of the appropriate entities. In other words, the fact that an event record contains a reference to the event-code block alleviates the need for scanning and testing all event code blocks to determine which ones should be executed, as in the activity approach. The advantage of the activity approach, on the other hand, is that the events-activities inter-relationships need not be explicitly specified, resulting in the simplicity of formulation of a multi-resource or logically-complex systems models.

To improve on the execution efficiency of the activity approach, the Disaggregated Activities List approach is proposed (LASK 65). Instead of having one activity list only, that list is sub-divided into sub-lists according to the entities whose events may result in these activities and according to activities whose execution may result in further activities. Thus, at any entry to the activities phase, the information regarding the entities involved in a particular event from which activities can now result is not lost and is used to determine which sub-lists should be entered. This alleviates the need to scan the whole of the activity list and hence increases
the execution efficiency of the simulation system. Thus, the event-
scheduling approach contrasts with the activity scanning approach
in that a detailed list of scheduled events with their occurrence
times is maintained. The number of events grows with the
growing number of activities, thus increasing the computer time
required to create event records, file them in the event list,
select them for execution and destroy them once executed.

The activity-scanning approach, however, substitutes less time-
consuming logical checking in the model at each time advance, for
the required event scheduling steps. From the foregoing, it is
evident that both of the approaches have their advantages and
disadvantages. However, it is interesting to note that the event
scheduling approach is the most widely use, thanks to languages such
as SIMSCRIPT, which is event-based and widely available, in contrast
to CSL, say, which is activity-based.

A third approach exists, namely, the process-interaction approach
which is believed to combine the sophisticated event scheduling
of the event-scheduling approach with the concise modelling power
of the activity-scanning approach. From a simulation point of
view a process is a collection of events that describes the total
history of a system's component through the system. The process
interaction approach is felt to be most natural in a variety
of modelling situations (BIRT 73). The mode of describing actions
is serial here and this is felt most natural when considering a single
system component. The process approach encompasses, within a
single framework, the features of duration, parallelism and interaction
which characterise components of dynamic systems.
Thus, in this approach, the simulation analyst first identifies the system components and then for each component examines the system behaviour from the point of view of that component and hence develops a scenario of behaviour for that component. Hence, a component’s life pattern is described by a scenario which includes the transformational rules which that component or entity applies and its possible need to request or react to application of transformational rules which are part of the scenario descriptions of other components (FRAN 77). Therefore, a system simulation program is made up of scenario descriptions of possibly several components and their interactions. This approach to a component within a model permits localisation of statistics gathering, the attributes necessary to define the state of the component at any time and the protocol for interaction with other objects in a single structural entity, thus resulting in a modularly structured simulator. The scenario descriptions are known as processes and a model, after initialisation, is thus composed of process instances of components’ scenario descriptions which co-exist in parallel and progress, in simulated time, in a piece-meal fashion through their scenarios or life patterns independently, yet interacting with other instances.

A more formal definition of a process is due to MacDougall (MACD 73A):

"A process is a dynamic entity. A singularly occurring instance of execution of a set of logically related activities. Processes composed of like sets of activities are considered to belong to the same class. At any point in time a number of such processes may exist in a system model in varying stages of execution. Each is an instance of object of its class, uniquely identified among other members of that class by certain attributes. The behaviour of processes of the same class may be described by a single set of rules describing the activities of all processes from that class (the action statements)"
together with a set of attribute(s) values for each of
the existing processes of that class (activation record)."

Thus each process has its own copy of the actions and data
structures (attributes) of the class to which it belongs together
with a local sequence control or a local pointer which points to
the statement in its copy being executed. In the process approach,
time may elapse in the model, in contrast to other approaches
which provide snapshots only. As processes in a model
progress in a quasi-parallel or concurrent fashion, process
interaction or synchronization has to be taken care of to resolve
any conflicts of overlapping processes.

Processes can delay themselves for a specified period of time
by using scheduling constructs such as 'DELAY' or 'HOLD'. The
effect of these on the simulation control program is to suspend
a process execution and file an event notice indicating the
future reactivation time. The process's 'little green finger'
(Local Sequence Control or LSC) then points to the reactivation point,
i.e. the statement following the scheduling construct where
execution will continue after the last abandonment. A process
may also suspend its execution for an unspecified period of time,
using scheduling constructs such as 'WAIT' where the simulation
executive program removes the event notice or record associated
with that process from the event list. Here, a process relies
on another process to schedule an event notice for its
reactivation using scheduling statement such as 'ACTIVATE PROCESS X'.
Thus, for a model based on the process-interaction approach,
we require processes and synchronization primitives for the
manipulation of processes' event notices and hence activation
and reactivation times.
Process synchronization implies a constraint on the order of process operations in time. Three types of synchronization mechanisms are required to co-ordinate and manage processes' interactions to simulate a system behaviour. Firstly, the mutual-exclusion mechanism or primitive is needed for competitive process interaction to manage the allocation of single-capacity resources exclusively to a process. An example is the competition of jobs or programs to acquire a CPU or customers to acquire a service station. When the resource is relinquished, it is allocated to one of the waiting customers according to some specified discipline.

Secondly, a producer/consumer synchronization primitive is required to manage co-operative processes interactions. This is to avoid a situation where either a producer process is producing units at such a rate that the consumer process can not possibly cope, or the consumer process is trying to consume non-existent units. Thus, a consumer process must stop and await the arrival of additional units if it runs out of units and the producer process must then re-start the consumer process after producing one or more such units and then suspend itself. Its activation is then the responsibility of the consumer process when it runs out of units. Thirdly, some processes may require the existence of a partial or complete system state before their execution may be continued. The simulation system must allow such processes to check for the conditions of reactivation at the occurrence of appropriate events. Such a synchronization is referred to as mass retries (FRAN 77). For example, a process whose actions are suspended in a conditional wait using a scheduling construct such as 'WAITUNTIL' must be allowed to check the condition of its reactivation at the occurrence of favourable events. Some simulation systems vest this responsibility with the simulation control program or executive at the price of increasing run time. This is because the simulation executive has to activate a monitor after the occurrence of each event. The monitor will then activate processes suspended in a 'WAITUNTIL' and filed in a conditional wait list, to test for themselves whether they can proceed or not (HILL 76, VAUC 73). Other simulation systems vest the responsibility wholly with the user e.g. SIMULA 67 (DÄHL 70) or partly with the user who signals when a possible 'WAITUNTIL' condition occurs e.g. DEMOS (BIRT 79). The latter system attempts to strike a balance between run time efficiency and user convenience.
It is worthwhile noting that the concept of a process as expressed here is similar to that employed in operating systems design (HANS 73, HANS 77). Hence, a simulation system based on the process approach was felt a natural choice for simulating SPC system controlled by special-purpose computers using inter-communicating operating systems and application processes. More about the simulation language selection is given later on in the Chapter.

2.1.4.2 The Simulation Executive Program

A simulation system provides a world view to system modelling. It provides a conceptual framework for precise thinking (DAHL 68). The manner in which the framework accounts for the passage of simulated time is central to the framework since time and its representation is the essence of discrete-event simulation (KIVI 69A). This is the extra dimension to be accounted for in writing discrete-event models programs as opposed to ordinary type of programming activities.

The heart of every simulation system is a simulation executive program. It is referred to alternatively, as a simulation-control program, a time-control program, a timing mechanism and the like. In all cases, it always performs the same functions: to advance simulated time and to select a code segment that performs a specified simulation activity. Regardless of the simulation approach, the basic structure of the simulation executive program is always the same (Figure (2.2)).

A simulation system may be thought of as a hierarchical three-level structure. At the top level is situated the simulation executive program. The intermediate level is composed of simulation-oriented routines such as events, activities and processes. The bottom level is composed of routines that perform basic housekeeping functions, such as input/output, computation of mathematical functions, generation of random variates and the like.
With reference to Figure (2.2), the number and structure of the code blocks will depend on the sequencing approach adopted i.e. event, activity or process. For the event-scheduling approach, the code blocks are known as event routines. These are the specifications of the transformational rules or activity that define or accompany each event. Each code block contains both a test at the top and action statements which are a description of the situations that can take place whenever an event occurs (KIVI 67). For the activity approach, the code blocks again consist of two parts, a preamble or test part at the top and an action part. The test part may be a complex logical test involving time-cells values and different attributes values. The test is first performed and, if positive, then the activity defined by the action part is executed. For the process approach, the code blocks are the simulation processes as defined before. The scheduling statements within a process body such as WAIT, DELAY and WAITUNTIL, are provided by the simulation system to enable the user to interact with the simulation executive program. DELAY and HOLD type of scheduling statements are referred to sometimes as imperative sequencing statements as they are explicit in stating what should happen and at what time (DAHL 68) e.g.

\[ \text{PROCESS } p \text{ HOLD( } X \text{ UNITS OF TIME)} \]  

On the other hand, it may not be possible, sometimes, to predict in advance the time at which a given event should take place. This is due to the inter-dependence of system components or processes. In such instances, the scheduling statement WAITUNTIL is used to interact with the simulation executive to take care of the process in such a situation, e.g. 

\[ \text{PROCESS } p \text{ WAITUNTIL( EVENT S OCCURS)} \]
This conditional type of scheduling is referred to as interrogative scheduling.

The simulation executive programs for the event, activity and process approaches are depicted in more detail in Figures (2.4), (2.5) and (2.6) respectively. Note that the dotted lines imply transfer of control between the simulation executive program and the appropriate code block. The flow charts reflect the basic macro-level structure only, omitting any details that might otherwise obscure this basic structure. In Figure (2.4), after the execution of an event routine, its event notice is either destroyed or re-scheduled, possibly together with other new event notices. The conditional event list is then checked to see if any of the pending conditional events could be executed. In Figure (2.5), the RECYCLE flag is set only when certain activities are executed whose execution will result in the execution of some other conditional activities. The conditional event list for processes in a 'WAITUNTIL' in Figure (2.6) is scanned every time after each event. This is an over-simplification of the possible algorithms to deal with such a situation (VAUC 73). The automatic creation and destruction of event notices is implied in each of the above flowcharts of Figure (2.4) and Figure (2.6).

2.1.4.3 An Example and a Summary

The differences between the nature of the code blocks of Figure (2.2) when expressed in different approaches can best be demonstrated by a simple example. Consider the simple case of simulating a petrol filling station with a fixed number of pumps and attendants. Cars arrive randomly and service is offered when
FIG(2.4): SIMULATION EXECUTIVE ROUTINE - EVENT APPROACH

NOTE: --- > DOTTED LINE ARROWS IMPLY TRANSFER OF CONTROL BETWEEN EXECUTIVE ROUTINE AND CODE BLOCKS.
FIG (2.6) : SIMULATION EXECUTIVE ROUTINE - ACTIVITY APPROACH

1. **Simulation Executive**
2. **Scan Entities Time Cells and Determine Least Value One**
3. **Subtract This Value from All Time Cells**
4. **Scan All Activities and Execute Those Whose Tests Are Positive**
5. **Recycle**
6. **End of Simulation**
7. **Output Results**
8. **Stop**
Fig. (2.6): Simulation Executive Routine - Process Approach
both an attendant and a pump are idle (the station is not self-service). Simulating this situation in the event approach would require writing three code blocks; one for the attendant, one for the pump and one for the car. Each block would contain test and action statements. For example, the code block or the event routine of the event 'attendant becomes idle' might look like:

Test Part: If a car is available AND If a pump is idle then do Action 1 ELSE do Action 2.

Action 1: Engage attendant Engage pump determine time attendant will be engaged determine time pump will be engaged schedule 'attendant becomes idle' event schedule 'pump becomes idle' event return to executive program.

Action 2: Put attendant in idle state return to executive program.

Expressed in the activity approach, we have to specify the conditions for the state of a service and the actions taken when such conditions are satisfied. One possible description of this activity is as follows:

Test Part: If a car is available AND If a pump is idle AND If an attendant is idle THEN DO Activity Body ELSE go to next activity or if last one to executive program.

Activity Body: Engage attendant Engage pump determine time attendant will be engaged determine time pump will be engaged Set attendant-clock to time attendant will be idle Set pump-clock to time pump will be idle Go to next activity or if last activity to simulation executive program.

Using the process interaction approach, the above system dynamics behaviour may be expressed by a scenario from the point of view of either the car, the attendant or the pump. For example, from the point of view of a car, the service process might look something like
Process Car: Generate a new car after a random
inter-arrival time.
Acquire an attendant resource
Acquire a pump resource
Hold the attendant resource for appropriate period
Hold pump resource for appropriate period
Release attendant resource
Release pump resource
Quit filling station.

Modelling the situation in the event approach requires three event
routines; one for each of the three events, pump available,
attendant available and a car arrival. Using the activity approach,
on the other hand, only one activity is required. However, while
an event routine is only executed when its event occurs, the
activity test preamble is checked with every time advance. Thus,
the event approach has a higher execution efficiency and a lower
storage efficiency, while the activity approach has a lower
execution efficiency and a higher storage efficiency. On the
other hand, the process approach combines both execution and
storage efficiencies, by making use of both imperative and
interrogative scheduling characteristics of event and activity
approaches respectively.

However, each of the approaches has distinct advantages in certain
modelling situation for a particular class of problems and each
of the three approaches can certainly do whatever the other two can
do. From what has been said, it is clear that a simulation
system based on the event approach gives the simulation analyst precise
control over the execution of the programs. That based on the activity
approach simplifies the modelling of multi-resource systems by
allowing conditional statements of resource availability to be
specified in one place. The process approach helps to reduce the
number of 'overhead' statements a programmer has to write, since he
or she can combine many event routines in one process description. Moreover, the overall flow of a system is clear, as all the logic of a system component is contained in one module rather than several.

To sum up we list characteristics of the three approaches as summarised by Hills (HILL 73A):

**EVENT:**
- Not easy to specify in complex situations
- Easy to explain once specified
- Hard to modify since control statements are scattered.
- Efficient in execution but consumes more storage than activity model.
- Suitable for models which are:
  a) Well-defined in real life
  b) Event/activity relationship is 1:1 e.g. single resource
  c) Where the real situation is defined in terms of events or break points.
- Examples: Air-traffic control, military models and communication systems.

**ACTIVITY:**
- Easy to specify
- Easy to explain
- Control statements grouped together
- Lower execution efficiency than event models.
- Suitable for models which are:
  a) Complex or ill-defined
  b) Where objective is to study changes in control
  c) Where the natural plant is thought of in terms of jobs to be done.
- Examples: Workshops, Warehouses and ports.

**PROCESS:**
- Hard to specify in some cases
- Efficient and compact
- Suitable when:
  a) Efficiency is important
  b) A well-defined simulation or previously specified model (in activity or event) exists.
  c) The real system is naturally defined in terms of processes or easily identifiable objects.
- Examples: Production plants, Refineries, Communication and computer systems.
So much so for the different simulation approaches. We will now turn our attention to the requirements demanded by simulation users from special-purpose simulation languages that implement those approaches and concepts and how those languages go about providing them.

2.2 DISCRETE-EVENT SIMULATION PROGRAMMING LANGUAGES

2.2.1 Introduction

Discrete-event simulation models are particularly difficult to program and debug in a high-level language such as FORTRAN or ALGOL. This is because they entail creation, filing and destruction of records, searching of lists, generation of random variates, data collection, analysis and display and model initialisation. The commonality of these and other features has prompted the development of special-purpose simulation languages to reduce the modelling and programming effort.

The first suggestion for a special programming language to aid in simulation model building is due to TOCKER and dates back to 1960 (TOCK 60). Since then, a proliferation of simulation programming languages (SPLs for short) has resulted. Recently, it has been reported that the number of discrete-event SPLs alone is as high as seventy (SIMU 79). This proliferation of SPLs stems partly from a genuine need to provide SPLs oriented towards particular classes of problems or application areas. However, for the majority of SPLs it was that the 'pride of authorship kept many going when common sense should indicate that other available languages would fulfil the needs' (KAY 72).
In any case, SPLs help to provide a conceptual framework for system components identification, together with the required notation for dynamic behaviour description and model development aids. In the following discussion, we will confine ourselves mainly to a small sub-set of SPLs from the whole universe of SPLs. This sub-set will include four of the most widely known SPLs; two originating in the USA (GPSS, SIMSCRIPT) and two in Europe (CSL, SIMULA). We will use them as a platform for our discussion of SPLs features.

Within the SPLs, some are at a higher level than others. For example, SIMULA and SIMSCRIPT are very powerful general-purpose languages that can define complex data structures, handle lists and allocate memory dynamically. Yet they contain only a relatively small number of special features directed towards simulation, e.g. simulated system time and imperative scheduling. On the other hand, languages such as GPSS (GORD 78) and SOL (KNUT 64) are at a higher level because they have predefined objects (e.g. facilities and storages), interrogative sequencing and automatic reporting. However, SIMULA and SIMSCRIPT generate more efficient code and are more general and flexible in their application. Moreover, experience has shown that these extra 'niceties' can still be developed in the language source code to augment the language simulation facilities already existing and, hence, reduce model development costs further e.g. SIMON (HILL 76), DEMOS (BIRT 79).

Some SPL experts distinguish between a simulation language and a simulation system (KIVI 67). While a language provides certain concepts and a framework for model building, a simulation system is the implementation of that language on a particular computer or class of computers together with the model development aids. Others propose classifying SPLs differently e.g. Tocker (TOCK 65) suggests classifying SPLs according both to the approach adopted and the dominant type of entity in the language i.e. whether active or passive e.g. machine or material. According to this classification, SPL development in USA was essentially materially-based extending FORTRAN e.g. SIMSCRIPT, GPSS, GASP etc. and Europe machine-based extending ALGOL e.g. SIMULA. In our discussion, however, we will stick to the classification of SPLs by the approach to modelling. This classification is depicted for various SPLs in Table (2.1).
<table>
<thead>
<tr>
<th>Event Oriented Languages</th>
<th>Activity Oriented Languages</th>
<th>Process Oriented Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>GASP</td>
<td>CSL</td>
<td>SIMULA</td>
</tr>
<tr>
<td>SIMSCRIPT</td>
<td>ECSL</td>
<td>GPSS</td>
</tr>
<tr>
<td>TELESIM</td>
<td>SILLY</td>
<td>SOL</td>
</tr>
<tr>
<td>SIMPAC</td>
<td>AS</td>
<td>ASPOL</td>
</tr>
</tbody>
</table>

**TABLE (2.1): Categorisation of Some of the Well-known SPLs According to Approach**

<table>
<thead>
<tr>
<th>Type of Object Class</th>
<th>SIMULA</th>
<th>CSL</th>
<th>SIMSCRIPT</th>
<th>GPSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predefined Permanent</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>User defined Permanent</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Predefined Transient</td>
<td>(no)</td>
<td>(no)</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>User defined Transient</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

**TABLE (2.2): Types of Object Classes Available in Some SPLs**

REF: (DAHL 68)
The functions that SPLs are expected to provide are as follows:

- a) Static Representation
- b) Dynamic Representation
- c) Simulation Support

In the following, we will turn our attention to each of these functions in greater detail.

### 2.2.2 Static Representation

Static representation is concerned with the status of a system at a given point in time. A system is thought of in terms of the objects it contains. The system status is determined by the status of each object and their inter-relationships. An SPL has to provide the tools for the definition of the classes of objects within a system. The class concept simplifies to a great extent the description of a system, because it limits the great number of objects possible in a system to a handful of classes of objects. The definition of a class serves as a template description for all objects belonging to that class.

Objects are represented as data structures of different classes. These are records of the objects attributes. A simulation operates on these records as simulated time progresses. An SPL has to enable the definition of attributes that can both describe and differentiate between objects of the same class. Some of these attributes are system defined, i.e. defined by the simulation system at the time of object creation. The system-defined attributes are either hidden (not accessible to the user) or protected (only their values can be accessed). The remainder of object attributes are user-defined, and these can assume numerical or logical values, eg by assignment. The collective values of an object attributes determine the status of that object. For example, in a port simulation, we may identify the classes of objects as SHIP (with attributes SIZE, TIME OF ARRIVAL, CARGO etc.), CRANE (with attributes CAPACITY, POSITION, IDLE), BERTH (with attribute VACANT) etc. All ships belong to the class SHIP and individual ships are identified by different values of attributes.
Objects can either be temporary or permanent and an SPL is expected to provide for the generation of both types. In terms of job-shop models, which are essentially networks of inter-connected elements of M/G/1 queues (M/G/1 indicates Markovian or stochastic arrival, General service time distribution and one server), transient objects are the units of flow e.g. a customer in a supermarket or a car in a filling station. They are generated, pass through the system and fade away. When a transient object 'dies' or fades away, the storage allocated to it is reclaimed and allocated dynamically for the generation of a new transient object. Permanent objects, on the other hand, are fixed in their number throughout a simulation run. Mostly they represent the resources in a model which give service to or are controlled by transient objects. Such resources are either time-shared, e.g. a CPU, a machine etc. or space-shared e.g. a store. They are acquired by statements such as COOPT SEIZE and ENTER and relinquished by statements such as RETURN and LEAVE e.g. GPSS (GORD 78), SIMON (HILL 76).

In GPSS: a transient object known as a transaction, is generated by the statement:

```
GENERATE <List of Arguments>
```

with specified priority and user-defined attributes.

In SIMSCRIPT: CREATE <class> CALLED <variable>

an object of 'class' is created and referenced by the reference variable <variable> whose value is the object being referenced

The statement

```
DESTROY <class> CALLED <variable>
```

destroys the transient object referenced by <variable>.

In SIMULA: PROCESS CLASS <class> (<formal parameter list>)

```
NEW <class> (<actual parameter list>)
```

generates a process instance of class name <class> and the value of the expression is a reference to the process instance generated. This reference value can be stored in a number of ways viz:
a) by assignment to a reference variable e.g.

\[ \text{REF(CAR)A; A:- NEW CAR} \]

b) by entering a process instance into a queue e.g.

\[ \text{NEW CAR. INTO(WAITQ)} \]

c) by scheduling an event for the process instance e.g.

\[ \text{ACTIVATE NEW CAR} \]

No delete or destroy statement is used in SIMULA. A process remains in the system as long as it is referenced. A 'reference count' which is a hidden attribute of a process is updated each time a reference to a process is stored or deleted. A process instance is automatically deleted when its reference count becomes zero. If a referenced process instance terminates, i.e. exhausts its actions and passes through the last END statement, it will still remain in the system, but only as a data structure whose attributes are accessible.

In CSL: Only a fixed number of objects is allowed to be generated and remain throughout the simulation run, i.e. only static memory allocation e.g.

\[ \text{CLASS CAR.100} \]

will define and create 100 objects of class CAR. Table (2.2) shows the types of object classes available in some SPLs and Table (2.3) summarises object generation and related concepts.

Aside from object generation, an SPL must enable relating objects to one another and to their common environment. This is achieved through relational mechanisms, such as sets, queues and lists. For example, in CSL, we may define a set called READY to contain all ships ready to depart from the port. Selection of a particular ship for departure (e.g. of particular tonnage) is achieved by searching through the set. As a matter of fact, list processing is a dominant feature in SPLs due to the need for flexibility and efficiency in data manipulation and search.
<table>
<thead>
<tr>
<th>CONCEPT</th>
<th>SIMULA</th>
<th>CSL,</th>
<th>SIMSCRIPT</th>
<th>GPSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational Concept</td>
<td>HEAD</td>
<td>SET</td>
<td>SET</td>
<td>User Chain Group</td>
</tr>
<tr>
<td>Example</td>
<td>REF(HEAD)Q MAN. INTO(Q) (Man Queued in (Q))</td>
<td>SET OCEAN SHIP 1 INTO OCEAN (Ship 1 moved to Set Ocean)</td>
<td>FILE MAN FIRST IN SET(I) (man inserted into set(I) as first mem.)</td>
<td>LINK I, FIFO (Current Transaction Put first in chain I)</td>
</tr>
<tr>
<td>Object Generation</td>
<td>Generate a new process when required (Dynamic)</td>
<td>No Dynamic Object Generation (Static)</td>
<td>Generate a new entity whenever required (Dynamic)</td>
<td>Generate a new transaction when ever required (Dynamic)</td>
</tr>
<tr>
<td>Example</td>
<td>QE2:- New Ship</td>
<td>-</td>
<td>CREATE A DOG CALLED SNOOPY</td>
<td>GENERATE 6,2</td>
</tr>
</tbody>
</table>

**TABLE (2.3) : Relational Concept and Object Generation Method**

<table>
<thead>
<tr>
<th>Scheduling Type</th>
<th>SIMULA</th>
<th>CSL</th>
<th>SIMSCRIPT</th>
<th>GPSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imperative</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Interrogative</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>

**TABLE(2.4) : Scheduling Properties of SIMULA, CSL, SIMSCRIPT & GPSS**

48
2.2.3 Dynamic Representation

This is concerned with the system changes of state as a function of simulated time. The concept of simulated time and its passage with the consequent changes in system state is implemented through a simulation executive program, provided by the simulation system. The structures of the executive programs for the three simulation approaches have been discussed in (2.1.4.2). It is interesting to note here, that as far as the simulation activity is concerned, three different time concepts may be identified. These are:

a) Real time, in which the actual system operates.
b) System time or simulated time which is the representation of real time within the simulation model.
c) Computer time, the time taken by the computer to run the model.

It is also interesting to note the following relationships between the above mentioned time concepts:

1) An activity in the model that results in a change of state consumes computer time while taking place instantaneously in simulated time.

2) An activity that consumes simulated time e.g. the scheduling statement \texttt{WAIT(X UNITS OF TIME)}, is instantaneous in computer time.

In CSL and SIMSCRIPT, the dynamic structure is given entirely in terms of events, the events being implicit in CSL. The events are associated with objects by operating on or making reference to the attributes of the objects. In GPSS and SIMULA, the concept of scheduling statements, e.g. \texttt{WAIT}, \texttt{WAITUNTIL} etc., representing the passage of simulated time within a transaction or process respectively, permits the stringing together of events that take place at different points in simulated time. GPSS allows both imperative and interrogative scheduling e.g. \texttt{ADVANCE < arguments >}. Here, the transaction executing the statement is scheduled $\Delta T$ time units later, say, $\Delta T$ being determined by the arguments values and the transaction is suspended for $\Delta T$ units of time. The interrogative scheduling is implied in statements such as \texttt{SEIZE} and \texttt{ENTER} for acquiring system resources. When the resources are not available,
the transactions are queued. GPSS statements are executed interpretively and correspond to powerful sub-routines.

In SIMSCRIPT, where the three building blocks are entities, attributes and sets, only imperative scheduling is allowed, using statements such as CAUSE, SCHEDULE and RESCHEDULE e.g.

SCHEDULE AN ARRIVAL IN 5 MINUTES.

The statement CANCEL P CALLED X

removes the event notice of object X of class P from the event list.

SIMULA offers imperative scheduling only, though the range of scheduling statements is comprehensive. e.g.

\[
\begin{align*}
\{ \text{ACTIVATE} \} & \quad \times \quad \{ \text{AT} \} & \quad \text{T} & \quad \{ \text{PRIOR} \}
\end{align*}
\]

is for direct activation of process X either at simulated time, T (AT) or at TIME + T, where TIME is the current simulated time (DELAY). If the option PRIOR is used, the event notice is inserted in the event list in front of any other event notices of processes scheduled for the same system time. The statement

\[
\begin{align*}
\{ \text{ACTIVATE} \} & \quad \times \quad \{ \text{BEFORE} \} & \quad \text{Y}
\end{align*}
\]

is for relative activation of process X with respect to process Y.

The statement HOLD(T) suspends the running process for T units of simulated time and CANCEL(X) removes the event notice of process X from the event list if it is scheduled, otherwise the statement has no effect. Every event notice has the following attributes: SUCC and PREDE to reference the successor and predecessor of the event notice respectively, EVTIME, the time of next activation of a process and PROC, a reference to the process to whom the event notice belongs.

In CSL, sequencing may be considered as interrogative only, since time-cells values may enter into complex logical tests before the execution of an entity. The user interacts with the simulation
executive program through assigning numerical values to the time cells. The scheduling characteristics of the four SPLs are summarised in Table (2.4).

2.2.4 Simulation Support

A simulation exercise requires a number of supporting facilities during the development and production runs of a model. One such facility is the generation of stochastic variates from different statistical distributions. Data collection, analysis and display is another important aspect. GPSS automatically performs data collection and estimation and then prints summary statistics at the end of a run. In SIMULA and SIMSCRIPT, the user has to program the data collection analysis and display, though SIMSCRIPT provides a convenient way of computing sums, means, variances, standard deviations etc. SIMULA provides the system procedures HISTO and ACCUM for histograms and weighted queue lengths. DEMOS (BIRT 79 and SIMON (HILL 76), both written in SIMULA source code, provide automatic data collection, analysis and display. CSL provides histogram compilation and printing.

SPLs are expected to assist in the monitoring and debugging of a simulation model. This includes trapping of syntactic and semantic errors at compile time rather than run-time and this reflects the level of security in an SPL. In SIMULA this level of security is high. For example, referencing is checked at compile time and so is the fact that reference expressions evaluate reference values. Remote accessing is also checked at compile time and a user is forced to qualify his references. On the other hand, the security level of SIMSCRIPT is low (DAHL 68); attribute references are not checked. For example, if a reference X to an object is in error, then the effect of assigning to an attribute A(X) is
unpredictable, as is the use of DESTROY when X does not reference an existing object. In CSL, attribute references are easily checked, since object references are ordinary numbers in a known list or set. Monitoring and debugging are also assisted by the availability of a tracing facility that echoes the system dynamics as it evolves through time and displays the contents of tables, counters, queues etc. Some SPLs possess tracing facilities of varying strength, while others leave it for the users to provide their own e.g. SIMULA.

2.3 CRITERIA FOR SIMULATION PROGRAMMING LANGUAGE (SPL) SELECTION

Some 140 simulation languages and packages have been written (SIMU 79) since Tocker et al's pioneering paper (TOCK 60). Out of those, 70 are discrete-event languages, 30 are continuous and 30 are combined (discrete-continuous). Continuous simulation languages are targeted at representing a model by a set of differential equations and solving these equations either by using the circuitry of an analogue computer or by numerical computation. Combined discrete/continuous simulation languages are a recent development and are aimed at modelling systems (such as a chemical plant) where, for example, differential equations govern the reaction rates, while the events of switching the reactors on and off are provided discretely, (CUNN 76).

Selection of a particular simulation language from this ocean of languages proved to be a frustrating and formidable job. The factors that influence the selection process are many and diversified. Unfortunately, there exists no definite set of rules for this selection process.

52
There are few simulation languages reviews in the literature. They do help in giving a flavour of some of the most commonly-known languages through contrasting their approaches, facilities, capabilities etc. e.g. (DAHL 68, TOCK 64, KIVI 69A, KAY 72, TEIK 67). Some authors did provide a quantitative comparison of a few of the well-known languages, though the factors taken into account were not comprehensive. e.g. Tognetti et al (TOGN 72) reported a quantitative comparison of SIMSCRIPT II and SIMULA 67 using a single-server queuing program as an example and reporting on aspects such as documentation, capabilities and efficiency. The conclusion was that SIMSCRIPT II, is better documented and easier to learn while SIMULA 67 is more capable and efficient. Virjo (VIRJ 72) has reported a similar comparison for GPSS, SIMULA and SIMSCRIPT, concluding that SIMULA is the most capable of the three. Hills (HILL 73A) compared the activity, event and process approaches to simulation using an example of a steel mill written in SIMULA, and concluded that the process approach is the most compact and efficient. But the most comprehensive survey of users' views on simulation languages was conducted by Kleine (KLEI 70, SHAN 73, KLEI 71). A questionnaire was distributed among simulation users in the USA and 103 responded. Four questions were posed in the questionnaire. These were familiarity and experience with the language, preference, evaluation of ease-of-use and evaluation of capability. The languages considered were GPSS, SIMULA, SIMSCRIPT 1.5, SIMSCRIPT II, GASP, PROGRAMMING BY QUESTIONNAIRE, FORTRAN, PL/1 and APL. The survey was by no means comprehensive. The sample size was too small to warrant valid statistical inferences, Kleine himself commented that "One can only conclude that one should try to conclude very little about opinion surveys" (KLEI 71).
However, the general consensus was that simulation languages were preferred to general-purpose languages as far as capability and ease of use are concerned especially SIMSCRIPT, GPSS and SIMULA. As for experience and preference, the balance was very much in favour of SIMSCRIPT, GPSS and FORTRAN. One might wonder, as did Palme (PALM 71), as to the outcome of an identical survey had it been conducted in a different country, say Sweden, where the dominant language is SIMULA. A more comprehensive study is now underway at the centre in Simulation, University of Lancaster, U.K. The objective is to provide a comprehensive easy-to-use document on simulation languages, their properties and criteria of choice (ELLI 78). If a survey is conducted, one would hope that the sample would include simulation users from USA, GB and Europe where the three approaches event, activity and process (and hence the languages based upon them) each have a strong hold.

As stated before, many factors effect the decision as to which language to adopt for a particular application. Judging by our own experience, the factors most relevant are the following:

1. The characteristics of the system to be simulated
2. The specifications of the resulting model
3. The portability of the model programs
4. Simulation Programming Languages available on site.
5. Cost of installation, learning and maintenance of a new simulation language.

The first factor determines the approach best suited for the problem at hand. The second factor determines the extent of the capability and flexibility required of the language adopting that approach in order to fulfil the stated goals of the simulation study. Here is included the modelling power, monitoring and debugging facilities, automatic reporting, etc. The other three factors are self-explanatory.
The specification of the simulation package to be developed for the telecommunication-oriented, multi-processor system (Mark II BL) is summarised in Section (5.2). The portability of the model programs between Plymouth Polytechnic where the research was conducted, and GEC (Telecommunications) Ltd., Coventry, where the system is being designed and built was of prime importance. So was the cost incurred if a new simulation language was to be installed at Plymouth Polytechnic, Computer Centre. It was felt that a base language (i.e. FORTRAN) was available at both sites and hence a simulation language based on FORTRAN would resolve the portability problem.

CSL is available at the Polytechnic Computer Centre. It is FORTRAN-based and uses a translator program to translate CSL statements into FORTRAN. It is capable of handling complex logical decision making. It was developed in this country and hence widely used together with its successor ECSL. So it was decided to use CSL to build the simulation model. The decision was thus heavily influenced by the availability, portability and cost factors.

The telecommunication-oriented multiprocessor system and its telephony environment was modelled by being abstracted in a number of activities sub-programs and sub-routines in CSL. The model was developed and ran successfully and a number of results were obtained. The model program with its flow charts are included in Appendix (A). A number of comments on the CSL model are appropriate here:

1) Mark II BL system is highly modular in nature—both in hardware and software. The software development engineers developing the system use a block-structured language, CORAL 66, in developing the operating system and application software.

2) CSL proved to be restrictive in its modelling power for this kind of application where dynamic memory allocation i.e. generation of transient objects, parallelism and the ability to construct hierarchically—structured models are of prime importance.

3) It was rather difficult to explain the model to the software engineers who are the ultimate users as an aid in their software development. This is because the system dynamics were fragmented into a number of activities with no obvious correspondence between the activities and the system software and hardware modules.
Thus if the simulation package is to be easily usable by software design engineers with little or no knowledge of simulation techniques, the first two factors in SPL selection criteria (i.e. characteristics of system and model) should have the over-riding consideration over the other remaining three. The search for an alternative SPL in the light of these new considerations continued. At one time, the use of a program generator package, DRAFT (MACH 74) was proposed. This was rejected as being unsuitable for a problem of this size and complexity in a discussion with its author. Opinions of other experts in the field were sought and the general view was that SIMULA is most suitable for this particular application. Some features of SIMULA have already been discussed. We will now divert our attention to the characteristics of the language that influenced its choice.

2.4 SIMULA 67

SIMULA 67 is the successor to SIMULA (DAHL 66). It has ALGOL as a sub-set because its basic structure lends itself to extension (NAUR 63); i.e. it is ALGOL +. It extends the ALGOL block concept to enable the generation and naming of blocks that can exist as coroutines. These represent objects in a model generated from a template of class or process declaration. The language was designed by Dahl, Myhrhang and Nyguard at the Norwegian Computing Centre, Oslo, Norway. The language was developed as an all-purpose kernel where application packages for specific problem areas can be built. This was thought an appropriate solution to the problem of proliferation of SPLs. However, the realisation of model programs using the process approach was the major impetus (FRAN 77), first used by Knuth in the design of SOL language (KNUT 64).
The language is available on most computers e.g. IBM, ICL, CDC, BURoughS, DEC10, UNIVAC etc., and portability of SIMULA programs is maintained by the implementers of SIMULA systems on different computers adhering to a common standard set by the SIMULA standards group in the SIMULA Common Base Language (DAHL 70).

The main features of the language are modularity, parallelism and hierarchical model structure. The language, thus, provides mechanisms to:

a) Decompose a system into natural, easily conceived components and specify process descriptions for the components consisting of complex data structures and action parts.

b) Generate and name a variable number of instances of those processes which co-exist, interact and progress in parallel in simulated time. Processes interact through interaction mechanisms such as object references, remote accessing, quasi-parallel operation and block and class concatenation.

c) Concatenate class declarations to other classes and blocks to realise a hierarchical model structure.

d) Provide a powerful list processing and sequencing capabilities to relate and operate a collection of classes and processes.

e) Reduce debugging effort by providing 'reference security' where invalid data referencing is spotted at compile time.

The features (a) to (e) are the very ones required to build a flexible hierarchical model structure where a hardware or software module in the real system is mapped in a one-to-one correspondence to its model module. The superior capability of the language compared to other SPLs is beyond doubt (HILL 73A, TOGN 72, KLEI 71). The language itself is more difficult to learn and use than other SPLs e.g. SIMSCRIPT, GPSS, but the effort put in that direction is worthwhile.
SIMULA 67, itself contains sufficient primitives to simulate any model. However, it does not contain all the facilities or 'niceties' one expects from a whole-hearted SPL such as automatic reporting, process tracing etc. But being an application-oriented language, a few simulation wizards can develop these features and others in packages intended for different application areas (HILL 76, BIRT 79, LOKE 74, CUNN 76, VAUC 71 etc.). A number of packages now exist in areas as varied as simulation, computer graphics, data base management, communication networks, etc. In simulation, one such package, SIMON, (HILL 76) is being used extensively in industrial simulation and teaching of simulation courses and operational research.

DEMONS, which appeared recently (BIRT 79) includes new pre-defined resource entities and a rather efficient 'WAITUNITIL' algorithm. These packages are very powerful, easy to learn and use, thanks to the concatenation feature of SIMULA.

From the point of view of simulating the MARK II BL multiprocessor system and other System X sub-systems, SIMULA provided those characteristics which solve many of the problems experienced in the earlier experiments with CSL, namely:

1) The inability of CSL to produce a modularly-structured simulator, where hardware components and software processes can be modelled in a one-to-one transformation process.

2) CSL does not lend itself to the modelling of transient objects, nor does it facilitate the modelling of parallelism in a system with complex interactions between the parallel processes.

3) The difficulty experienced by the software engineers in understanding and using the simulator. This was mainly due to the lack of resemblance between the simulator and the real system.
CHAPTER 3

3.1 THE TELECOMMUNICATION NETWORK

The Telecommunication Network is by far the biggest man-made system in the world, interconnecting some 400 million telephones, sustaining millions of independent conversations simultaneously and in an ever-varying pattern. The network is growing at an average rate of 7% and providing more and more varied services e.g. telephone, telex, data transmission, viewdata etc. and the annual expenditure in the 'Western Block' alone amounts to 2.5 billion pounds (TIPP 77).

This monstrous man-made system was born just over a hundred years ago with the invention of the telephone by Alexander Graham Bell in 1876. The word 'telephone' itself was first used in 1796 for a purely acoustic method of communication and later for telegraph systems where the received electrical signal generates a sound heard and interpreted by the operator (FLOO 76). As the number of subscribers grew, it became clear that it was impractical and uneconomical for each subscriber to have links to every other subscriber. The solution adopted was to develop routing or switching equipment at one central location to which all the subscribers were star-connected. Thus in 1878, the first public telephone exchange came into being interconnecting subscribers.

However, the telephone service continued to grow, and the number of subscribers became inconveniently large to be handled by one local exchange and thus a number of local exchanges were built to
deal with each community. Moreover, subscribers of one local exchange required to establish conversations with subscribers of another local exchange and so the local exchanges were either mesh connected or star-connected through a 'trunk' exchange, thus a network hierarchy of exchanges resulted.

The U.K. network hierarchy is shown in Figure (3.1). The routes or circuits between local exchanges are called junctions while circuits to higher level exchanges are called trunks. Due to economical considerations; the network is not entirely star-connected and hence interconnections between exchanges in the same level do exist. In the U.K. communication network, there exists about 6200 local exchanges, 370 primary trunk exchanges (Group Switching Centres), 27 secondary trunk exchanges and 9 tertiary trunk exchanges interconnecting some 25 million telephones (HARR 79). This network is interconnected to the global network via international exchanges.

In the international network, transmission is via submarine cables, microwave radio and communication satellites. High-frequency radio transmission (HF) is still used in certain areas where the traffic is low. The transmission quality is poorer as it is affected by fading due to the changing nature of the ionosphere.

The total number of telephones in a network is a fair indication of the size of the network, given that the quality of service is comparable. Table (3.1) summarizes the telephone statistics of the 12 countries with the highest number of telephones. One noticeable thing is the number of telephones in the United States which roughly equals those in the rest of the world.
FIG(3.1) UK NETWORK HIERARCHY
<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>NUMBER OF TELEPHONES X10(^0)</th>
<th>PERCENTAGE INCREASE FROM 1967</th>
<th>TELEPHONES PER 100 OF POPULATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>155</td>
<td>57</td>
<td>72</td>
</tr>
<tr>
<td>SWEDEN</td>
<td>6</td>
<td>51</td>
<td>69</td>
</tr>
<tr>
<td>CANADA</td>
<td>148</td>
<td>75</td>
<td>60</td>
</tr>
<tr>
<td>JAPAN</td>
<td>48</td>
<td>203</td>
<td>43</td>
</tr>
<tr>
<td>AUSTRALIA</td>
<td>6</td>
<td>85</td>
<td>40</td>
</tr>
<tr>
<td>UK</td>
<td>22</td>
<td>94</td>
<td>39</td>
</tr>
<tr>
<td>NETHERLANDS</td>
<td>5</td>
<td>115</td>
<td>39</td>
</tr>
<tr>
<td>WEST GERMANY</td>
<td>21</td>
<td>122</td>
<td>34</td>
</tr>
<tr>
<td>FRANCE</td>
<td>16</td>
<td>137</td>
<td>29</td>
</tr>
<tr>
<td>ITALY</td>
<td>15</td>
<td>136</td>
<td>27</td>
</tr>
<tr>
<td>SPAIN</td>
<td>9</td>
<td>180</td>
<td>24</td>
</tr>
<tr>
<td>USSR</td>
<td>18</td>
<td>114</td>
<td>7</td>
</tr>
</tbody>
</table>

**TABLE (3.1): Telephone Statistics of 12 Countries with Highest Telephones/100 of Population**

(WORLD TELEPHONE STATISTICS, 1977)
The highest number of telephones per 100 of population are found in the USA and Sweden whereas the fastest growing network is that of Japan.

The public telephone network can conveniently be considered to comprise of three basic building blocks (LEAK 77); terminal equipment to match the user's or subscriber's mode of communication with that of the network, transmission equipment to convey the information from the sender to the recipient and routing or switching equipment to enable the sender to select the recipient of his choice.

A public network has also some important parameters whose consideration are paramount in the planning of the network. These parameters may be identified as the numbering plan, the routing plan, the congestion standards, the transmission standards, the charging plan and the signalling plan (FLOO 75).

From the foregoing, it is evident that the international telecommunication network is a vast and complex entity interconnecting some 400 million telephones, and the number is doubling approximately every ten years. With the introduction of high capacity submarine cables and satellite systems, international links are by far the greatest growth area. Although the international network is used for other services such as telex, facsimile, television etc., the telephone service still constitutes the highest proportion of traffic carried.
3.2. TYPES OF TELEPHONE EXCHANGES

3.2.1 Introduction

A varied number of telephone exchange systems do exist in the world today, though this variation is fairly transparent to the user of the basic telephone service. Basically, these variations have arisen due to the need for capital and maintenance economy in exchange and external plants, resulting in a number of different basic systems being introduced by different telecommunications manufacturing companies.

Until the 50's the starting point of an exchange organisation and design was usually a switch mechanism proposed to solve a particular problem at the time. For example, the Strowger switch function was to dispense with manual operators for economical reasons and give customers remote control of their connections by sending electrical signals (dialling a number). Likewise, the crossbar switch solved the reliability problem in remote small exchanges before its potential was realised in larger exchanges. However, in newer systems, the designers emphasis has shifted to other areas such as the concentration and enhancement of the processing power of the exchange.

Basically, an exchange consists of a control unit and switches multiplied together and provided on a per-line basis as shown in Figure (3.2). The control sub-system has a number of important functions. It has to receive the routing instructions from the subscriber in a form of a directory number, to translate that number, establish, supervise and terminate a call. Other subsidiary functions include call charging, equipment monitoring and fault avoidance.
FIG(3.2): A BASIC SWITCHING MACHINE
The system depicted in Figure (3.2) will be prohibitively expensive for a large number of customers. To make such a system economically viable, extra stages in the switch network have to be introduced to reduce the number of crosspoints, together with resource sharing of the control sub-system. Thus, practical systems employ resource sharing techniques such as the use of a common switch network and control, functional sub-division and time-sharing decision making (HILL 76A).

The different fundamental exchange organisations can be classified under two categories:

a) the manual exchanges featuring manual control and
b) automatic exchanges - direct control, register control and common control.

3.2.2 Manual Exchanges

The function of a telephone exchange is to interconnect on demand two or more of the exchange terminations for a period, of time, usually to permit speech signals to pass. In a simple conventional manual exchange, the terminations appear on multiplied jacks interconnected by human operators using double plug-ended cords. Each operator is provided with a headphone, keys and lamps to carry the control functions, and normally deals with up to 17 cord circuits. A striking feature of the manual exchange system is its flexibility and power.

To start with, signalling between the subscriber and the exchange is by the spoken word, including names and numbers. The use of human operators here made possible facilities which stored
program controlled systems are trying to achieve, e.g. automatic call transfer based on a customer's visiting habits, ring back when free, etc. Human intelligence also made fault detection, fault and internal blocking avoidance more efficient. The attending, interconnecting, supervisory and operator functions of manual exchanges correspond to the individual line circuits, switching block, supervisory unit and the control unit of a modern exchange respectively. As the communication network grew, it became evident that the number of operators to be provided became prohibitive and call set-up times unacceptable. As a result, an automatic system had to be introduced.

3.2.3 Direct Control Systems

The direct control System is referred to as the step-by-step or strowger system. Here the subscriber is given the ability to remotely control the set-up of his call. In the strowger switch, the cord of the manual system was miniaturised and the plug modified into 'wipers'. It is a two-orthogonal-motion switch control. A subscriber uses a decadic rotary dial to set-up the switches. Each digit is arranged to operate a switch in each stage. A ratchet mechanism moves the wiper vertically in accordance with the number of pulses in a dialled digit. The wiper then hunts in a rotary manner for a free outlet to the next rank of selectors. All the switching stages except the last are operated by single digits, the last stage being operated by the last two digits, to select one of a hundred subscribers. The only common equipment provided here are those for functions such as routing, tone generation etc., thus in a basic strowger system, the control is fully dispersed (Figure. (3.3)).
FIG (3.3): STROWGER STEP-BY-STEP SYSTEM
Since the subscriber does not have knowledge of the overall state of the exchange, the exchange capability is limited. Alternative routing is not possible as the trunking is rigidly fixed by the digit code. In addition, calls between local exchanges are characterised by a multiplicity of dialling codes with variable lengths depending on the number of switching stages, thus there is no unique code identifying a subscriber in the network. Despite these disadvantages, the Strowger switch is simple to understand, cheap and allows flexible modular growth of an exchange. As a matter of fact a large proportion of exchange equipment in the global telecommunication networks is still of Strowger type.

3.2.4 Register Control Systems

The disadvantages of the step-by-step or Strowger control in the rigidity of routing and in non-uniform numbering were solved by the introduction of a register-translater (or director) - Figure (3.4). This enables a standard number to be dialled, only dependent on the location of the called subscriber. The translation function is made to vary from exchange to exchange to translate a directory number into an equivalent number (equipment number) which is actually used to route the call. The director is located immediately after the first concentration stage and through a further concentration stage (the register-access switch). The first few digits of a subscriber number represents the unique code digit of his exchange in his area (local exchange). The director translates these into a corresponding set of digits chosen to meet the requirements of switching and routing economy at the particular originating exchange. The digits after the exchange code are passed without translation. Thus, the prime aim is to eliminate the mutual
FIG (3.4) A STROWGER EXCHANGE WITH A REGISTER TRANSLATOR
restrictions between telephone numbers and the switching machine.

A second reason for using a register is in the situation when the switching mechanism is unsuitable for the reception of dialled digits directly. Such situations include mechanical unsuitability of the switch in some motor-driven switches, or when the switch is not decadic or it is inefficient to be so as in a cross-bar switch. Hence registers are introduced in crossbar exchanges to store and translate the dialled digits in conjunction with a translator. Figure (3.5) shows the trunking arrangements of one cross-bar exchange - the Ericson type ARF. Three principal stages of the cross-bar switches are involved: line-stage switches (SLA, SLB, SLC, SLD), register-access switches and group selector switches. At each stage, calls must pass through more than one switch and the operation of the switch is controlled by marker equipment common to many of the switches at each stage.

For an originating exchange call, SLA and SLB initially connect the subscriber to a free supervisory relay set by the SL stage marker. The supervisory unit is then connected to a free register by the register finder marker via the register connecting switches. The signalling information from the subscriber is then transferred to the register. The register performs the digit translation and signals the group selector marker at high speed to mark the appropriate connections.

For an outgoing call, a junction circuit is allotted, while for an own exchange call a path is connected back to the line-stage switches. Advantages of cross-bar systems include independence between directory and equipment numbers, high reliability, low post dialling delay in multi-exchange calls due to the use of multifrequency signalling between exchanges and backward signalling which
FIG(3.5): TRUNKING OF LM ERICSSON ARF CROSSBAR EXCHANGE
enables an originating exchange to drop a connection and make another attempt to set it up over the same or an alternative route. The use of common control necessitates duplication of equipment and fault detection and isolation equipment.

To economise on cross-points link-coupled trunking is used, where a common control interrogates the paths through two or more stages virtually simultaneously and establishes the connection as a result of this overall appreciation of the network. From about 1950 cross-bar systems were being installed in large quantities. The reliability of the cross-bar switch was extremely useful, particularly when subscriber trunk dialling was introduced (STD). This is because the probability of encountering undetected faulty equipment with Strowger type of exchanges increases owing to the additional equipment necessary to complete a connection.

3.2.5 Electronic Systems

Electromechanical common-control systems still suffered from certain operational disadvantages. For instance, the number translating capability may have to be provided separately in every register and registers have a restricted choice of switch paths. Electronic techniques, both in the switching block and central control were introduced to alleviate these problems. The high speed of operation of electronic components coupled with powerful processors enabled more centralised control, more switching stages, better appreciation of the overall state of the network and flexibility of operation (Figure (3.6)).

Electronic systems are either hard-wired or stored-program-controlled, where the control functions are implemented by software in a central processor. Electronic exchanges are more and more approaching
FIGURE (3.6): A GENERAL MODEL OF A COMPUTER-CONTROLLED SWITCHING SYSTEM
the flexibility and power of manual systems. In the switch block, the availability of cheap crosspoints in integrated circuit form and the use of digital switching (Pulse code modulation and time-division multiplexing) enabled the realisation of economic multi-stage switch blocks with very low probability of blocking. In the following sections, we consider in greater detail the characteristics of stored program controlled (SPC) systems.

3.3 STORED PROGRAM CONTROLLED (SPC) SYSTEMS

The control of telecommunications switching systems has evolved during telephony's first century from manual through electro-mechanical (in various forms) to electronic, both wired-logic and stored program control. The driving force behind this evolution is primarily economics (reduction of equipment and labour costs) as well as the need for enhanced capabilities. In this respect stored-program-controlled systems possess an overall capability not known to conventional systems. This overall capability is revealed by the following features:

(i) The system's ability to detect and isolate faults and reconfigure itself so as to provide a reasonably good service; that is system security.

(ii) The ability of the system to interwork with the existing network with its limited signalling capability, low speed and noisy environment; that is introducability.

(iii) SPC systems have powerful in-built maintainability features that provide rapid diagnosis, reporting and isolation of faults and reporting of defective parts or their restoration to full service after recovery. This is an important feature in the light of higher maintenance cost and scarcity of expertise. This ability is being utilised to centralise the maintenance in a few maintenance and administration centres.
(iv) SPC systems offer new facilities for both the administration and the customer. For the customer, it offers the possibility of new services such as conference calls, abbreviated dialling and call transfer. For the administration it offers improve network management, maintenance control and charging flexibility, all at an economic price.

(v) SPC systems have the evolutionary potential in both hardware and software. This is an important feature allowing the incorporation of changing design concepts and technologies over the life of the system.

(vi) SPC systems result in space saving, power saving and higher traffic capacity as compared to conventional systems.

(vii) SPC systems may cash in on the vast developing technology of digital computers resulting in further cost reduction.

All of these features have been brought about by the employment of software using powerful telecommunication-oriented processors. It is appropriate to consider in greater detail the different characteristics of these telecommunication processors.

3.4 TELECOMMUNICATION PROCESSORS

3.4.1 Their Characteristics

Telecommunication processors differ from commercial computers in that they have to:

1) Provide continuous service even in the presence of faults.

2) Operate normally in the exchange environment without special measures to control closely the ambient temperature, humidity, dust and electrical noise.

3) Be easily extendable in processing power without interruption to service.

4) Operate from the standard exchange power supplies and battery.

5) Use equipment practice compatible with the rest of the exchange.

6) Be maintainable by the exchange maintenance staff.
Although they are similar to commercial computers, they possess features unique either in nature or degree of application.

The real-time functions demanded by telecommunications processing may be classified under the following four classes:

(i) Scanning: the monitoring of the status of lines, trunks and service circuits at a frequency which is a function of the urgency required of a given class of entity.

(ii) Translation: the derivation from the directory number, the equipment number, class of service, routing information etc.

(iii) Call processing proper: hunting network paths, setting up paths, call supervision, charging, path clear down etc.

(iv) Maintenance: Checking, diagnosing, isolating and reconfiguring to various degrees.

The need to perform these functions efficiently has its impact on the processor structure and design. The need to perform scanning at rates independent of the processor load calls for a sophisticated interrupt mechanism to handle time-driven activities efficiently. Beside input/output data processing involving digit reception, digit sending, scanning and peripherals communication functions, the interrupt mechanism is implemented to service software and hardware traps and interrupts as well as a priority-based process structure. This interrupt facility is explained in more detail in Chapter 4.

The need for translation, call processing, input/output control and maintenance, calls for a processor with an extensive instruction repertoire: bit and data field manipulation, masking, rotating and shifting for translation; Boolean logic operations and address manipulation for the call processing, and special instructions for input/output, maintenance and diagnostics.
Telephone calls processing programs tend to be highly
decision-oriented. Extensive testing and branching
instruction are provided for this purpose. Input/output
operations include transfer of peripheral equipment status (such
as lines and trunks) and transfer of control, and addressing
information. Highly-efficient macro-instructions for repetitive
functions are constructed using micro-programming techniques,
such as the calls to the process allocator in Mark II BL System
(Chapter 4). Micro-programming also allows a machine to emulate
another machine and could be used to replace a processor by a more
versatile and updated one such as in the updating of No. 2
processor by No. 3 processor in the ESS family of SPC systems
(MAND 76). For higher programming efficiency, an unusually high
number of registers is provided and accessible to the programmers.

Earlier systems used read-only memories (ROM) for programs,
translation tables and exchange parameters. More recently,
cheap semiconductor RAM memories have been used extensively, with
disc, drum or tape back-up. With increased use of micro-
processors, however, ROM and EPROM are much in use again.

Serial, parallel and serial/parallel bus structures are used
for input/output control. In these systems with such a large
number of peripherals, parallel bus with input/output blocks and
address decode capability are found to be necessary.

The reliability and security levels required in SPC systems
call for processor configurations with duplication or
triplication of some equipment. In Chapter 4, these aspects
are explained with reference to the GEC Mark II BL system.
Arithmetic operations required for telephony functions are
addition, subtraction and logical operations, though some telecommunication processors provide arithmetical functions similar to those of commercial computers. Table (3.2) summarises the telecommunication processors types in the world.

3.4.2 **Configurations of the Processor Utility**

To achieve the high level of hardware reliability required from the processor control sub-system of an SPC telephone exchange (typically 2 hours down-time in 40 years), redundancy is used and implemented in a number of configurations. These may be classified as follows:

(i) Dual Worker, Stand-by  
(ii) Dual Synchronous  
(iii) Dual load-sharing  
(iv) Multiprocessor  
(v) Distributed Control

Figures (3.7.1) - (3.7.3) are simplified schematic diagrams of the first three configurations, whereas Figure (4.1) exemplifies a multi-processor structure.

In dual systems, the word processor implies a processor-store combination. In a worker-stand-by system, one processor is taking the whole load. The stand-by processor is only switched on to take the load when the working processor develops a fault. The stand-by processor may either be 'hot' or 'cold' i.e. the power on or off. Provided the switch-over time is less than the failure defined time (of the order of milliseconds), the reliability of such a system is the same as that of a dual system.

79
<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>Introduction Date</th>
<th>No. of Processors</th>
<th>Word Length</th>
<th>Modes</th>
<th>Micro-program</th>
<th>Gen. Registers</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>United States</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bell System:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. 1 ESS</td>
<td>1965</td>
<td>1 pair</td>
<td>37/23</td>
<td>SYNC</td>
<td>N</td>
<td>0</td>
<td>Ferrite Sheet</td>
</tr>
<tr>
<td>No. 2 ESS</td>
<td>1970</td>
<td>1 pair</td>
<td>10,21/16</td>
<td>SYNC</td>
<td>N</td>
<td>0</td>
<td>IGFET</td>
</tr>
<tr>
<td>No. 3 ESS</td>
<td>1976</td>
<td>1 pair</td>
<td>16,32/16</td>
<td>STANDBY</td>
<td>Y</td>
<td>16</td>
<td>CORE-IC</td>
</tr>
<tr>
<td>No. 1A ESS</td>
<td>1976</td>
<td>1 pair</td>
<td>24,48/24</td>
<td>SYNC</td>
<td>N</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Gen. Tel &amp; Elec:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CI EAX</td>
<td>1967</td>
<td>1 pair</td>
<td>20</td>
<td>STANDBY</td>
<td>N</td>
<td>0</td>
<td>Diamond Ring</td>
</tr>
<tr>
<td>No. 2 EAX(2A)</td>
<td>1977</td>
<td>1 pair</td>
<td>32</td>
<td>SYNC</td>
<td>N</td>
<td>7</td>
<td>MOS(Dynamic)</td>
</tr>
<tr>
<td>No. 3 EAX(2B)</td>
<td>1978</td>
<td>≤4 pairs</td>
<td>32</td>
<td>SYNC/MULTI</td>
<td>N</td>
<td>7</td>
<td>Plated wire core</td>
</tr>
<tr>
<td>North Electric:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ETS-4 (APZ-130)</td>
<td>1975</td>
<td>≤7 pairs</td>
<td>16</td>
<td>SYNC/SHARE</td>
<td>N</td>
<td>4</td>
<td>CORE-IC</td>
</tr>
<tr>
<td>NX-1E(OMN14)</td>
<td>1971</td>
<td>≤4 pairs</td>
<td>16</td>
<td>SYNC/SHARE</td>
<td>N</td>
<td>4</td>
<td>CORE</td>
</tr>
<tr>
<td>Canada</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Northern Telecom:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPL</td>
<td>1971</td>
<td>1 pair</td>
<td>24</td>
<td>SYNC</td>
<td>N</td>
<td>0</td>
<td>Ferrite Sheet</td>
</tr>
<tr>
<td>DMS 100/200</td>
<td>1978/79</td>
<td>1 pair</td>
<td>24</td>
<td>SYNC</td>
<td>Y</td>
<td>16</td>
<td>Semicon</td>
</tr>
<tr>
<td>United Kingdom</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GEC:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mark II</td>
<td>≤12</td>
<td>16</td>
<td>MULTIPRO.</td>
<td>Y</td>
<td>16</td>
<td></td>
<td>Semicon</td>
</tr>
<tr>
<td>Plessey:</td>
<td>≤12</td>
<td>24</td>
<td>MULTIPRO.</td>
<td>Y</td>
<td>8</td>
<td></td>
<td>Plated wire core</td>
</tr>
<tr>
<td>S250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continued......
<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>Introduction Date</th>
<th>No. of Processors</th>
<th>Word Length</th>
<th>Modes</th>
<th>Micro-program</th>
<th>Gen. Registers</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E10(CS40)</td>
<td>1970</td>
<td>5</td>
<td>32</td>
<td>FUNCTIONAL</td>
<td>Y</td>
<td>N</td>
<td>Core</td>
</tr>
<tr>
<td>E11(ITT3200)</td>
<td>1976</td>
<td>2</td>
<td>32</td>
<td>LOADSHARE</td>
<td>N</td>
<td>N</td>
<td>Core</td>
</tr>
<tr>
<td>E12(CS40)</td>
<td>1973</td>
<td>2</td>
<td>32</td>
<td>LOADSHARE</td>
<td>Y</td>
<td>N</td>
<td>Core</td>
</tr>
<tr>
<td>W. Germany</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESK10,000(ELST 801)</td>
<td>1966</td>
<td>≤35</td>
<td>12</td>
<td>SYNC</td>
<td>N</td>
<td>N</td>
<td>Diamond Ring</td>
</tr>
<tr>
<td>EDS</td>
<td>1975</td>
<td>1 pair</td>
<td></td>
<td>SYNC</td>
<td>N</td>
<td>N</td>
<td>Core</td>
</tr>
<tr>
<td>EWS01</td>
<td>1973</td>
<td>1 pair</td>
<td></td>
<td>SYNC</td>
<td>N</td>
<td>N</td>
<td>Core</td>
</tr>
<tr>
<td>EWSF1</td>
<td>1978</td>
<td>1 pair</td>
<td></td>
<td>SYNC</td>
<td>N</td>
<td>N</td>
<td>Core</td>
</tr>
<tr>
<td>Sweden</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ericsson : ARE 1973</td>
<td>2-3</td>
<td>16</td>
<td>LOADSHARE</td>
<td>N</td>
<td>N</td>
<td>Core</td>
<td></td>
</tr>
<tr>
<td>AXE 10,11(APZ 210)</td>
<td>1975</td>
<td>1 pair</td>
<td>16</td>
<td>SYNC</td>
<td>N</td>
<td>N</td>
<td>Core</td>
</tr>
<tr>
<td>AK12(APZ110)</td>
<td>1968</td>
<td>1 pair</td>
<td></td>
<td>SYNC</td>
<td>N</td>
<td>N</td>
<td>Core</td>
</tr>
<tr>
<td>12(APZ120)</td>
<td>1968</td>
<td>1 pair</td>
<td></td>
<td>SYNC</td>
<td>N</td>
<td>N</td>
<td>Core</td>
</tr>
<tr>
<td>13(APZ150)</td>
<td>1971</td>
<td>≤8 pairs</td>
<td></td>
<td>SHARE/FUNCT</td>
<td>N</td>
<td>Y</td>
<td>Core</td>
</tr>
<tr>
<td>Netherlands</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHILIPS:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRX205(TCP18)</td>
<td>1972</td>
<td>≤4 pairs</td>
<td>16</td>
<td>SYNC/SHARE</td>
<td>Y</td>
<td>8</td>
<td>Core</td>
</tr>
<tr>
<td>TCP36</td>
<td></td>
<td>≤8 pairs</td>
<td>32</td>
<td>SYNC/SHARE</td>
<td>Y</td>
<td>8</td>
<td>Core</td>
</tr>
</tbody>
</table>

continued........
<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>Introduction Date</th>
<th>No. of Processors</th>
<th>Word Length</th>
<th>Modes</th>
<th>Micro-Program</th>
<th>Gen. Registers</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Japan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D-10</td>
<td>1971</td>
<td>≤ 2 pairs</td>
<td>32</td>
<td>SYNC/SEP</td>
<td>N</td>
<td>16</td>
<td>Core-plated wire</td>
</tr>
<tr>
<td>D-20</td>
<td>1973</td>
<td>1 pair</td>
<td>16</td>
<td>SYNC/SEP</td>
<td>N</td>
<td>4</td>
<td>MOS, DRUM</td>
</tr>
<tr>
<td>ITT Metaconta</td>
<td>1973</td>
<td>≤ 8</td>
<td>16/32</td>
<td>LOADSHARE</td>
<td>N</td>
<td>16</td>
<td>Core</td>
</tr>
<tr>
<td>10C(ITT1600/3700 3202)</td>
<td>1973</td>
<td>16/32</td>
<td>LOADSHARE</td>
<td>N</td>
<td>16</td>
<td>Core</td>
<td></td>
</tr>
<tr>
<td>11AC(ITT1600/3200)</td>
<td>1968/72</td>
<td>2</td>
<td>16/32</td>
<td>LOADSHARE</td>
<td>N</td>
<td>16</td>
<td>Core</td>
</tr>
<tr>
<td>TCSS(ITT1050/1652)</td>
<td>1974</td>
<td>2</td>
<td>16</td>
<td>LOADSHARE</td>
<td>N</td>
<td>16</td>
<td>Core</td>
</tr>
</tbody>
</table>
**Fig (3.7.1): Main and Stand-by System**

**Fig (3.7.2): Dual Synchronous System**

**Fig (3.7.3): Dual Load Sharing System**
The stand-by processor might be updated by the worker and given background work to perform. Disadvantages of such a configuration are that detection of errors is difficult, switch-over circuitry is costly, and calls in the set-up stage are lost when one machine fails.

In dual synchronous systems, both processors are locked together at the clock frequency to perform in synchronism the same function, but only one processor provides the output. The result of each operation is compared by the comparator circuitry and if disagreement occurs, diagnostic routines are run to determine which of the machines is faulty. This is the most widely-used configuration in terms of the number of systems designed and those in service. Advantages include easier fault location due to instantaneous comparison of data at all stages of processing and absence of contention between machines when handling the same call. On the other hand, the configuration poses some problems, namely the correct determination of the faulty machine, the reliability of the comparator, the 'deadly embrace' problem where the faulty machine takes out of service the good machine and the difficulty of connecting backing store units such as drum, disc or magnetic tape units to the synchronous machines. Examples using this configuration are No. 1 ESS (USA) and SP-1 (Canada).

In dual load-sharing systems, the work load is divided between the two processors by a scheduler. In classical load-sharing, the two processors are switched on and off say 10 msec on and 10 msec off. During its on period, the processor handles the traffic that originates during that period. Each machine updates the other by using messages. Advantages of load-sharing include lower probability of simultaneous program faults and better overload characteristics.
compared to dual synchronous systems. It is also more flexible in that the load sharing percentage may be reduced from 50-50 to 100-0 to allow for on-line updates and for the introduction of new facilities. Its problems include the requirement for comprehensive diagnostic software and self checking circuitry and the possibility of 'deadly embrace'.

An example of a load-sharing system is the Metaconta L (KOBU 72). Some dual systems such as the E 11 (France) and the Metaconta 10 and 11 (ITT) can be switched dynamically from load-sharing during busy periods to a synchronous pair during light traffic periods (BRIL 77). Such a system has better overall performance characteristics.

A common disadvantage of all dual systems is the high cost of incremental growth since the processors are added in pairs with increased problems of inter-processor communication and fault location. A multiprocessor configuration alleviates this problem and allows the m + n redundancy principle to be used. In the Mark II BL System: m processors take the load and n are redundant. In this configuration, the processors and storage modules are separate and connected to a common highway and each processor can access any of the storage modules. Jobs may be run on any available processors. Examples of this configuration are the GEC Mark II BL system and the Plessey S250 system. The Mark II BL is described in greater detail in Chapter 4. A major problem with multiprocessor systems is fault detection, containment and system recovery due to faulty processor corrupting areas of the common store. Sophisticated techniques have been developed to deal with this problem (OWEN 73, EDGE 72).

Other examples of multiprocessor systems are the E12 (France) and the PRX (Netherlands).
An example of the relative reliability of dual and multiprocessor systems is given below, based on the procedure used in the Post Office Requirements Document 1075 (HALT 77).

\[ F_{100} = \frac{8.76 \times 10^5}{mC_{n+1}} \left( \frac{D}{u} \right)^{n+1}(n+1) \]

where

- \( F_{100} \): number of failures per 100 years
- \( D \): Mean time to repair
- \( u \): Mean time between failure of module
- \( m \): Total number of identical modules
- \( n \): Number of spare modules
- \( 8.76 \times 10^5 \): Number of hours in 100 years

\( D \) is normally taken to be 5 hours. \( U \) is calculated from the reliability figures of components in a module and is typically 8920 hours or approximately one year.

Consider a dual processor system where the maximum load can be taken by one processor, then

- \( U = 8920 \) hours
- \( m = 2 \)
- \( n = 1 \)

\[ mC_{n+1} = \frac{m!}{(n+1)!(m-n-1)!} = 2C_1 = \frac{2 \times 1}{1 \times 2} = 1 \]

\[ : F_{100} = 175 \times 10^3 \times 1 \times 31.3 \times 10^{-8} \times 2 = 0.10955 \]

\[ : \text{MTBF} = 912 \text{ years. MTBF is the mean time between failures.} \]

On the other hand, considering a multiprocessor system where the load is taken by 3 processors with only one processor spare \( i.e. \) \( m = 4 \) and \( n = 1 \), we get
\[ F_{100} = 0.6573 \]

and

\[ \text{MTBF} = 152 \text{ years}. \]

The advantage of the multiprocessor system is that it provides a reasonably high degree of reliability at an economic price and allows for a smooth economic growth. If dual processors are used for the above multiprocessor systems, a total of 6 processors would have been required.

With the advent of microprocessors, the trend is again shifting towards distributed control using loosely-coupled microprocessors in a multi-microprocessor configuration (NISS 79, CULL 79). These systems are still under development.

3.5 SPC SOFTWARE ORGANISATION

There is no standard classification of SPC software types, however. One possible such classification is:

1) Call Processing Proper
2) Real-time Operating System and Support
3) Maintenance and diagnostic
4) Administrative.

The total amount of software in an SPC system is huge and costly. Formalised methods of specifying and producing modular, efficient and manageable software have been proposed and used. Such methods include structure-oriented modelling (BRAE 79), specification and description languages based on state transition diagrams (CCIT 77, KAWA 71, GALE 75, GERR 74), structured programming (DIJK 72, BAKE 75), generic program production (KAWA 79) and verification-oriented software specification (CUNN 81).
1) **Call Processing Proper:**

These are the programs that implement the telephony functions and facilities in a particular exchange, for example, call detection, digit reception, route translation, path search, and supervision. They also implement new facilities, such as conference call, camp-on, and abbreviated dialling.

Three approaches in implementing the suite of programs are generally followed, namely, function division, time division and call division or state-of-call (SOC) (HILL 76A, LAW 72). A combination of these approaches is normally employed, for example function and time division in System X software. Program modules that have stringent real-time requirements or carry important functions are activated periodically and have a higher priority than other programs.

2) **Real-Time Operating System and Support**

The real-time operating system manages the resources of the processing utility, including the CPUs and store blocks, and provides facilities such as timing, interrupt handling, I/O and communications between the software modules. Its structure depends to some extent on the architecture of the machine for which it is written. It runs on-line.

Support software includes compilers, linkers, loaders and debugging aids and these are off-line programs to effect modifications and/or extensions of existing programs or additions of new ones. Debugging aids include module logic
testing using emulators, system testing using an on-line break-point program or data tracer and simulation for design checking, performance evaluation and system tuning.

3) **Maintenance and Diagnostics:**

The overall maintenance process is composed of fault detection, fault recovery, fault diagnosis and fault repair in this sequence. Great importance is attached to these maintenance programs in view of the high reliability required of SPC systems (1 hour in 20 years downtime) in spite of the multiplicity of hardware and software faults that may arise. Half of the total exchange software is likely to be for maintenance and diagnostics (Table (3.3)).

Fault detection is aided by hardware parity checking, comparison of outputs from duplicated equipment, validity checking, routing and time outs. Fault isolation and recovery is carried out before fault diagnosis because of the necessity for guaranteed continuity of service. Diagnostic software is then invoked which will examine the faulty entity if it is a hardware fault or re-initialise it if it is a software process. Sophisticated techniques to implement this function have been developed for various systems (OWEN 73, EDGE 72, ARGO 79).

4) **Administrative**

Ease of management of SPC systems is important to operating administrations. Software in this area is used for operations 'soft data' or semi-permanent data operations relating to routing, junctions, exchange configuration control and extension, traffic measurement and subscriber and system monitoring.
<table>
<thead>
<tr>
<th>SYSTEM</th>
<th>MAINTENANCE CODE (WORDS, APPROX.)</th>
<th>APPROX. PROPORTION OF OPERATIONAL SOFTWARE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS1 (USA)</td>
<td>100K</td>
<td>53%</td>
</tr>
<tr>
<td>D10 (JAPAN)</td>
<td>67K</td>
<td>54%</td>
</tr>
<tr>
<td>CNET (FRANCE)</td>
<td>50K</td>
<td>50%</td>
</tr>
</tbody>
</table>

**TABLE (3.3):** Percentage of Maintenance Software for some of the Well Known SPC Systems

<table>
<thead>
<tr>
<th>SYSTEM</th>
<th>Termination Capacity</th>
<th>Switch Capacity (Switched erlangs)</th>
<th>Processing Capacity (busy hour call attempts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplexer</td>
<td>24 or 30</td>
<td>4 or 5</td>
<td></td>
</tr>
<tr>
<td>Concentrator</td>
<td>2000</td>
<td>160</td>
<td>8000</td>
</tr>
<tr>
<td>Small local Exchange</td>
<td>2000</td>
<td>160</td>
<td>8000</td>
</tr>
<tr>
<td>Medium local Exchange</td>
<td>10000</td>
<td>2000</td>
<td>80000</td>
</tr>
<tr>
<td>Large local Exchange</td>
<td>60000</td>
<td>10000</td>
<td>500000</td>
</tr>
<tr>
<td>Medium Trunk Exchange</td>
<td>8000</td>
<td>2000</td>
<td>80000</td>
</tr>
<tr>
<td>Large Trunk Exchange</td>
<td>85000</td>
<td>20000</td>
<td>500000</td>
</tr>
<tr>
<td>Medium International Transit Exchange</td>
<td>8000</td>
<td>2000</td>
<td>50000</td>
</tr>
<tr>
<td>Large International Transit Exchange</td>
<td>85000</td>
<td>20000</td>
<td>400000</td>
</tr>
<tr>
<td>Combined local and Trunk Exchanges</td>
<td>10,000 subscribers of 5000 trunks in Combination</td>
<td>2000</td>
<td>80000</td>
</tr>
</tbody>
</table>

**TABLE (3.4):** SYSTEM X FAMILY OF EXCHANGES (TIPP 79)
The administration communicates with the exchange through I/O devices, such as a teletype using a man/machine language, in addition to a high speed paper tape punch, a magnetic-tape unit and a test panel or console. All the above functions are in operation while the exchange is carrying live traffic. In addition, centralised maintenance and administration centres for remote exchanges are possible using SPC systems.

Beside the operating system and applications software, the SPC software also contains the exchange-dependent data base relating to customer data, translation tables, routing and charging information and call records.

3.6 SYSTEM X

3.6.1 System X Characteristics

The present telecommunication network in Britain is dominated by electromechanical switching and signalling systems. The ability to exploit these systems on a network basis is constrained by a variety of limitations inherent in these systems. These constraints and limitations of the national network can be summarised as:

1) Domination by 2-wire switching and channel associated signalling.
2) Multiplicity of limited-capacity signalling systems.
3) Relative slow set-up time for multi-link calls.
4) Transmission loss varies with call routing.
5) Prone to noise and distortion.
6) Limited capacity for further evolution and provisioning of new facilities.
7) Mechanical switches prone to wear.
8) Manufacture and maintenance of equipment highly labour intensive.
9) Out of tune with modern technology.

Thus, in the late sixty's, the need was realised for a system which will allow the present network to evolve into one with much greater capabilities was identified by the Advisory Group on Systems Definitions (AGSD) (MART 79).

This fact, coupled with the advances in device technology, led to the adoption of a total approach to the present network problems in a joint venture between the British Telecommunications and the three major telecommunication equipment manufacturers, GEC, STC and Plessey. As a result, System X was born. System X is based on a family of new switching and associated systems characterised by the use of micro-electronics technology, integrated digital transmission and switching, common-channel signalling and stored-program-control. System X is characterised by:

a) 2-wire or 4-wire subscriber switching.

b) 4-wire junction and trunk switching.

c) Extensive signalling capability using common-channel signalling.

d) Fast set-up time for multi-link calls.

e) Transmission loss independent of circuit length because of the use of PCM transmission.

f) Low noise and distortion by the use of digital switching as opposed to electromechanical switches.

g) Extensive capability for evolution and provisioning of new facilities.

h) Facilitates interworking with international networks.
i) Digital switch reliability.

j) Potential automation of manufacture and maintenance of equipment.

k) Exploitation of modern electronic techniques.

l) Integrated transmission and switching using digital techniques minimises the equipment required at the interface between the transmission and exchange equipment. The findings of the UK Trunk Task Force (UKTTF) in the late 60's indicated that by opting for an integrated digital network (IDN), reduction in the total main network cost, in terms of annual charges could be as much as 50% (HARR 79). This is mainly due to the elimination of the intermediate primary multiplexing equipment, pre-circuit signalling equipment, cheaper electronic components and lower manufacturing, accommodation and maintenance cost (See Fig. (3.8)).

m) Use of common channel signalling, because signalling equipment is no longer associated with individual channels there is a significant reduction in signalling equipment required in the system. The signalling sub-system is used to send both control and network management information in the form of flexible, open-ended data messages. Other concepts of System X are remote control of exchange functions and concentrator working. (JONE, 79).

3.6.2 System X Architecture

Among the concepts imbedded in System X design is adaptability and in-service flexibility. This concept is realised through a modular architectural approach both in hardware and software. An individual System X exchange is thus made up of an appropriate number of software and hardware modules or building blocks drawn from a large set of modules.

The sub-division of System X into a number of modules or sub-systems is done on a functional basis. A sub-system interworks with other sub-systems across well-defined functional interfaces which form the boundaries for the sub-systems. They also provide convenient points for growth and adaptability.
(a) EXISTING ARRANGEMENT

(b) INTEGRATED SWITCHING AND TRANSMISSION WITH COMMON CHANNEL SIGNALLING

FIG (3.8): CHARACTERISTIC FEATURES OF EXISTING AND NEW NETWORKS
This approach is attractive in many ways. Changes in user service requirements in traffic or operation strategy can easily be effected by additions or modifications of individual modules. With the rapid advances in microelectronics, new improved devices are easier to incorporate into the hardware modules. The generic nature of System X software provides a central software facility for the generation and assembly of the software sub-systems necessary for particular exchanges. The modular structure with clearly defined interfaces reduces the probability of hardware and software errors propagation. Finally, and most importantly, the modular structure enables the development of the sub-systems to be undertaken by a number of development teams simultaneously and at different physical locations.

3.6.3 System X Sub-systems

(a) HARDWARE SUB-SYSTEMS

1) Processor Utility Sub-system (PUS)

The PUS provides data processing facilities for traffic handling and control of its own and other exchanges. As mentioned before, two versions exist: a smaller version (Mark IP) used in a worker-stand-by configuration for small and medium-size exchanges, as a preprocessor and in SPC assisted electronic systems. The second version is the Mark II BL multiprocessor system for large local and trunk exchanges. Both types are managed by a sophisticated real-time operating system. This sub-system is developed by GEC and is fully explained in Chapter 4.
2) **Digital Switching Sub-system (DSS)**

A DSS has a time-space-time switch configuration as is used to interconnect time-division multiplexed, pulse-code-modulated channels. Each time-switch provides for the connection of 32 line systems with 32 time-slots in each system, and in a large exchange up to 96 such time switches may be provided (TIPP 79, JONE 79) to interconnect both transmit and receive channels with very low blocking probability. This sub-system has a software handler.

3) **Signal Interworking Sub-system (SIS)**

SIS facilitates interworking with existing exchanges that use a multiplicity of signalling systems. It also provides tones and recorded announcements. It has an SIS software handler.

4) **Message Transmission Sub-system (MTS)**

The MTS performs common channel signalling functions over digital bearers at 64 kbits/s. The MTS is based on the British Post Office Signalling System Common-Channel No. 1 (SSCC No. 1) (Jone 79) which may be made compatible with the CCITT signalling system No. 7 for national and international applications.

5) **Analogue Line Terminating Sub-system (ALTS)**

An ALTS is a system for processing signals before entering and after leaving a System X switching node, combining an analogue to digital converter and a digital to analogue converter.

6) **Network Synchronisation Sub-system (NSS)**

The NSS ensures that all System X exchanges in the network are operating at the same average bit rate by synchronising the exchanges clocks to a master network clock.
7) **Subscriber Switching Sub-system (SSS)**

An SSS is used for traffic concentration into heavily used common circuits at a local exchange.

(b) **SOFTWARE SUB-SYSTEMS**

1) **Call Processing Sub-system (CPS)**

A CPS interfaces with other sub-systems to control calls progress through the system. This sub-system is dealt with in greater detail in Chapter 7.

2) **Call Accounting Sub-system (CAS)**

A CAS is located at the local exchange and is responsible for collecting data for call-charging purposes.

3) **Maintenance Control Sub-system (MCS)**

The MCS is used for system diagnosis under faulty conditions and provides help for maintenance staff.

4) **Management Statistics Sub-system (MSS)**

The MSS is used for the collection of telephone statistical data for short and long term planning.

5) **Overload Control Sub-system (OCS)**

The OCS monitors the load on the processor sub-system and when the load on a particular sub-system or the total system load exceeds a pre-determined level, this sub-system interferes to modify the mode of operation of one or all of the other sub-systems.

6) **Multiparty Connection Sub-system (MPCS)**

The MPCS provides for conference calls where three or more parties participate in a conversation.
7) **Automatic Announcement Sub-system (AAS)**

The AAS synthesises announcements from digitally recorded segments of speech for customer exchange interactions. This is used particularly for supplementary services.

8) **Man/Machine Interface Sub-system (MMIS)**

The MMIS provides communication facilities between operation and maintenance staff and the processor sub-system for monitoring, controlling and maintaining an exchange.

### 3.6.4 System X Family of Exchanges

The range of System X family of exchanges is shown in Table (3.4). It is characterised by a wide range of capacities. Its prime role is in the large and medium systems to meet the needs of the telecommunications administrations both here and abroad (TIPP 79). At the lower end of the range (concentrator and small local exchange) with very low traffic, the system designers have to adapt the sub-systems and hardware layout in order to provide economically viable common control, digitally switched exchanges.

At the higher end, the problem is the achievement of the high level of throughput shown (500,000 busy hour call attempts). For a central control using a multiprocessor configuration, this calls for a very efficient operating system and application software. This efficiency also depends on how much software is processed in the central control and how much is devolved to microprocessors located with the hardware of specific systems as well as on the processor speed. Such a high throughput is difficult to achieve and test. One of the objectives of this research is the provision of a software tool to enable the testing of this high throughput (c.f. the DMNSC in Chapter 7). A block diagram of a typical System X Trunk exchange is shown in Figure (3.9).
FIG(3.9): SYSTEM X TRUNK EXCHANGE (DMNSC)
The System X family of exchanges includes, also, local administration centres (LAC) (HARR 79) for exchange and network administration. Each individual local administration centre caters for the administration of a group of exchanges. A local administration centre provides a control centre for maintenance, a concentration point for data being transferred between exchanges and data processing centres. It also provides a concentration and control point for man/machine communication with exchanges.

The penetration of System X in the existing network is expected to be gradual and over a number of years. As more System X and digital transmission equipment is introduced the network will evolve into an integrated services digital network providing 64 kbits/s transmission capability initially, and higher rates subsequently to provide for a multiplicity of services.

3.7 PERFORMANCE EVALUATION OF COMPUTER AND SPC SWITCHING SYSTEMS

3.7.1 The Objectives

The objectives of evaluating the hardware and software performance of computer systems used in commercial and scientific applications and those used in SPC switching systems are similar. The evaluation of computer performance is of vital importance in the selection of computer systems, the design of applications and equipment and the analysis of existing systems.

Historically, only computer hardware performance was evaluated. This included parameters such as the organisation of the machine: the word size, data path and number of addresses per instruction,
I/O channels and secondary storage. With the evolution of the third-generation computers, the programming system has become an integral part of modern computers. Hence, the evaluation process must consider the software as well as the hardware in performance evaluation.

The capabilities of the operating system are central to the performance of a computer system and in particular its multi-programming and multi-processing features. Application programs and software utilities are also a part of the computer system and their performance contribute to the total system performance. Thus, more had to be learned about the internal system operations such as scheduling algorithms and resource allocation policies. Time-sharing, interactive processing and real-time processing emphasized the need for performance evaluation and analysis. This is quite understandable in the light of the high cost of software development and hardware designs, large investments in installed computer equipment and the high cost of running a computer centre. Hence, means and ways of increasing system efficiency through hardware and/or software changes are very desirable.

To put the objectives of computer systems evaluation into perspective, a number of classification schemes have been suggested by practitioners in the field. Lucas (LUCA 71) cited three general purposes of performance evaluation: selection evaluation, performance projection and performance monitoring.

Selection evaluation is concerned with selecting a computer system from among a number of systems on the basis of certain performance criteria. Performance projection is oriented towards designing a new system, a hardware component or a software package.
Performance monitoring provides data on the actual performance of an existing system, by the use of software and hardware probes. This information is used to forecast the impact of changes in the system such as a reconfiguration of the hardware or an improvement in the frequently executed software modules. It is also used to obtain a profile of the use of the system to aid in making strategic decisions, such as the allocation of priorities to process instances in Mark II BL System.

On the other hand, Svobodova (SVOB 76) differentiates between two types of performance evaluation: comparative and analytic.

The former includes:

a) Lease or purchase of new hardware and software.
b) Selection of a supplier of computing services.
c) Classification of existing systems.
d) Comparison of the performance before and after a modification. (DAVI 74, KASP 74)

In analytical evaluation, the performance of the computer system is assessed against variations in the system parameters and work load. This is usually done in an attempt to:

a) Improve the performance of an existing system (system tuning).
b) Maintain the performance of an operating system within specified limits (Performance Control).
c) Design and implement new systems (FAGA 74).

A third categorisation is, due to Bell (BELL 72). He identifies five categories of objectives for performance evaluation and analysis of computer systems. These are feasibility analysis...
proposed a methodology whereby the issues in a simulation are associated with the objectives in a matrix form with the solutions as entries. He also compacted the five objectives into three to make his matrix methodology easier to handle (Table (3.5)).

The three alternative categories of objectives suggested were:

i) Absolute Projection: Here the objective is to make basically dis-similar comparisons. This is the case for example, where a system's processing capacity for a particular load is tested against the maximum allowable time or the response times compared with the stated requirements. A characteristic of these simulations is the necessity of evaluating an objective function in absolute terms with a high degree of absolute accuracy. An example of this is the evaluation of the delays in the DMNSC model, (COHE 68), (STAN 68).

ii) Sensitivity Analysis: This category includes those performance evaluation studies where the emphasis is on similar comparisons (KUCH 75, SHER 72, UNGE 77, KATZ 66, FRAN 73, FACA 74). Here high accuracy is only required in the areas in which the two cases are not identical and the areas that significantly interact with the non-identical areas. A basic characteristic of sensitivity analysis is the comparison of slightly-different alternatives, for example the effects of changing a hardware component, a software module or the scheduling algorithm. A relevant study here is that of experimenting with an alternative.
<table>
<thead>
<tr>
<th>ISSUE</th>
<th>OBJECTIVE</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Absolute Projection</td>
<td>Sensitivity Analysis</td>
<td>Diagnostic Investigation</td>
</tr>
<tr>
<td>Resources</td>
<td>Critical</td>
<td>Important</td>
<td>Desirable</td>
</tr>
<tr>
<td>Changes</td>
<td>Desirable</td>
<td>Critical</td>
<td>Important</td>
</tr>
<tr>
<td>Boundaries</td>
<td>Desirable</td>
<td>Important</td>
<td>Critical</td>
</tr>
<tr>
<td>Costs</td>
<td>Desirable</td>
<td>Important</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>Experimental</td>
<td>Important</td>
<td>Critical</td>
<td>Desirable</td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detail</td>
<td>Macro</td>
<td>Moderate</td>
<td>Micro</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Critical Overall</td>
<td>Critical in places</td>
<td>Reasonableness only</td>
</tr>
<tr>
<td>Validation</td>
<td>Value Comparison</td>
<td>Derivative Comparison</td>
<td>Sequence Checking</td>
</tr>
</tbody>
</table>

TABLE (3.5): Issues vs. Objectives
scheduling algorithm for Mark II Bl system where a process running is only interrupted if it is a background process (Chapter 6).

iii) Diagnostic Investigation: The main objective here is to gain insight into the detailed manner in which the system's components interact and behave or tracing the progress of a transaction or an object to determine whether it goes through the system as expected. The verification experiment reported in Chapter 6 falls into this category. Other cited references include LEHM 68, BARK 69, UNGE 77, REI 68, BLUN 74. The issues or problems particular to the simulation technique of performance evaluation are associated with these three objectives in matrix form as shown in Table (3.5), and will be dealt with in greater detail in (3.7.3).

3.7.2 Performance Evaluation Techniques

3.7.2.1 Introduction

Performance of a computer system can be looked at from two different angles. It may either be defined as the effectiveness with which the system handles a specific application or be defined as a measure of internal efficiency (SVOB 76).

Effectiveness is what is seen by the user, that is "How well does the system enable me to do what I want to do?" Efficiency is how the system uses its resources in order to process a particular work load; "How well does the system do what it is intended to do?" Moreover, system performance can be evaluated only with respect to a particular workload. By workload we mean the total of resource demands from the users community.

105
But workload characterisation and modelling still remains one of the formidable problems in the field of computer performance evaluation and much attention has been paid to it. The following section looks more deeply into this problem area.

3.7.2.2 Workload Characterisation and Modelling

In most computer systems, the instantaneous workload changes quite unpredictably. This is especially true for interactive systems where the speed of the user's response plays an important role in shaping the load generated at individual system entry points. It is this uncontrollable fluctuation of the system workload that makes the characterisation of the workload and hence the system performance valuation so difficult.

Many parameters are used to describe the workload in computer performance evaluation studies. For example job CPU time, I/O requests, inter-arrival time and priority. A complete list of parameters used in workload characterisation is found in (SVOB 76).

Although the workload of a particular system normally has statistical properties that remain constant over reasonably long time periods, these characteristics change as the users community changes, and as new facilities are introduced. This has made workload characterisation difficult to undertake and has lead to the new beatitude: "Blessed is he who found his computer's workload. Let him ask no other blessedness"

Fortunately, in SPC switching systems, the workload is represented by the arrival of telephone calls, characteristically
a Poisson arrival process. Moreover, the resource demands generated by the arrival of a telephone call are known beforehand and deterministic in nature. In other words, the processing of a call is identical to that of all other calls of the same type and hence the computer resource demands are identical. This is unlike commercial and scientific computer systems where every individual job arriving has different resource demands. In such cases, workload models have to be constructed as workload drivers for an actual system or a model of it. A number of such workload models have been suggested and used.

One such model is the instruction mix. This specifies the relative frequencies of usage of different instructions (e.g. ADD, MULTIPLY, JUMP etc.) in a particular application, and it is different from application to application whether scientific or commercial, but one particular mix, the Gibson mix (GIBS 70) is believed to be applicable to a wide class of applications.

Another workload model is a benchmark. This can be an instruction, a special program or a sequence of calls to selected software components. In most cases, however, the term benchmark is used to mean a set of jobs (selected by random sampling of the job stream) that represents a typical workload of the evaluated system. A good benchmark is meant to represent the classes of jobs present and to exercise all the system functions such as scheduling, file management and I/O (LUCA 71).

Another workload model is a synthetic benchmark or a synthetic program. As the name implies, it is a program or set of programs constructed either from the resource demands or service demands but does not necessarily exist before hand (BUCH 69). It includes I/O
considerations, files and environment provided by the operating system (LUCA 71).

Another workload model is the trace. This is a record of selected events that preserves the exact sequence in which these events occurred in the system. It is obtained either from the system natural workload or from a representative benchmark mix. It is used to drive simulation models where the pattern of sequence of events is important (CHEN 69, SHER 72, NOE 74).

Probabilistic workload models, where the workload is characterised by a probability distribution such as the negative exponential distribution, are also reported to have been used both in analytical models and simulations (SVOB 76). These are, of course, approximate models of workload characteristics of computer systems because they are difficult to represent by simple mathematical distributions (ANDE 72).

Models used in conjunction with interactive systems are known as interactive workload drivers. These models take care of user characteristics (such as think time, type time and user-generated interrupts) as well as modelling of resource demands by synthetic benchmarks or ones constructed from the real system commands and input data.

In SPC systems, as telephone calls arrival is characteristically a Poisson process and resource demands of calls of a particular type are identical, the workload characterisation and modelling is simplified to a great extent. A central call generator is used to inject calls into the model at intervals drawn from a negative exponential distribution whose mean is determined by the telephone traffic. For each call generated, the time taken from the start of
ranging to the called party answering and the call duration are drawn from negative exponential distributions with means of 6 seconds and 3 minutes respectively. Resource demands per call are represented by modelling the events occurring at the line circuit. SPC workload characterisation and modelling are discussed in greater detail in relation to the DMNSC in section (7.2.5).

3.7.2.3 Performance Evaluation Models

Computer system performance is a function of a number of parameters, the most important of which are the following:

1) System configuration
2) Resource management policies of the operating system.
3) Efficiency of system and application programs
4) Effectiveness of the processor instruction set
5) The speed of the hardware components.

The performance characteristics are shaped through modelling and measurement during the system design, system implementation and when matching the system to a given workload. These characteristics are shaped through the adjustment of system control parameters, change or modification to resource management policies, load balancing through system reconfiguration and replacement or modification of system components. With the system software, the efficiency of a program is determined by the efficiency of the used algorithm, the programming style and the implementation language.

The general classification of models was outlined in Chapter 2. For the purpose of evaluation of computer and SPC systems, these include empirical models, such as regression models (TSAO 72, SALT 74), analytical models and simulation models.
An analytical model is a mathematical representation of a computing system derived by analysis of the behaviour of the system. A number of such models have been developed, those based on queuing theory are particularly numerous in the literature. They are frequently employed to provide performance data on one particular system component such as CPU scheduling, though whole time-sharing multiprogramming and multiprocessing systems have been approximated by queuing models (FRAN 74, KUCH 75). Unfortunately, representativeness and mathematical tractability in analytical models are conflicting requirements. The class of problems that is solvable with existing mathematical methods is limited and many simplifying assumptions have to be made for other than simple systems. In spite of these simplifications, analytical models play an important role in performance analysis: they provide insight and a quick first-order approximation of system performance. An added advantage is that once a model has been developed, then the cost of obtaining results for that particular class of system is less compared to simulation, say.

On the other hand, the simplifying assumptions necessary to develop analytical models of complicated systems have got to be checked. One way of doing that is through the use of a detailed simulator. This is the technique adopted by the analytical modelling group at GEC Hirst Research Centre, where their simplifying assumption of nodal independence in their development of Mark II BL analytical model, is being checked by the use of the DMNSC simulation model (Chapter 7) developed as part of this research work.

Analytical models do not generally include a comprehensive set of operating system functions nor do they consider the quality of software performance. It is also difficult to include the random effects of multiprogramming and multiprocessing, and for some models, it is difficult to change the parameters for testing different aspects of the system. The development and particularly the revision of models is tedious and time consuming (HUES 67). In many cases, the entire system may be too complex for analytical modelling, giving the interactions between hardware, software, applications programs and a sophisticated interrupt handling structure. Simulation is then used in the detailed study of such complex systems.
3.7.3 Performance Evaluation through Simulation Modelling

The most potentially powerful and flexible of the computer-systems evaluation techniques is simulation, which provides a testing ground for and insight into the functioning of the system (CALI 67). It can be viewed as a combination of modelling and measurement. The process of simulating a computer system consists of building of a model of the system, a model of the workload and a simulation system. The simulation system organises the activities of the model as it evolves in simulated time. This aspect of simulation modelling is fully explored in Chapter 2. Here we are more concerned with the application of the technique in this particular problem area.

Since more details may be incorporated in a simulation, it is used as an extension to analytical modelling where a closed form expression cannot be obtained (LAVE 75) or to validate analytical models, as mentioned before. The level of detail that is difficult to incorporate in an analytical model includes features such as dynamic memory allocation, interrupts in a multiprocessing environment and various system overheads. Moreover a simulation model does not have to use workload models described by stationary probability distributions only. Thus, studies of operating systems' storage allocation and scheduling strategies often require simulations. It is also the only method of estimating the performance of hypothetical systems and new designs before actually implementing them (KOSY 73).

In short, it has been used to fulfil all the objectives in (3.7.1). The ideal parameterised simulator, capable of simulating any proposed computer system is not feasible because they differ so much in their organisation. However, attempts have been made to
develop models for a particular class of computer systems (UNG 72, NEIL 67).

Simulation as a technique has its pitfalls and problems that must be watched carefully so as not to render a costly simulation effort fruitless. An essential ingredient for a successful simulation is a clearly defined and agreed set of realisable objectives. These usually depend on answers to questions such as "What is to be learned about the system under study?" or "What decisions will be based on the simulation results?" Moreover, these objectives cannot be defined without the active participation of the end user. Defining the goals is the first step in any simulation project and perhaps the one most commonly bypassed.

For a simulation project to fulfil its objectives with the minimum time and effort, the simulation team must possess knowledge and experience in at least four areas: firstly project leadership to motivate, lead and manage. Secondly modelling skill to design a conceptual model that mimics the system under study at the appropriate level of detail, thirdly programming ability to transform the conceptual model into a readable, modifiable working program and lastly, sufficient understanding of the modelled system to guide the modelling effort and judge the validity of the simulation results. The model-building team must work with the user organisation from start to finish if both are to have the confidence and understanding necessary to use the completed work effectively.

A model is a simplified representation of a system, and it should incorporate only those features of the system that are important to attain the objectives. Too few details, render the model unreliable; too much detail and it will be costly both in development and use. (TEOR 73)
Selection of the appropriate simulation programming language is an important factor in the success or failure of a simulation project. The language has to be English-like, self documenting and readable by the user. It must support the proper concepts and be powerful enough to cope with the modelling requirements. This problem area has been thoroughly investigated in Chapter 2.

Verification and validation of the simulation model are essential to gain confidence in the model and its results for all parties concerned. This particular problem area is more thoroughly investigated in Chapter (6).

Failure to use modern software engineering tools and techniques (such as structured and modular programming) to manage the development of a large, complex computer program will result in long development time to the extent that it might be too late for the model to be of any use.

Other factors to bear in mind are the resources (man power and machine) required and available, ease of change of a model, total cost and experimental design (BELL 72).

In spite of all these problems and pitfalls, simulation of computer systems will remain the most general, most flexible and most powerful technique for studying and predicting system performance (SVOB 76).

3.7.4 Simulation in the Field of SPC Switching Systems

Before the era of SPC switching systems, the techniques used in the performance evaluation of the switching systems and communication networks in general were, the application of teletraffic engineering
to the dimensioning and capacity studies of the switching systems, field trials, artificial traffic generators and traffic simulation systems (KOST 70, COLE 64, POVE 65). With the merging of computer and communication technologies, the techniques of computer systems are spilling over into the communication engineering field. Examples of this are the specification and description languages (BREA 79, KWA 79) and the simulation of computer systems applied to telecommunications processors.

Real-time environment simulators have been used recently to provide accurate models of SPC switching systems. The simulator is executed in real-time and simulates the whole environment of the system processors, with exact repetition of events as often as required. An environment simulator uses a separate processor to substitute all the elements of the SPC system except the system processors and their software. This includes the cross-points of the switching network, the junctors, peripheral devices, subscribers and trunks, all of which are represented by single bits in the simulator processor memory. This simulated environment is transparent to the system processors and the system state is changed by changes to those bits by the system processors and external events such as call arrivals. Figure (3.10) shows the schematics of a real SPC system and when the network and periphery are substituted by a simulator. The external events such as calls on subscribers or incoming lines, dialing, answer, release, etc. are simulated in the system by a call-pattern program (FONT 71).

Real-time environment simulators have been pioneered and implemented by BTM, a Belgian company of ITT, since 1966 for checking the software of SPC systems such as their METACONTA family of exchanges.
FIG(3.10.1): SPC SWITCHING SYSTEM IN REAL STATE

FIG(3.10.2): SPC SWITCHING SYSTEM WHERE NETWORK AND PERIPHERY SUBSTITUTED BY ENVIRONMENTAL SIMULATOR
With relatively small additional programming and engineering effort, they could also be used for traffic studies of call handling capacity, testing of overload control strategies and the study of software efficiency under load conditions. The measurement programs provided in the system processors are written to suit the user administration requirements for statistics for the correct management of the system in the field, for traffic forecasting and for the measurement of the grade of service.

The calls are generated automatically by repetition of the basic call patterns determined beforehand for each type of call. Each time a call is generated, its parameters are modified. The mean inter-arrival time of each type of call is chosen separately in accordance with the traffic of each type and the call mix. Such types of simulators are reported to have simulated time:real-time ratio of the order 10:1 to 30:1 depending on the traffic (GRUS 76), (LAMP 79).

In an environment simulator, all processors (including the simulator processor) run under the control of one clock located in the simulator interface. By having only one clock in the system, all processors work in complete synchronization. Thus, if the initial conditions are the same, exact repetition of a given simulation is possible. The system allows the system processors to think that they run in real-time. This is achieved by stopping them whenever they receive information from or output information to the interface. When this occurs, the simulator receives an interrupt from the interface. During the time that the system processors are stopped, the real-time counter in the interface is also stopped, so that as far (as the control programs are concerned,) real-time is preserved.
This real-time counter is also stopped when required by the simulation program, and each time an interrupt is generated in the simulator by the interrupt control system located in the interface (every 1 ms), this interrupt initiates a program that updates the environment.

The main objective of an environment simulator remains the debugging and correction of program errors which takes more than half of the total effort spent in the production of real-time software. It is most useful when the hardware is not ready and the software engineers need to check their software, or when it is essential to repeat the environment conditions that generated a software fault. Other uses of environment simulation include evaluation and calibration of a processor and simulation of hardware faults for checkout of test and diagnostic programs. A reduction of up to 30% in debugging and correction time has been claimed by the use of environment simulators (GRUS 76, FONT 71).

Interestingly enough, environment simulators have spilled over to the field of computer engineering and computer science for example the environment simulator for the IBM System 360 (CHAR 78). This is yet another instance of the "convergence and interaction of the two technologies of computers and communications, the initiative this time being taken in the communication engineering field.

One can trace the history of simulation of telecommunications traffic problems back to 1908. The manual methods and special purpose simulating machines (KOST 70) developed then were used until the early 1950's when digital computers and later simulation languages were adopted. However, there has been little cross-pollination between the telecommunication traffic field and other areas hitherto.
That is to say, the development in simulation languages and techniques was not influenced by the nature of problems in the teletraffic engineering area, but rather by the nature of problems in other fields of Operational Research. That was so, because electromechanical switching systems were simple and the objective of a simulation study such as a grade of service or an average waiting time could be obtained conveniently with a roulette-type simulation, which is simpler to construct and faster in operation.

But with the introduction of special purpose computers in the control of switching systems, the objectives of the simulation studies changed to testing of configurations, alternative scheduling strategies, waiting-time distributions and so on. This then called for simulations which included the time dimension (unlike roulette) and which vary in complexity and size according to the objectives of the study. These simulation models were built for particular SPC switching system as well as communication networks (DIET 75, BUSS 68, BERN 68, THOM 79, SCHM 79, YAN 78, FRAS 75, ANDE 72 etc.).

3.7.5 Mark II BL Simulator Package in Relation to Other SPC Systems Simulator Packages

As we have noted, before the era of SPC switching systems, traffic studies were undertaken using teletraffic theory or straightforward and uncomplicated time-oriented or roulette simulations. Hence, there was no cross-pollination between this simulation application area and other areas which might have resulted in new simulation methodologies or approaches (KOST 70).

With the convergence of computer and communications as exemplified by SPC switching systems, the designers and analysts of such
systems started to make use of performance evaluation and monitoring tools already being used in the computer technology field. (GERR 79, SEDG 70).

The simulation package reported in this thesis is an attempt to provide a CAD tool for the performance evaluation of a class of microprogrammed telecommunications - oriented multiprocessor system, with a difference. The difference is that it is meant to be an integrated package that simulates both the system processors and the exchange environment in a flexible level of detail and covers a whole range of System X exchanges.

This is in contrast with the previous simulation work in SPC field where the simulators were either environment simulators or concentrated on one aspect or sub-system only, such as traffic studies, networks and control-sub-system studies.

To provide such an integrated package, a hierarchical modular structure is adopted. Both of these features were exploited to the extreme by the choice of the powerful process-oriented simulation language SIMULA, with its CLASS and concatenation features (Chapter 2). These features enabled a one to one transformation of the system modules into their corresponding simulation modules in a multi-level fashion.

Multi-level simulation has been suggested before (ZURC 68). However, Zurcher et al. used the term multi-level modelling in a different context and meaning. What they suggested was an iterative method with the concurrent existence within a single model of several representations of the system being modelled at different
levels of detail using an activity based simulation approach.
The methodology and philosophy we suggested in this research for SPC systems simulation is primarily concerned with the inexperienced user who can build up his model of a particular SPC system from a library of models, where no more than one representation of a module exists. There are other fundamental differences between the two methodologies and these have been elaborated on in the Introduction Chapter (Chapter 1).
CHAPTER 4

THE GEC MARK II BL MULTI-PROCESSOR COMPUTER SYSTEM

4.1 INTRODUCTION

GEC Mk II BL communications processor is a powerful multiprocessor system which can perform all the functions required in the control of telecommunications switching networks. It has been designed to meet the operational requirements expected of normal switching systems with life expectancy well over 30 years. During its life span, a total switching system failure is expected with a very low probability indeed, and only limited periods of service degradation due to faults or errors is tolerated.

To meet such stringent requirements, the system designers have adopted the approach of using redundancy and dividing the hardware and software into modules with well defined interfaces, thus preventing the propagation of hardware and software faults.

In spite of the cost inherent in this approach, SPC systems must be cost-effective compared to conventional switching systems. To this end, the system is designed to be capable of controlling different types of switching functions within the network. The software organisation should allow both a realistic concentration of programming resources as well as the updating of existing software modules and the addition of new ones on-line while the system is carrying traffic. The hardware organisation should allow an initial reduced system to be used, which can grow in computing 'power' as the work load increases. The hardware must allow the benefits of technological advances to be exploited such as semiconductor technology both to enhance machine power and to counter the effects of component obsolescence. (WARD 72A)
In the GEC Mk II BL multiprocessor system, the store blocks and peripheral devices are shared between the CPUs. The computing load is shared equally between the CPUs, without assigning any specific function to any of the CPUs. Thus, if a program is interrupted in one CPU, it may be resumed later on a different CPU without affecting the program results. That is why the system is both a multiprocessor and a multiprogrammed system.

4.2 Mark II BL Hardware

The prime considerations in the system hardware design were security, reliability of operation and the ability of the system to grow in a modular fashion. Thus the system is divided into basic security modules with well-defined boundaries and interfaces. These basic modules are the CPUs, main random-access memory store blocks, input/output blocks, backing store, interrupt units and DMA multiplexers (Figure 4.1.)

If any fault is detected in a security unit, the entire unit is generally taken out of service. The number of redundant security units allows for the loss of one or even two units of any type without much service degradation.

Virtual addressing is used in a paged-store system. A two-stage translation is used to get the page physical address using firstly a table unique to the process, the process page table, and secondly the system page table. The status of the page is checked at the same time. For protection purposes, each software process is only allowed a limited range of virtual addresses and is restricted in the type of access to a given page within that range (read only, read/write or execute only). The effective address accessed by an instruction is the contents of the base register, plus the value
FIG. (41): EXAMPLE OF MKIIIBL CONFIGURATION
used in the index register (if used), plus the offset within
the instruction. I/O channels are addressed in the same way
as pages and hence the page protection mechanism applies.

As seen from Figure (4.1), the basic modules are the CPUs, main
random-access store blocks, input/output blocks, backing store, the
interrupt triplicates and DMA multiplexers.

Peripherals are of two types: those requiring direct CPU
control are associated with an I/O block which can address up
to 4K peripherals, they are connected by sub-channels. There
are 256 sub-channels to a channel and sixteen channels to a block.
Peripherals requiring DMA are associated with peripheral controllers
which are connected to DMA multiplexers. Typical peripherals are
the backing store units such as drums which require DMA, teletype
writers, VDUs and the different exchange equipment components.

Mk II BL is characterised by a powerful instruction set compatible
with higher-level languages. The instructions allow manipulation
of individual bits and groups of bits, a desirable feature for
efficient store usage and a feature not found in most computers.
This is coupled with a large number of registers and register
instructions which result in fewer store accesses, thus reducing the
store contention and increasing the useful run time. There is a
total of 16 'scratch pad' registers that can be manipulated by a
programmer using register instructions. The store blocks have
16 access ports each. Thus, there is a limit of 16 on the number
of CPUs and DMA multiplexers connected to the store block. The
present Mk II BL version can have a maximum of 11 CPUs.
For security reasons, to make sure that faulty CPUs do not corrupt large areas of memory and that a fault in a store block is limited to that block, each store block has its own power supplies and sequencing control. This asynchronous mode of operation of the store blocks allows store blocks of different speeds to work side by side without having to force the faster blocks to work at the speed of the slower ones as in some systems. Each word consists of two 8-bit bytes with a parity bit per byte. A store block performs parity checks on addresses as well as data.

If a CPU or DMA multiplexer wishes to access a store block, the address (and data) is sent together with a request signal to the appropriate store block. The selection logic of the store block chooses which request to honour in the case of simultaneous accesses on a "round-robin" basis. It blocks the other requests and activates the microprogram to service the request.

The store blocks are either core-stores or semiconductor stores. Due to the semiconductor stores having the advantages of lower cost, higher speed and packing density they are becoming the more attractive form of main memory. The volatility problem of semiconductor memories is overcome by using ROM or PROM to store the essential parts of the operating system that deal with re-start and re-load from backing store. An added advantage in this case is the protection of the vital programs and data against corruption by noise transients and software bugs. This 4K memory module is provided on a per-CPU basis and treated as part of the CPU security unit. However, the addressing and protocol are the same as for the main memory blocks.
The protocol between CPUs and I/O blocks is the same as that between CPUs and store blocks. I/O blocks are used for communication between CPUs and peripherals. There are three types of I/O blocks in the system:

(i) Application I/O block
(ii) PU private I/O block
(iii) DMA multiplexer I/O block

An application I/O block allows addressing up to 4K peripheral units. The peripherals are divided into 16 application groups (channels) each group having up to 256 peripherals (sub-channel). Thus to a CPU a channel is equivalent to a page and a sub-channel to a word.

The private I/O block gives CPU access to CPU access ports, the interrupt unit, teletypes and paper tape units.

A DMA multiplexer I/O block is the same as an application one except that some buffer boards not required are removed because of the proximity of the DMA multiplexer with its own I/O block.

The DMA multiplexers are capable of addressing and selecting up to 240 peripheral controllers. The peripherals can be connected to both I/O channels for direct CPU control, and to peripheral controllers for DMA. To save on channel equipment, the direct CPU control can be performed via the DMA multiplexer. The multiplexer will have its own I/O address within this block, so that the CPU can address the DMA multiplexer itself. Neglecting parity bits, the address highways from CPUs to the store and I/O blocks are 20 bits wide and the DMA multiplexer address highways are 12 bits wide. Separate in and out data highways each transmit two bytes with a parity bit per byte.
The implementation of interrupt facilities for a multiprocessor system has its special problems (WARD 72A). The interrupt system must be independent of the CPUs and be capable of deciding which CPU to interrupt. The interrupt unit is triplicated for security reasons. The interrupt system is used to stop a CPU from running a process and to cause it to access external devices when they need service. Devices send their interrupts to the interrupt unit where they are detected and stored as levels in appropriate groups. There is a maximum of 8 groups and 16 levels to a group.

Interrupt signals to the interrupt system are classified as immediate or non-immediate. Immediate interrupts are processed immediately while the non-immediate ones have to wait for the arrival of an immediate interrupt. The real-time clock is an example of the immediate interrupt. The longest time that a non-immediate interrupt has to wait is a clock period (10 msec).

The interrupt unit contains a register that holds the identity of the CPU running the lowest priority process (LCPU). This register is updated by the process allocators when scheduling processes to run in their respective CPUs. When an immediate interrupt arrives, the interrupt system sends an interrupt to the LCPU and starts a time-out. If a reset signal is not received before the time-out expires, the interrupt system notes this fact and sends an interrupt to another CPU. Attempts are made on CPUs cyclically until a CPU services both the immediate and non-immediate triggered levels.
All highways and registers in a CPU are 18 bits wide so that they contain a parity-bit per byte for security. The CPU contains an exceptionally large number of registers not found in comparable commercial computers. Registers can be divided into two groups; scratch pad registers and control registers. Scratch pad registers are those that can be addressed by a programmer and there are 16 of them. The arithmetic unit performs arithmetic and logical operations. These include addition, subtraction and multiplication with division as an optional extra, inversion, AND, OR, exclusive OR and 'INVERT and AND'. It also performs bit manipulation, shifting and searching.

The heart of the CPU is the microprogram. It interprets the instructions and provides the control stimuli to the highways, registers and the arithmetic units. It also performs virtual-to physical address translation and contains the process allocator, the central part of the operating system to be described later.

The functions of the DMA multiplexer can be summarised as, firstly, allowing peripherals to access main memory through their peripheral controllers without the need for having separate ports on each store for each peripheral controller. In this sense, the multiplexer acts as a concentrator for the DMA traffic. Secondly, the controller allows individual CPUs to input directly to peripheral controllers, without the need for the controllers to have separate ports for each CPU. In this respect, it is acting as an extension unit on the store highway. Thirdly, it passes on interrupt requests from the peripherals through their controllers to the interrupt unit. Thus, the multiplexer has interfaces to the main store blocks, the peripheral controllers, highway, CPUs and the interrupt unit. It is microprogrammed and at least two multiplexers are provided in an installation.
The peripheral controllers are application-dependent devices, acting as an interface between the application requirements and the DMA multiplexer protocols, as for example in the drum controller.

For backing store on the Mark II BL system, fixed-head-per-track magnetic drums or discs are provided. Each drum is divided into 256 tracks with 8 sectors per track. A sector holds a page of information. The size of the backing store normally depends on the application. For example, two small 1.5 Mbit drums were sufficient for the CCITT No. 6 signalling system. All the fetches and dumps are done via the peripheral controller and DMA multiplexer.

For security purposes extensive fault diagnosis and recovery facilities are provided. A console is provided with each CPU to monitor the various register contents (including microprogram registers) and extensive facilities to aid in fault diagnosis. A fault channel is also provided for automatic recovery. The trap process in the faulty CPU enables this channel and sends an interrupt to the system. The CPU servicing the interrupt then runs extensive test routines and either takes the CPU out of service or restarts it.

Preprocessors may be added to the Mark II BL system to increase the computing capacity, increase the efficiency by reducing store contention and increase the mean time between system failures where a more distributed system is preferable in this respect. Whether an installation has no preprocessors, Mark II BL preprocessors or Mark IP preprocessors, depends on the size and nature of the particular application. Preprocessors are incorporated as modules into the Mark II BL system by connecting them directly or indirectly to the DMA multiplexers. The ability to add preprocessors
FIGURE (4.2) : PROCESS STATES
to the system gives the necessary flexibility in the range of computing power required by the diversity of applications for Mark II BL system intended by the British Post Office (GEC 75A).

4.3 Mark II BL Software (Figure (4.7))

4.3.1 Introduction

A modular approach is adopted in the design of Mark II BL Software as well as its hardware. Modules are processes which can be defined as entities with a unique priority and protection status. Processes are non-re-entrant, that is to say a process cannot be run in more than one CPU at the same time, and will normally have dedicated program and data which it can only use, though it may share program and fixed data with other processes. Generally, for security reasons, sharing of working storage between processes is not desirable. Processes are further divided into either operating system or application processes.

The operating system is of the inter-communicating-processes type. A clear boundary is defined and maintained between the operating system and the application software. By so doing, the operating system can be used throughout the range of applications without modification. This, also, helps to define a clear line of responsibility between those who design the operating system and those who design the application software.

Processes are function-oriented such that each process has a function or a group of functions to carry on a range of data.

In a telephony application, information about individual telephone calls is passed in tasks (Figure (4.3)) from one process to another to enable each process to perform its functions.
FIGURE (4.3) : MK II BL TASKING SYSTEM

CALLING PROCESS A
REGISTER CONTENT

G0  X
G1
G2
G3
G4
G5
G6
G7

OUTPUT TASK WITH
TASK INDEX, X

2X

ENTRY SHOWING CALLED
PROCESS B TASK INDEX, Y

CALLED PROCESS A
PROCESS DESCRIPTOR

SYSTEM & PROCESS
INFORMATION

TASK INPUT QUEUE POINTER

REGISTER NEST
SPACE

INPUT TASK WITH
TASK INDEX, Y

Link to Next Task

PRIORITY

A  Y

G1
G2
G3
G4
G5
G6
G7

10 WORDS OF
STORAGE -
TASK HANDED TO
PROCESS B

CALLING PROCESS B
REGISTER CONTENTS

G0  Y
G1
G2
G3
G4
G5
G6
G7

132
4.3.2 The Real Time Operating System

The operating system can generally be divided into two parts: the real-time operating system which runs on-line with the application software and the 'support software' used for software development. The 'support software' (which includes items such as a compiler/assembler, a linker, a lister, loaders and packages for debugging) may not be required permanently on site and is not subject to the same timing constraints as the real-time operating system.

The real-time operating system provides functions such as timing, store allocation, CPU and system scheduling, input - output buffering, message printing, diagnosis of process utility faults, as well as permitting a modular organisation of the software.

In its widest sense, the operating system includes the microprogram, and hence includes the process allocator which is implemented in microprogram. The rest of the operating system is organised into processes whose programs are stored in the main memory, backing store or private ROM per CPU (or a combination of these). In what follows, the major components and functioning of the real-time operating system is explained in more detail.

4.3.2.1 The Process Allocator

The process allocator is at the heart of the real-time operating system. It is concerned with the scheduling of the software processes to run on different CPUs, the communication between those processes and the servicing of interrupts in the complex multi-processor system environment (GEC 76A).
FIGURE (4.4): A TASK BLOCK STATE TRANSITION DIAGRAM

1. Free task

HAND

INTERRUPT

N

Selected by PA

Y

copy information in task block

task inserted in I/P queue

2. Waiting

SEEK(N)

BLOCK(N)

FETCH(N)

N

Picked from I/P Q?

Y

Task details copied out and task returned to free tasks
The process allocator is entered either as a result of a 'process allocator call' by a running process or as a result of a hardware interrupt mechanism. Interrupts are triggered by chosen peripheral devices when they require attention and by the process allocator itself when intending to schedule the system as a whole.

**PROCESS STATES:**

Processes in the system pass through a number of states while they are performing their processing functions. Basically, a process is always in one of four states (Figure 4.2). When a process has its own register contents set in the registers of a CPU, it is in the running state and will be executing instructions. A change of state is most likely to occur when a process executes a call to the process allocator or an interrupt occurs and is steered by the interrupt unit to the CPU in which the process is running. A process will 'call to block' when it is waiting for the arrival of a particular event which will be signalled by the arrival of a task of a particular priority at its input queue. In this case, provided that the task has not arrived yet, the process allocator will change the state of the process from running to blocked with a parameter between 0 and 15 to indicate the priority of the expected unblocking task. Thus, there are virtually fifteen states within the blocked state, that is from block (0) to block (15).

When an interrupt occurs on its CPU, the process state will be changed from running to suspended (Interrupted). When an unblocking task occurs, the process will then be changed from the blocked state to suspended (Unblocked). The differentiation between the two types of suspended state determines the degree of de-nesting later when the process allocator decides to run the process. A process is also
FIGURE (4.5) : A PROCESS STATE TRANSITION DIAGRAM

1. RUNNING

2. SUSP(INT)

3. SUSP(UNBL)

4. BLOCKED

5. TRAPPED

A Process
trapped

A Process
blocked

Susp.Q.
Int.

Selected
To run?

\[ \text{unblocking task} \]

\[ \text{Task Found?} \]

\[ \text{Block Process} \]

\[ \text{Untrap} \]

\[ \text{Full Nest} \]

\[ \text{Interrupt} \]

\[ \text{Trap} \]
transformed to the suspended state when another process calls the process allocator to untrap it. Finally, a process state is changed from Running to Trapped if a trap is detected by the process allocator or the process calls to be trapped.

**TASKS:**

The process allocator maintains a pool of blocks of words, each block is 10 words long, and it uses these blocks as message or task carriers between the different processes (Fig. 4.3). The process sending the task is called the 'calling process' and that receiving it 'the called process'. The first word in the task block is used to store its location with respect to other task blocks. The second word stores the task priority which determines its position in the process input queue.

The third word is used to store an index supplied by the calling process and referred to as the *incoming* task index. This index is used by the process allocator to index down a per-process table to extract information such as the identity of the called process, the task priority and the value of the incoming task index which will be passed to the called process. Thus, in general, the task index changes as the task is handed from one process to another. This gives processes the freedom to choose task indices best suited to them and reduces the chances of changes in one process affecting the others. The rest of the task block (Words 4 - 10) are used to store the information written by the process into the general registers G1 - G7.

**CALLS TO THE PROCESS ALLOCATOR** (Refer to Figures (4.6.1) and (4.6.2))

A process wishing to pass information to another process, will load
the information into the general registers $G_1 - G_7$, loads an outgoing task index in $G_0$ and executes the instruction $HAND$. This will activate the powerful microprogram sequence of the process allocator. The process allocator then links a free task block out of the free tasks list and stores the information as mentioned under tasks. It then links the task to the process input queue at a point behind all tasks of equal or higher priority. Priorities are in the range $0 - 15$; the lower the number, the higher the priority. If the called process is in the blocked state and the task just handed is unblocking, then the process state will be changed from blocked to suspended (unblocked). It will also be inserted in the suspended state map so that it may be selected to run later. The process allocator then, checks the priority of processes running on other CPUs. If the priority of the process just suspended is higher than the priority of any of the running processes, the process allocator will trigger a special interrupt level, known as the 'suspended queue interrupt', in the interrupt triplicates unit. This will force the multiprocessor system to re-schedule itself, as will be explained later in the interrupt handling sequence of the process allocator.

If a running process requires a task of a minimum lower priority to be linked out of its input queue and loaded into the general registers, it will execute $FETCH(N)$. The process allocator will check the priority of the tasks in the input queue. If a task of priority higher or equal to $N$ is found, it will be linked out of the input queue.

The information will be loaded into the general registers and the task block returned to the pool of free task blocks. The process allocator then sets the condition codes to 1 or 0 according to whether the required task is found or not, and exits back to the process.
FIGURE (4.6.1): PROCESS ALLOCATOR STATE TRANSITION DIAGRAM (STD) - LEVEL 0

1. IDLE
   - HAND
   - INTERRUPT
   - BLOCK (N)
   - FETCH(N)
   - SEEK (N)
   - SELF (M)

2. RUNNING
   - Perform appropriate action
   - Performed
   - Exit to Process

KEY
- EVENT
- INTERNAL ACTION
- EXTERNAL ACTION
- STABLE STATE
Thus, the calling process will always continue from the next instruction and will know whether the task is fetched or not by scanning the condition codes.

SEEK(N) is identical to FETCH(N), except that a task of a particular priority, $N$, is required in this case. As tasks are ordered according to their priorities, the process allocator has to examine the whole queue of tasks to see if the requested one exists. In case of FETCH(N), only the first task in the queue need be examined.

If a process executes BLOCK(N) and a task of priority greater or equal to $N$ is found, the sequence is the same as that for FETCH(N). If, however, there is no task in its input queue of priority at least $N$, the process allocator nests the process's non-general registers into its process descriptor and sets it blocked at level $N$. It will remain blocked until it is handed a task of priority at least equal to $N$. While it is blocked, it is not allowed to run on any CPU. The process allocator will then select another process to run in its place, from the pool of suspended processes.

A process may hand a task to itself using the instruction SELF(N) where $N$ is the priority of the task. The called process is identical to the calling process. The incoming task index is copied straight from the outgoing task index. Hence, the process allocator does not need to access the process's task index table. The process has, also the other option of handing a task to itself using the instruction Hand where the translation using the TIT (Task Index Table) will result in a called process index identical to that of the calling process.
FIGURE 4.6.2: PROCESS ALLOCATOR STATE TRANSITION
DIAGRAM (STD) - LEVEL 1
A process may also call to TRAP. The process allocator then nest all the registers (scratch pad) and sets the state of the process to trapped in the process descriptor. A special task will be loaded and handed to the storage allocator or the trap process according to the nature of the trap. The process allocator then re-schedules on this CPU. Since the process load does not increase, there is no need for system scheduling using the suspended queue interrupt mechanism.

Processes in master mode may call the process allocator to UNTRAP another process. When the process allocator is entered, it checks whether the calling process is in master mode; if not, it branches to the trap sequence mentioned above. The state word of the trapped process is set to suspended (interrupted) in its process descriptor and the process is included in the suspended state map. The process allocator then performs a system scheduling.

In general, processes are either periodic or non-periodic (or aperiodic). A non-periodic process will pick up all its tasks using a BLOCK(N) instruction. Thus, subject to CPU availability, it will process its tasks as they arrive. To reduce the process allocator Lockouts occupancy, several processes are made periodic. Such a process will handle all its tasks when running. When its input queue becomes empty, it is blocked and incoming tasks are piled up into its input queue and not handled until the process is unblocked by the arrival of a periodic unblocking task.

**TASKS TO AND FROM PERIPHERAL PROCESSES**

Tasks handed from processes in the processor utility (PU) to peripheral processes are indicated to the process allocator by the fact that the called process index is out of range. The process allocator then links the task to the peripheral process input queue and performs
FIGURE (4.7): MK II BL SOFTWARE STRUCTURE
a 'kick-off' I/O. The process allocator performs this 'kick-off' I/O to cause the peripheral process (or controller) to search down its queue for new tasks. When the peripheral controller wishes to hand a task back to a PU process it alters the task details, writes in a new outgoing task index, clears the priority word and triggers an interrupt. When the process allocator is serving an interrupt it will look down the input queues of the peripheral controllers corresponding to the triggered levels. On finding a task with cleared priority, the process allocator links it out of the input queue and hands it to the appropriate PU process using the outgoing task index to index down the peripheral controller task index table.

**TASKS BETWEEN THE PU AND BACK-UP STORAGE**

Mark II BL storage devices, such as drums or discs, have peripheral controllers associated with them. Some PU processes such as the storage allocator, communicate with these by passing tasks to effect the transfer of information between working and backing storage. The peripheral controller uses a DMA multiplexer to effect the transfer which will trigger an interrupt when it is completed.

**TASKS BETWEEN THE PU AND PPU**

To the PU, the PPU looks like a peripheral controller with one or more peripheral processes. The peripheral controller responds to the 'kick-off' I/O and triggers an interrupt to the PPU. The PPU process allocator performs I/Os to load the task into registers and uses the task index table to index down a TIT in its own data (one TIT per peripheral process) to hand the task to a process on the PPU.
INTERRUPT-SERVICING

In the interrupt unit there exist interrupt levels grouped into four groups 0 - 3. Group 0 comprises the DMA interrupt levels which are dealt with by the process allocator itself. For Group 1 (Timeouts and Clock), Group 2 (CPU and DMA faults) and Group 3 (Man/machine) the process allocator hands these to INTIM process to deal with. They are called ordinary interrupts.

Interrupts are also classified as immediate or non-immediate interrupts, regardless of whether they are DMA or ordinary interrupts. An example of an immediate interrupt is the clock interrupt which is triggered every 10 msec and the suspended queue interrupt.

When an immediate interrupt is triggered, the interrupt unit sends a signal to the CPU running the lowest priority running process. The process allocator in the interrupted CPU finishes performing its current instruction and branches to its interrupt servicing sequence.

The process allocator nests all the registers of the current running process and sets its state to suspended (interrupted) in its process descriptor. It will also include it in the suspended state map. The process allocator then performs I/Os to staticise the triggered interrupt levels as bits in the CPU registers. It then checks whether any of the DMA levels are triggered and if so services them. The ordinary interrupt levels are stored in a task and handed to the INTIM process. The process allocator then selects the highest priority suspended process to run on this CPU, deletes it from the suspended state map and sets its state to running in its process descriptor.
According to whether the selected process was suspended (interrupted) or suspended (unblocked), the process allocator de-nests the general registers from its process descriptor or loads the first task from its input queue respectively. In either case, the non-general registers which store the environment of the process are de-nested from its process descriptor.

The interrupt is steered by the interrupt unit to the CPU running the lowest priority process. This is known as the LCPU. Whenever this value changes, the process allocator involved must inform the interrupt unit.

After sending the interrupt to LCPU, the interrupt unit starts a time-out. If LCPU does not send a reset within this time-out, the interrupt triplicates unit sends an interrupt to the next CPU and starts the time-out. This pattern continues until one CPU responds within the time-out. When a CPU responds, it checks whether it is actually the LCPU. If not, it services the interrupt in the normal way, but hands a modified ordinary task to INTIM specifying the value of LCPU.

**SCHEDULING**

A main function of the process allocator is to ensure that a process is allowed to run only when all higher-priority processes have run.

In the process allocator data, a two-dimensional bit array defines the indices of processes eligible to run on that CPU. The process allocator always checks this when selecting a process to run. In general, the majority of processes may run on any CPU. However, TRAP processes whose function is to investigate CPU faults and background processes to monitor store locations are assigned on a fixed per-CPU basis.
When a running process is blocked, that is, it has executed a BLOCK(N) instruction and a task of priority N or higher is not found, or when it is trapped or interrupted, the process allocator must select another process to run in its place. It does this by searching down a bit map where the bit position indicates the value of the process index (The Suspended State Map). Finding a suspended process, the process allocator then checks whether that process is allowed to run on that CPU, and if not, continues searching until it finds one that is allowed. The process allocator then sets to zero, the corresponding bit in the suspended state map and changes the state of the process to running in the process's process descriptor. It then compares all the CPUs, finds the new value of LCPU and sends it to the interrupt unit. The general registers are de-nested from the process descriptor if the process was suspended (interrrupted) and a fresh task loaded into the general registers if it was suspended (unblocked). Regardless of the type of suspended state, the non-general registers, which store the process environment are always de-nested from the process descriptor.

When a process allocator executes a HAND instruction, services an interrupt or untraps a process, it is highly probable that a high-priority process becomes suspended. It is desirable to get it running as soon as possible instead of waiting for a lower priority process to be blocked. For this purpose, a facility is provided for a process allocator to trigger a special interrupt the 'suspended queue interrupt'. The process allocator compares the priority of the highest priority suspended process with that of the process running on LCPU. If it is higher, it triggers the 'suspended queue interrupt'. This is an immediate interrupt and is steered to LCPU, forcing its process allocator to re-schedule. This sequence
is repeated if necessary until there is no suspended process of priority higher than that running on LCPU. This system scheduling using the 'suspended queue interrupt' mechanism is, therefore, usually employed when the process load increases.

OVERLOAD CONTROL

This is another of the process allocator responsibilities. There are two distinguishable overload conditions: system overload and process overload. A system overload occurs with high traffic when is not able to handle calls at the rate they are arriving. This can, ultimately, lead to system rollback when the process allocator free task list becomes empty. A process overload occurs when a process cannot process incoming tasks at their arrival rate, either because of its low priority or a hardware fault. For example, if the storage allocator receives a large number of tasks when a store block fails its input queue length increases considerably.

When the process allocator detects an overload condition, it hands the task that causes the overload normally, but also hands a special task to an overload control process. Subsequently, when the process input queue drops below a certain level or the free task list grows above a certain level, the process allocator sends another task to the overload control process to discontinue its remedial action.

LOCKOUTS

Lockout or semaphores are used to protect the critical data of the process allocator from simultaneous illegal accesses. Three lockouts exist within the process allocator. The suspended state lockout, SUSPLO, is held while including a process in the suspended state map or when removing one from it.
The free-task queue lockout, FREELO, is engaged while removing a free task block from or inserting a task block into the free task list. The interrupt lockout, INTLO, is engaged when accessing the interrupt triplicates unit to identify the interrupted levels, updating the value of LCPU or triggering the suspended queue interrupt.

PROLO, is a per-process lockout and is engaged while linking a task into or out of the process's input queue. PROLO is also engaged while interrogating the state of the process and its input queue and while taking it into or out of a blocked state.

It is crucial that lockouts are held to the minimum. One of the objectives of simulating the process allocator is, in fact, to study the lockouts' occupancy under different circumstances, as it is crucial to the optimum performance of the process allocator that the lockouts are held for the minimum possible time.

4.3.2.2 The Interrupt and Timing Process (INTIM)

The responsibility of this real-time operating system process is to deal with external interrupts (as opposed to traps) and associated timing. This can be stated more precisely as (GEC 75B):

a) Timing and Interrupt Handling

INTIM maintains a calendar which is constantly being updated on clock interrupts. In response to timing tasks, INTIM sends time data in the response task. INTIM uses the clock interrupt to perform timing operations for processes. For example, a process may request a response task to be sent after a specified time interval or the occurrence of a particular interrupt. A process may also request periodic response tasks with specified periodicity to be sent or request to cancel the actions requested by tasks involving timing.
b) **Interrupt Unit diagnosis and re-configuration**

Whenever a process allocator decides that a fault has occurred in the interrupt system, it sends a fault task to INTIM which will identify the cause and isolate the faulty unit among the interrupt triplicates and informs the process allocator.

c) **I/O Devices handling**

I/O man/machine communication is handled by INTIM in conjunction with two other processes, TYPER and CANAL, that is INTIM simulates DMA for man/machine communication. INTIM services I/O device interrupts and passes the characters input in a task to the CANAL process. Output tasks specifying device number and address of the buffer of characters for output from TYPER processes are sent to INTIM. The characters are outputted to the appropriate device under interrupt control. Thus, when a process wants to wait for an interrupt, say, it sends a task to INTIM specifying the source of interrupt and generally, a covering time-out, so that if the interrupt does not occur within this time-out, the process will be informed. The process may call to block waiting for the task such as a periodic process. In this case, when the interrupt occurs, the process allocator forms a task containing the identities of processes waiting for that interrupt and hands that task to INTIM which unblocks the specified processes.

Timing of the actions of a process can be achieved in a number of ways. In periodic processes, timing is generally done by counting the initiations of the process. In a non-periodic process timing can be done by the use of clock tasks. A further method of timing is available through the observation of a real-time clock count, maintained by INTIM.

4.3.2.3 **The Storage Allocator**

This real-time operating system process is responsible for the storage management. Generally, one can identify three main functions for this process.

One of its most important functions is to deal with requests for pages not in main memory. A process will be trapped, that is to say nested and stopped running, if it tries to access a page not in memory. The process allocator then hands a task with the page details to the storage allocator. The storage allocator accesses the system page table to see whether memory space has been allocated to this page. If not,
an overwrite search is commenced to select a page to be overwritten.
If the selected page status is read/written it has to be dumped to
the drum. The storage allocator then generates a task containing
the necessary track and sector information and hands the task to
the appropriate drum using the DMA task facility of the process
allocator. When the page transfer is complete, the trapped
process is untrapped allowing it to continue.

The storage allocator also handles request tasks from processes.
When a process does not want to be trapped by a 'not-in-core' trap,
it handles a request task to the storage allocator to fetch a page
from the backing store. When that is done, the storage allocator returns
a response task back to the process. This procedure allows a higher
throughput for the process.

A process may, also send request tasks to the storage allocator
to open or close one of its files. A file is a group of pages
with consecutive virtual addresses, each page having the same status
as a process. When a file is occupying part of the virtual address
range of the process, it is said to be open. A process usually
has three open files; the execute file, a read-only file and a
read/write file. If the whole file does not need to be accessed
or a file that belongs to a different process is required to be read,
then the part-file mechanism may be used. The part-file request to
the storage allocator always causes a copy of the information to be
taken from or written to the requested file, that copy being
transferred to or from an open read/write file of the requesting
process. Information transferred can be any number of consecutive words
up to a page, or a number of pages.
The combination of the 'Trap not-in-core' and the requests for pages and files gives the required flexibility and control in the management of the store while permitting the efficient use of both the main and backing store. The storage allocator brings into core the minimum number of pages to allow a process to run, and pages of low priority are only brought in when requested.

4.3.2.4 Thrash and Crash Processes

These are a suite of processes collectively known as RASH primarily provided to assist in debugging and commissioning Mark II BL system. RASH provides the man/machine interface and process data (file data, task index tables, process descriptor) for a user to control and run test modules. The control facilities to the programmer include starting and stopping the test module and timing it. The test module used for performance or functional testing can be run by one RASH process only or by up to six processes running at the same time. Each of the RASH processes, known as a test process, is controlled individually using the commands of the man/machine language.

An additional facility available to the programmer is the ability to pass data items to the test module before it starts running. A programmer can also link a number of test modules in a chain to execute consecutively and store within RASH tasks to be handed to other processes periodically or one-shot only, for debugging purposes.

The test modules include general purpose test-modules which test various aspects of the system. These could be run at any time to exercise the system, and thereby provide a heavy workload for system testing.
Each test process has an execution counter which is incremented every time the process executes the test module and this facility is also used for timing test modules against periodic timing tasks.

4.3.2.5 Other Operating System Processes

Operating system processes described in the previous sections of this chapter are those most important and relevant to this simulation research project. Other operating system processes are, therefore, not described here. These include CANAL process and TYPER process which deal with input/output respectively. TRAP process which handles software traps, CAP (CPU Access Ports) process for faulty CPU diagnosis, background process and others.
THE MARK II BL SYSTEM SIMULATOR

5.1 INTRODUCTION

As already mentioned in Chapter 2, the advantages of using SIMULA are related to the concepts provided by the language. At the heart of those concepts is the CLASS concept, a generalisation of the PROCEDURE concept in ALGOL 60. Also, the ability to prefix classes and blocks by other classes, so that they inherit their attributes and actions, allows the construction of hierarchical tree structures. This feature makes the language extendable and application oriented. In fact, that is how the list processing and simulation concepts are developed to convert the general purpose SIMULA language into a process-based simulation language.

SIMULA is a language for sequential processes which can only be executed one instruction at a time (BIRT 73). To enable the simulation analyst to model truly systems with several simultaneously operating processes, SIMULA has mechanisms which create the illusion of parallelisms, viz. quasi-parallel programming. This is a common term for the mode of operations of co-routines. Only one routine is executed by the computer at any one time, and the sequential execution of a routine is only interrupted at a sequencing (or scheduling) statement. With respect to the sequential co-operating processes definition (DIJK, 68), a quasi-parallel system may informally be defined in either of the following ways: (PIEN 73).

the sequence of operations between a sequencing statement and the next is a critical section, i.e. their executions are mutually exclusive.
in a process an atomic (indivisible) operation may be defined at will, as a sequence of non-sequencing statements (preceded and terminated with a sequencing statement).

Thus a SIMULA process is executed in 'chunks', each chunk representing an activity initiated and terminated by events represented by the execution of sequencing statements. During an activity, a process is active and interacts with other system components (processes) resulting in a change of system state. Since simulated time is only changed using the sequencing statements, an activity does not consume any simulated time.

The system to be simulated (Mark IIBL) is a multi-processor multi-programmed computer system. In a multi-programming system, the processes are executed, one at a time, by one processor. The active phases of the processes and their scheduling sequence are dynamically defined externally to the processes (i.e. by an operating system). With reference to a system of sequential processes accessing common storage, the atomic operations in the multi-programming system are the indivisible (by hardware) store and fetch instructions. For a multi-processing system, the atomic operations are the same, but the active phases of the processes are of infinite length, since each process is executed by a separate processor. By contrast, in a quasi-parallel system, the atomic operations are defined at will within a process and coincide with the active phases. Thus quasi-parallelism is the most general of the three concepts, in the sense that the logical operations of the other two systems, or a mixture of them (such as Mark II BL), may be described by a quasi-parallel system. Hence, it is quite possible to construct an 'exact' model of Mark II BL system in SIMULA using its quasi-parallel programming approach.
5.2 DESIGN PHILOSOPHY OF THE SIMULATOR

The simulator follows the same modular approach of Mark II BL software and hardware. It is structured for ease of debugging and understanding by those using the package for software development. For the same reason, the simulator must model the system and its data structures in a one-to-one translation. This should result in a high correspondence between the model and the actual system.

The simulator ought to mimic in great detail the actual system, in order to carry out useful investigations and experiments into the system's resource management policies and structures.

On the other hand, the simulator must be general enough to be used in conjunction with different application software structures for different exchange types. These two conflicting requirements call for a careful choice regarding the level of detail of each component in the simulator. While infrequently used processes, procedures and data structures are discarded, others whose operations are important in reproducing the behavioural characteristics of the system are explicitly modelled.

The simulator must retain the same interfaces between the system software modules or processes and must allow models of the application processes to use the same calls to the process allocators as they use in the actual system, for communication and task passing.

The transformation of a software process to its corresponding model must be made as simple and as easy as possible for the software design engineer. All that is required from him or her to do is to strip the different activities of a process of the actual codings representing those activities and replace the codings by the actual
The timing information of the different activities or missions within a process is readily available and is usually calculated from the number of instructions executed and the computer cycle time.

The simulator must allow the interrogation of the simulated CPU registers, e.g. the general registers G0 - G7, and allow the follow-up of individual messages and hence telephone calls.

It must permit the modelling of the system components at different levels of detail and the expansion of a process model from a macro to a micro level without affecting other parts of the simulator. The modular nature of the simulator must facilitate building up a library of software models, such that a software engineer can easily assemble the model structure of a particular application from that library.

The simulator must allow the software design engineer to input parameters, such as the simulated time and the number of CPUs in the configuration, on-line and obtain the run-result in a fairly short time.

The simulator must have a tracing facility that is triggered when required to give detailed system interactions and global table contents.

5.3. PROBLEMS IN SIMULATING A MULTI-PROCESSOR SYSTEM

A multi-processor system may be defined as a system possessing one or more sets of resources, with one or more identical members in each set, operating under central stored-program control and capable of processing one or more programs at any given point in time. Although
much money was and is being spent in this sector, yet there is little basic understanding of the components interactions and the internal working of the system (HUTC 73).

Unlike job-shop simulations, simulations of multi-processor systems have to take account of some additional complications. Frequently, in a multi-processor, a program's routine requires resources of several different types simultaneously. and, having once captured them, it must continually compete with all current routines to maintain their possession. In contrast, for a job-shop situation, once a job has acquired a resource (e.g. machine) it keeps the resource for the entire machining time without interruption. Whereas in job-shop simulations, the life cycle of a job is simply a linear list of a few entries in which it acquires resources and releases them, in a multi-processor system each incident of conflict between the active competing components must be modelled. Hence, it is necessary to describe each routine of each program to a degree of detail determined by the objectives of the simulation.

The resource management policies in a job-shop are easily simulated by routines for major functions such as assignment of jobs to machines. In the multi-processor system, the management functions are more complex and are performed by software, the operating system, which comprises programs that compete for and consume resources which might otherwise be assigned to workload programs. The time requirement of the operating system to carry its management functions can be highly significant in the multi-processor system.
Thus, due to the difficulties such as those mentioned above and the necessity of handling both minute details and higher-level activities, in addition to the more conventional requirements, some of the problems of systems simulation in general are emphasized for multi-processor systems. One of the problems is the degradation of the ability to differentiate between times of events occurring. This problem is particularly associated with discrete-event simulation languages that maintain a monotonically increasing master clock for the simulation e.g. SIMSCRIPT, SIMULA. Languages that use the concept of relative timing, e.g. CSL (BUXT 62) are free of this problem. In such languages, future events occurrence times are calculated relative to 'now' and stored in 'time cells' associated with the events. 'Now' means simulated time zero. Thus an event with the smallest time-cell value is the next event to occur.

On the other hand, event times for monotonically-increasing master-clock types of languages are calculated during the course of the simulation with added to the master clock time \( t_m \) to schedule the time of the occurrence of that event \( t_e \). The master-clock time value is stored in a finite number of bits, typically one computer word and floating point notation. At any \( t_m > t_e > 0 \), where \( t_e \) is the time to elapse for the occurrence of the event \( e \), there will be a \( t_e \) which is so small such that

\[
t_m + t_e = t_m
\]

This value of \( t_e \) designated by \( \hat{t}_e \) could be defined by the following equation (HUTCH, 73)

\[
\hat{t}_e = t_m / b^n
\]
where

\[ \hat{t}_e \]

is the largest \( t_e \) which can be added to \( t_m \) without increasing \( t_m \).

\[ t_m \]

the master clock time

\[ b \]

the number base of the computer

\[ n \]

is the number of bits used for the mantissa of the floating point number.

In ICL SIMULA implementation on System 4 - 72, the current simulated time can be retrieved using a SIMULA SYSTEM - defined procedure TIME. This procedure is of the type LONG REAL, i.e. the timing information is stored in two computer words. A system 4 - 72 word is 24 bits long and a REAL number has 9 decimal digits. Therefore,

\[ \hat{t}_e = t_m / 2.39 = 1.82 \times 10^{-13} t_m \]

If the time unit (or grain) is taken as 1 micro-second, then the maximum length of time to which the simulation could be run before degradation occurs is

\[ t_m = \frac{10^{13}}{1.82} \times \frac{10^{-6}}{3600} = 1.525 \times 10^3 \text{ hrs.} \]

This \( t_m \) value is well above any anticipated period for which the simulator may be run. The danger of having \( t_e < \hat{t}_e \) is to force the simulator into an endless loop for a recurring event with the simulated time remaining unchanged as \( t_e + t_m = t_m \). A second danger stems from the fact that, with the degradation of the ability to differentiate between events as \( t_m \) increases, the accuracy of the measurement system changes as the simulation progresses, contributing to the degradation of the simulation results. The penalty of using a double-precision word to store the timing information is a slight increase in execution time.
This time-grain problem normally occurs in hierarchically-structured models of different layers, with each layer being at a different level of detail (Figure 1.2). With reference to Figure (1.2), the micro level, level 1, represents the model of the operating system, where the process allocator is described at a micro-level with a time grain of one microsecond. This is thought to be necessary in order to carry out simulation experiments with the scheduling algorithms, lockout occupancy and interrupt servicing policy. The other processes that communicate with the process allocator are either at the intermediate or macro level. The time grain at the intermediate level is typically a millisecond and at the macro level is a second.

An alternative solution to the differences in time grains of a multi-level hierarchically-structured model is to summarise the performance analysis results of level 1 in a form easily usable in the next layer above, i.e. level 2 (KOBA, 78). Usually a scaling factor or a random variable with some distribution is used as summarised statistics for an interface of two layers of different levels of details.

Another problem in simulating multi-processor systems is that of the execution of simultaneous events in a sequential machine. Again, this problem is not unique to multi-processor systems, but the fact that the simultaneous-events density is higher in the case of multi-processors, especially when investigating resource allocation algorithms, makes them a candidate for the category of 'worst case'. The problem with the execution of simultaneous events sequentially is that the order in which events are executed depends on the simulation system itself and not the system under study, e.g. the simulation system (language) may adopt the algorithm that the first
event placed in the event list at this time shall be executed first. Under these conditions, the outcome of running the same model with identical data on different computers could easily be very different (HUTC 68). This problem could escalate when some simultaneous events schedule further simultaneous events and so on. A number of techniques are suggested to deal with such situations, e.g. the use of a suspense set for simultaneous events. This problem presented a major challenge in the simulation of the interrupt-handling mechanism of the Mark II BL system, as will be shown in Chapter 6. The simulation of a multi-processor system must not only reflect the logic of the system algorithms, but must also retain the sequence of occurrence of simultaneous events. This is achieved in this simulation by paying particular attention to the type and location of the language (SIMULA) scheduling statements.

5.4. THE SIMULATOR PROGRAM OF MARK II BL

By far the biggest process in the simulator program is that of the process allocator (Figure (5.1), 719-1713). As mentioned before, this is because the process allocator is central to the real-time operating system and a detailed study of the system's resource management policies requires a detailed simulation of the process allocator. The other simulation modules (processes) representing the other system hardware components and operating system processes are:

- CPU
- TASKBLOCK
- AP
- INTIM
- INTRIPLICATES
- CLOCKINTERRUPT
- SA (Storage Allocator)
- RASH
- BACKGROUND

The numbers in brackets refer to the listing of the System X simulator package in Appendix B.
FIG (5.2) PROCEDURES FOR THE PROCESS ALLOCATOR MODEL

A TASK BLOCK

<table>
<thead>
<tr>
<th>NO.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVITY</td>
<td>END-OF-CALLED-PROCESS</td>
<td>TASK PRIORITY</td>
<td>CALLED PROCESS INDEX</td>
<td>TASK DETAILS</td>
<td>TASK DETAILS</td>
<td>TASK DETAILS</td>
<td>TASK DETAILS</td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

A - ACTIVITY
5.4.1 The main program

The main program contains definitions of the global parameters used by the process allocators on different CPUs and other processes. The suspended-state map (where the identities of suspended processes are stored) is represented by a BOOLEAN ARRAY SUSPMAP (0:127) \[(155)\]

is an example of global tables. The maximum number of software processes (both operating system and application processes) is not expected to exceed 128, hence the size of the array. To reference processes regardless of whether they are operating-system or application processes, a reference array is defined, REF(AP)ARRAY P(0:127) \[(160)\]. AP is an entity which prefixes all software entities in the simulator.

Global parameters include the lockouts FREELO and SUSPLO, defined as resources for which the process allocators compete. FREETASKLIST is the queue for free task blocks. LPAQ is a queue where the interrupt triplicates model or entity waits for the process allocator on LCPU to finish servicing its process allocator call or interrupt, before sending its new interrupt to that process allocator.

The simulated time for a run (SIMPERIOD) and the number of CPUs in a configuration (NUM) are global parameters that are set by the user before starting the simulation \[(144, 146)\]. Pointer \[(151)\] is used to point to successive rows of the periodic processes table. The pointer is initialised to row No. 1 \[(4080)\] and is incremented or reset to 1 by the process allocator servicing the clock interrupt \[(1197, 1202)\]. Some global procedures to assist in the outputting of reports and error messages are also defined \[(171 - 213)\]. The main program creates instances of the entities in the simulator and refer to them by certain defined reference names e.g. \[(3929)\] or by the REF ARRAY P, \[(3931)\].
The task index tables of the entities are then initialised. The entities themselves are initialised as being in the state BLOCK(15) i.e. waiting for a task of priority equal to 15 or higher. For every process model or entity an input queue (INPUTQ) is defined to hold incoming task messages, and a lockout, PROLO (defined as a resource), to control the access of the process allocator to the input queue.

The main program then creates 100 task block entities and links them to FREETASKLIST. It then holds or waits for the simulated time to elapse (4085) before outputting a summary report for the run. When the main program suspends itself, it arranges for program control to pass to the clock interrupt entity (4084).

5.4.2 The Clock Interrupt (1717 - 1761) (Figure (5.3))

This entity generates instances of the CPU entity, the number of instances corresponding to the number of CPUs in the configuration and determined by the global parameter NUM. The value of NUM is either read in from the input data file or typed in by the user at the start of the simulation run. For each CPU, a background process model (i.e. AP CLASS BACKGROUND) is created with its state set to running (1742) in its process descriptor. The instances generated are then activated (1750 - 1751).

The clock interrupt entity then selects the CPU running the lowest priority background process as the lowest priority CPU, LCPU, and transmits its identity to the interrupts triplicates (1753). It sets the clock interrupt flag of the interrupt triplicates to TRUE signalling the arrival of a clock interrupt, and then activates the interrupt triplicates unit. This action is repeated every 10 msec to the end of the simulation run (1754 - 1760).
5.4.3 The CPU Process (379-411) (Figure (5.4))

Most of the hardware is represented by classes containing only data or very little actions. Data pertaining to the hardware component, CPU, is a set of 16 registers, a CPU number tag, condition codes, a reference to the current running entity and its associated process allocator and some parameters to calculate the occupancy of the background entities running on this CPU. The consumption of CPU power by the process allocator entity or any other process entity is represented by the 'HOLD' statement inside the bodies of those entities. The 'HOLD' statement represents the elapse of simulated time while the respective entity is 'holding' that system resource i.e. running on the CPU. The CPU occupancy by different processes entities is calculated by the process allocator when scheduling that CPU.

The only action of a CPU instance is to generate its own process allocator. It references the process running by CURP (Current Running Process). This will be a background process at the start of a run. It also makes its process allocator reference, CALLINGPROCESS, to point to CURP. When the CPU instance exhausts its actions it is terminated i.e. will never be active again. It remains existing as a data structure (with no actions) and is referenced by REF ARRAY C(I), (158).

5.4.4 Background Process (1796-1812) (Figure (5.5))

This is the simplest of all the entities. Its sole function is to occupy a CPU when there is no other useful work to be done. There is a background entity for each of the CPUs. The simulator is initialised with all the CPUs running background entities.
5.4.5 Taskblock (415 - 425) (Figure (5.6))

This is an entity with no actions. It exists as a data structure with a one-dimensional INTEGER ARRAY of size 10 corresponding to the size of the task block in the actual system.

5.4.6 Process AP (247 - 347) (Figure (5.7))

This entity embodies the data structure common to the processes of Mark II BL and procedures which allow a process entity to communicate with other processes entities through the process allocator. Every operating system or application process model is prefixed by AP and hence inherits its attributes (by the prefixing rules of SIMULA). The data structure in AP contains a 32-word process descriptor (defined as an array) in which information relating to the entity and its environment is stored. This information is used by the process allocator in the management of the system. The task index table (TIT) is also defined on a per-entity basis. A process allocator uses an outgoing task index to index down this table to obtain such information as the identity of the called process entity, the task priority and an incoming task index which determines the response of the entity receiving the task. Also defined is an array of size 8 to simulate the eight general registers GO-G7. This is defined here to allow the simulator user to assign the details of a task to the general registers within the application process model or entity, in much the same way as he does in the coding of the real application process.

A BOOLEAN variable PERIODIC is true for periodic processes. It is used by procedure HANDTASK AND SETSTATE in conjunction with periodic processes when changing their state from blocked to suspended.
The resource entity, PROLO, is defined to control access to the input queue by the process allocators. Each entity references the CPU in which it is running and the process allocator on that CPU by RELEVANTCPU and PA respectively. The real variable, REMAININGACTIME, is used to store the remaining time of a 'HOLD' sequencing statement when the process is interrupted in the middle of the 'HOLD', so that when selected to run again, this remaining activity time is executed first.

Calls to the process allocator available to a process are represented by procedures local to class AP. These calls are 'HAND', 'FETCH(N)', 'SEEK(N)', 'BLOCK(N)', 'SELF(N)' and a recent addition 'FBLOCK(P)'. The calls 'TRAP' and 'UNTRAP' are not relevant to the simulation objectives and hence not simulated. The body of a call procedure determines a call index - an integer value used later by the serving process allocator to branch to the appropriate servicing routine, via a SWITCH statement (1144). In case of a FETCH, BLOCK or SEEK call, the CPU conditions codes are reset to zero in the body of the procedure. The last action of a call procedure is to passivate the calling process and activate the serving process allocator. PROCEDURE CC is provided so that a process can check the condition codes of the CPU on which it is running.

5.4.7 The Storage Allocator Process (429 - 498) (Figure (5.8))

This is a macro-level model of the storage allocator which models the servicing of close file, open file and part file requests from processes. The storage allocator determines the nature of the incoming request from the value of G2. A value of 4 is an open file request, and a value of 5 is a close file request. The response action of the storage allocator to such a request is to 'HOLD' for the
corresponding activity time and then send back a response task to the requesting process. The storage allocator picks up any more pending tasks by a BLOCK(10) instruction. Therefore, when all tasks in the input queue have been processed, the storage allocator is blocked awaiting the arrival of a fresh task. It is prefixed by AP and hence inherits all its attributes.

5.4.8 INTIM Process (562 - 609) (Figure (5.9))

The functions of the INTIM (Interrupt and Timing) process relevant to the simulator are to unblock periodic processes at periodical intervals and to send timing tasks to processes which require them. With the arrival of each clock interrupt, INTIM is handed a task containing the identities of the periodic processes to be activated (their process indices numbers). INTIM will most probably be selected to run because of its high priority. It starts by checking the general registers G1 - G7 for the identities of the periodic processes. The unblocking periodic tasks handed by INTIM to the blocked periodic processes are of a high priority (5) to ensure that the tasks will be linked to the front of the respective input queues. This priority number (5) is also used by the process allocator to identify such an unblocking task and subsequently change the state of the called periodic process from blocked to suspended (unblocked). After handling all unblocking tasks, INTIM calls to block awaiting the next clock interrupt.

5.4.9 The Interrupt Triplicates Process (613 - 714) (Figure (5.10))

This entity simulates the functions of the triplicated hardware interrupt unit. It is activated as soon as an immediate interrupt is triggered (1758, -1014). The immediate interrupts relevant to the simulator are the clock interrupt (every 10 msec) and the
suspended queue interrupt which is triggered whenever a system scheduling is to take place.

This entity has two BOOLEAN attributes; SUSPQINT and CLOCKINT to indicate the arrival of a suspended queue interrupt and the clock interrupt respectively. PP_TABLE, is initialised in the body of the main program to contain the periodic processes identities to be activated at subsequent clock interrupts. The process allocators access this table to copy the identities of those processes in a task to be handed to INTIM. LCPU is a reference variable pointing to the CPU entity running the lowest priority process. This reference is updated by the process allocator whenever a CPU schedule is carried out. LPA is a reference to the process allocator on LCPU. CUSP and CUCP references the current suspended entity and the current chosen entity of LCPU.

The action part of this entity starts by updating the reference to the process allocator on LCPU, as, most probably, it has changed since this interrupt triplicates/served the last immediate interrupt. If the interrupt triplicate/is activated without an immediate interrupt occurring, an error message is outputted and the entity passivates without taking any further action.

However, if the suspended queue interrupt is triggered, the interrupt triplicates/will reset the SUSPQINT flag and checks whether the process allocator on LCPU is active or not. If it is active, then it is either servicing a process allocator call or an interrupt. In such a case, the interrupt triplicates will wait in a special queue (LPAQ) for the process allocator to finish. When it is active again, it resumes its actions by setting the interrupt flag in the process allocator (LPA). It checks whether the process on LCPU is running.
This is, of course, represented by the fact that the process is executing a 'HOLD' statement to represent in SIMULA the passage of simulated time corresponding to the execution of a particular activity. If that is the case, the interrupt triplicates will calculate the remaining activity time for that process and store it in REMAININGACTIME. Later on, when this process is selected to run again, that remaining activity time is first executed. The process allocator on LCPU, i.e., LPA, is then activated to serve the interrupt and the triplicates unit passivates awaiting the arrival of the next immediate interrupt.

On the other hand, if the immediate interrupt is that of the hardware clock, the action of the triplicates unit is the same as for the suspended queue interrupt, except that the CLOCK flag is set on LPA instead of SUSPQINT.

In the interrupt triplicates unit of the real system (MK II BL) no consideration is given to detect a particular sequence of events which might lead to an unnecessary servicing of an interrupt (and hence reduce the system throughput). Such a sequence of events was noticed from close examination of a detailed output trace during the verification and validation stages of the simulator. An example of such a sequence of events is the case where the interrupt triplicates, being activated because of a suspended queue interrupt, is waiting for LPA (the process allocator on the CPU running the lowest priority process) to finish servicing its current call. That call could as well be a call to block 'BLOCK(N)', and the process allocator on not finding the requested task of priority N blocks the running process and selects another one. It is then quite probable that the selected process is of a higher priority than the process intended to be selected for LCPU when servicing the suspended queue interrupt.
In such circumstances, after servicing the block call, the process allocator enters the interrupt servicing routine, de-nests the previously selected process, but selects it again on priority basis, nests its registers and sets it running. This unnecessary delay could have been avoided if the interrupt triplicates checks the priority of the process on LCPU against the highest priority suspended process before initiating the process allocator on LCPU to service the suspended queue interrupt. This additional feature to the actual system is included in the simulator and is found to result in an increased throughput and reduced process allocator overhead. More about that in Chapter 6.

5.4.10 The Process Allocator Process (719 - 1713) (Figures (5.1) (5.2))

This is the biggest and most detailed entity. It models the following functions of the process allocator at a micro level of detail:

(a) Handling of communication between different processes.
(b) Interrupts Servicing.
(c) Individual CPU scheduling as well as overall system scheduling.

However, this entity does not model DMA interrupts servicing nor the calls to trap and untrap a process. These are thought to be irrelevant to the fulfillment of the objectives of the simulator outlined in 5.2.

A number of procedures are defined as attributes to this entity to model specific functions. These procedures are described below.

The process allocator model is activated either by a process allocator call from the process model running on its CPU or by the occurrence of an immediate interrupt where the interrupt is steered by the interrupt triplicates unit to this CPU. In the former case, the activation is done within the procedures modelling the process
allocator calls (see 5.4.6), whereas in the latter, the activation is done by the interrupt triplicates entity (5.4.9).

When activated, control either passes to the interrupt servicing branch of the process allocator (1141) or to the appropriate call servicing routine using a SWITCH statement in conjunction with a particular call index indicating the call type (1144).

The following procedures are defined as attributes to the process allocator (797-1134):

PROCEDURE PROLO 1 (PROC): (797-814)
Checks the mutual exclusion primitive that guards the input queue of the process referenced by PROC. This primitive is referred to by the name PROLO Lockout. As mentioned before, each process has such a lockout to regulate the access of the different process allocators in a particular multiprocessor configuration to a process input queue. If PROLO is engaged by another process allocator, then this process allocator will wait until it is released in a special queue. When released, the first process allocator waiting will engage PROLO.

PROCEDURE FREELO 1: (828-845)
This procedure will check whether the lockout FREELO which guards the pool of free task blocks, is free and if not the process allocator joins a special queue and passivates until activated by the process allocator releasing the lockout. The process allocator then holds for 15.4 micro-seconds which is the time to return a free task block to the free task list. When the free task
block is returned, FREQLO is released and first process allocator waiting, if any, is activated.

PROCEDURE FREQLO 2: (848 - 873)
This is similar to FREQLO 1 procedure except that it fetches a free task block instead of inserting one. For the sake of storage efficiency, only 100 task blocks are generated - in the main program initialisation section - and inserted in the free task list (4082 - 4083). If then the freetask blocks pool is exhausted a new task is generated and inserted in the list. This is a diversion from the actual system where the number of free task blocks is fixed and determined at the initialisation stage of the system. If the free task block is exhausted an overload control process will be invoked which restores the level of the free task list to an acceptable one. Since no overload control strategy is simulated and the number of task blocks that could satisfy different applications and configurations is variable, this method of catering for the provision of free task blocks is convenient and more efficient.

PROCEDURE SUSPLO 1 (SUSPROC): (876 - 906)
SUSPROC references the process to be included in the suspended state map, a BOOLEAN ARRAY here. The process allocator checks SUSPLO Lockout and if engaged joins a special queue and passivates. The process allocator engaging SUSPLO, having finished accessing the suspended state map, will release SUSPLO and activate the first waiting
process allocator, if any. When this process allocator is re-activated, it engages SUSPLO for 10.2 micro-seconds, the time required to insert a process in the suspended state map. The process is inserted in the map by setting the array element corresponding to the process index (PI) to true i.e.

\[
\text{SUSPMAP(SUSPROC.PI): = TRUE;}
\]

SUSPLO is then released, and the first process allocator waiting is activated, if any.

**PROCEDURE SUSPLO 2: ( 909 - 960 )**

The actions of this procedure follows closely those of SUSPLO1(SUSPROC), however, the intention here is to locate the highest priority suspended process in, and remove it from the suspended state map. As processes' priorities are inversely proportional to their process indices, i.e the lower the process index, the higher the priority, the search for the highest priority process consists of scanning the suspended state map (the BOOLEAN ARRAY SUSPMAP (0:127)) from the beginning until a true-value element is found. The selected process is referenced and removed from the map by re-setting the subscript value to false.

CURP references this process model or entity.

Since when initialising the simulator, a number of background processes equal to the number of CPUs are created and set running on those CPUs, there will always be at least one suspended process in the suspended state map. If no such process is found, that will be an indication of malfunction and an error message will be outputted.
In the Mark IIIBL System, the time for which SUSPLO is engaged is partly dependent on the location of the selected process in the suspended state map and hence on the value of its process index. Thus the time for which SUSPLO is engaged is calculated on the basis of the selected process index, using the empirical formula

\[ T = 16.6 + 4(\text{PI} / 16) \text{ micro-seconds} \]

where

// denotes integer division
PI the process index value.

PROCEDURE LOAD TASK: ( 963 - 973 )

It loads the selected task block details into the CPU general registers Go - G7. This procedure is called when a suspended (unblocked) process is selected to run and when a FETCH, SEEK or BLOCK call is executed and the requested task is found in the process's input queue. It also copies the task details in PD (17 - 24).

PROCEDURE SYSCHED: ( 976 - 1019 )

It sets the scene for a system re-schedule when a higher priority process model (or entity) is suspended while a lower priority one is running. The suspended state map is scanned to identify the higher priority suspended process. If the suspended state map is found to be empty, an error message will be generated as that constitutes an illegal system state. This is because at least one, or more, background processes are expected to be suspended.
The established process index of the suspended process, if found, is then compared with the lowest priority process running i.e. LCPU.CURP. If the suspended process is of higher priority, then the system ought to be re-scheduled to allow the suspended process to run as soon as possible. This is achieved, in the real system, by triggering a special interrupt level in the interrupt triplicates unit; the suspended queue interrupt. This interrupt, being an immediate interrupt, will be steered to LCPU forcing it to reschedule and select the highest priority suspended process. In the simulator, the suspended queue interrupt is represented by a BOOLEAN variable, SUSPQINT. To set it TRUE, a lockout, INTLO unique to the interrupt triplicates unit has to be engaged first. If INTRIP, the interrupt triplicates entity (5.4.9) is not servicing a clock or suspended queue interrupt, it will be activated by the process allocator to service the triggered suspended queue interrupt.

PROCEDURE CPUSCHED ( 1022 - 1048 )
Calls SUSPLO2 procedure first to select the highest priority suspended process to run on this CPU. It then compares the priorities of the processes running on all the CPUs to derive the identity of LCPU, i.e. the new CPU running the lowest priority process. This updated reference of LCPU is then passed to INTRIP, the interrupt triplicates unit, after engaging its lockout, INTLO.
REAL PROCEDURE SECS: (1051 - 1068)

This procedure calculates the time taken to insert a task in a process input queue. That time will depend on a number of parameters. They include whether the task inserted is the last in the queue, the queue was empty to start with, the process is in a blocked state etc. The time, also includes that required to engage the PROLO lockout.

PROCEDURE HANDTASKANDSETSTATE: (1071 - 1134)

It inserts the task to be handed into the process's input queue ranked by its priority. The task priority is obtained from the task index table of the calling process i.e the process handing the task.

The procedure then checks to see whether the called process is a periodic one (i.e the process handed the task). This will be indicated by the flag PERIODIC in the called process being set TRUE. If this is the case, the procedure will check whether the task just handed is an INTIM, periodic unblocking task. This is indicated by the task having a priority value of 5. The called process will also be checked for a blocked state. If both conditions are true, the process state will be changed to suspended (unblocked) and the process inserted in the suspended state map.

If the called process is not a periodic one but is blocked, the procedure will check whether the priority of the task just handed is greater or equal to the priority of the requested task (indicated by the level of the block).
If that is the case, the process state will be changed to suspended (unblocked) and inserted in the suspended state map as before.

At the end of a service routine, the process allocator always selects the next process to run as follows:

The process allocator checks the queue LPAQ. If INTRIP is not waiting then the selected entity is scheduled to be activated after the process allocator entity. Otherwise if LPAQ is not empty, INTRIP's reference CUCP is made to refer to the selected or calling process and INTRIP (which is the entity waiting in LPAQ) is scheduled to be active after the process allocator. In either case, the process allocator passivates awaiting a service call.

When activated by a service call it goes out of the queue and repeats the service cycle.

The above-mentioned procedures matches closely the equivalent routines of the process allocator in microprogram and represents its most important attributes.

The process allocator model is always in one of two states: either servicing a call or an interrupt or waiting passively for a service call. (Figure 4.6.1) is a gross state transition diagram (STD) of the process allocator life cycle.

By having the process allocator model waiting passively or 'going to sleep' until awakened by a service request, considerable saving is made on the model runtime as compared to the other alternative of having it checking continuously for the arrival of a service request. The former case is coded in SIMULA as:

181
WHILE NOT (CALL FOR SERVICE) DO PASSIVATE
ACTION -

Whereas the latter case is coded as:

WHILE TRUE DO
ACTION -

The penalty in the former case which is more efficient
is that the activation of the process allocator must be
made explicitly by the call requesting entity. That
entity or process model will transmit the requested call
service index to the process allocator activates its process
allocator and then passivates.

There are five types of calls for service a process can
issue to its process allocator. These are:

a) HAND - to hand a task to another process.
b) FETCH(N) - a request to fetch a task of
priority N or less from the process input
queue.
c) SEEK(N) - as for FETCH(N) except that the
task priority should be exactly equal to N.
d) BLOCK(N) - as for FETCH(N) except that the
process wishes to stop running and be blocked
if the requested task is not found.
e) SELF(N) - as for HAND except that the task
is inserted in the process's own input queue.

The calls to TRAP and UNTRAP a process are irrelevant to
the objective of this simulation and hence not considered.

When the process allocator model is activated either by
its running process or the interrupt triplicates it checks
whether the requested service is for an interrupt or a process
allocator call. This will be indicated by the BOOLEAN
variable INTERRUPT, which is local to the process allocator,
being TRUE or FALSE respectively. This variable is always set by the interrupt triplicates before 'awakening' the process allocator to handle the interrupt.

If it is an interrupt, the process allocator model will nest (store away) the general registers of its CPU (G0 - G7) into the process descriptor of the running process (PD(24:31)). The process descriptor is a per-process array (PD(0:31)) to store the environment of an interrupted process and any other relevant information. The CPU condition codes will also be stored in PD(23). The running process state is then changed to suspended (interrupted) (PD(g) = 2) and the process included in the suspended state map, SUSPMAP. The process allocator then check INTOLO lockout, and when free engages it for a period of 46.8 micro-seconds. The BOOLEAN variable CLOCK is checked next for a clock interrupt. It is set by the entity CLOCKINTERRUPT (1756) every 10 msec.

If it is a clock interrupt, then it is time to check the list of periodic processes and see if any are due to be activated. The process allocator fetches a free task block from the free tasks list after engaging FREELO lockout. This task block is used to store the process indices of the periodic processes to be activated at that clock interrupt. The process indices are read from the periodic processes table, PPTABLE; a two-dimensional Array indexed by a pointer which is incremented at every clock interrupt. The minimum length of the array PPTABLE is determined by the periodicities of different processes as the least
common multiple of the periodicities e.g. if process X has periodicity of 2 (activated every other clock interrupt), process Y periodicity 3 and process Z periodicity 4, then the minimum length of PPTABLE is 12. When POINTER reaches the value of 12 it is reset to 1. The Array PPTABLE is part of the interrupt triplicates model data structure.

The process allocator will hand this task to INTIM which will be blocked waiting for it and hence will be suspended (unblocked). The process allocator then schedules its CPU (CPUSCHED) which sets the reference variable UNSUSPENDED to refer to the selected suspended process. This is then followed by a system schedule (SYSCHED) to ensure that INTIM will soon be run. The process allocator checks whether the process is suspended (Interrupted) or suspended (unblocked) by examining the value of PD(8). A value of 2 indicates the former state and that of 3 the latter. This information is required to restore the values of the general registers.

The selected process state is then set to running (PD(8) = 1) and reference by CALLINGPROCESS. This reference is used in the communication between the selected process and its process allocator (1227).

If the selected process was suspended (interrupted) then the general registers and condition codes value are de-nested (copied back) from the process descriptor of the selected process (1230 - 1233). The de-nesting of the condition codes is to cater for the case of a process issuing a SEEK, FETCH or BLOCK call to its process allocator. The process allocator finds the requested task, but before exiting back to a process an interrupt signal arrives and the process
pre-empted, i.e. suspended (interrupted). To enable it to check on the result of its last call when allowed to run again, the value of the condition codes need to be nested and de-nested later. A positive value of the condition codes indicates that the requested task is found.

The process allocator next checks to see if any interrupts are pending. This will be indicated by the interrupt triplicates waiting passively in LPAQ. If no interrupt is pending, it then checks whether the selected process was last interrupted while processing i.e. in the middle of a 'HOLD' statement, which represents the passage of time taken by the activity on which the process is engaged. If this is the case, then the remaining activity time would have been calculated by INTRIP, the interrupt triplicates and stored in the real variable REMAININGACTIME. So if this variable is greater than zero, it will schedule the selected process at a simulated time equal to the current simulated time plus REMAININGACTIME, otherwise it is scheduled after the process allocator.

On the other hand if INTRIP is waiting for the process allocator in LPAQ to finish its current service call, then INTRIP is scheduled after the process allocator passivates and a reference of the selected process is passed to INTRIP.

If the last state of the selected process is suspended (unblocked) i.e. PD(8) = 3, then the unblocking task will be linked out of the input queue and its contents loaded into the general registers before either scheduling the selected process or INTRIP in the manner described above.
When activated by a service call, the process allocator always checks for the arrival of an interrupt first by checking the flag INTERRUPT. If not an interrupt then it is a process allocator call. The process allocator uses a call index transmitted by the calling process (CIX) in conjunction with a SWITCH statement to branch to the appropriate call servicing routine. (1144).

For a FETCH(N) call (1350 - 1440), where N is the minimum priority of the task to be fetched, the process allocator first engages the calling process's PROLO lockout. This may imply a delay. All the lockouts are simulated as resources for which all process allocators compete. Having engaged PROLO, it scans the input queue for the task requested. If the task is found, it will be unlinked from the input queue and PROLO released. The task contents (words 3 - 10) are copied to the CPU general registers (Procedure LOADTASK). Lockout FREEL0 is engaged and the task block returned to FREETASKLIST, so that it can be used again. The condition codes are set to indicate a successful fetch. On the other hand, if no appropriate task is found, then PROLO is released and the condition codes are set to zero. In either case, the process allocator entity will reach by then the end of FETCH(N) call servicing routine. What remains is to decide on the next entity to schedule. If the interrupt triplicates unit is waiting to interrupt this process allocator then the triplicate unit (INTRIP) is activated else the calling process is activated. The process allocator passivates awaiting the next cycle of service.
For a SEEK(N) call, the course of action taken by the process allocator is identical to that for a FETCH(N) with the exception that the task is to have exactly a priority equal N (1443 - 1495).

A BLOCK(N) call (1498) is identical to FETCH(N) in case the requested task is found. If the task is not found, then the process state is changed to blocked in its process descriptor (i.e. PD(8) = 4) and the priority of the requested task (N) is stored in PD(9). The condition codes are set to zero and PROLO released. The process allocator model then executes a CPUSCHED. Depending on whether the selected process is suspended (interrupted or unblocked) the working environment of the selected process is restored as described above starting at the label RUNSSELECPROC (1215).

If the call index translates to a HAND call (1606 - 1672) the process allocator determines the value of the outgoing task index (OGTI) from the register G(0). This OGTI is supplied by the calling process wishing to hand a task, to enable the process allocator to obtain information such as the identity of the called process, incoming task index for the called process and the task priority. Such information is obtained by using the OGTI as an index down the calling process TIT (301). An error message will be outputted if the array subscript storing the called process index is found to be zero.

The CPU general registers are defined by the process model ARRAY G(0:7), an attribute of entity AP (301) i.e. instead of writing the task contents into the CPU registers (G(0:7) local to CPU entity), they are copied to the process model G(0:7).
The same is done when requesting a task. This has the advantage that inside its body, the process model can access and interrogate \( G(0) - G(7) \) without the need for the remote accessing of the CPUs \( G(0) - G(7) \) using the dot (.) or INSPECT constructs of SIMULA (e.g. \texttt{RELEVANTCPU. G(0) vs G(0)}) . This strategy gives the software engineer the illusion that he is dealing with a real process as he can refer to the CPU registers directly, the thing he or she is accustomed to in practice. In turn this will make it easier for the software engineer to design and code models of real processes.

The process allocator on servicing a \texttt{HAND} call, engages \texttt{FREELO} lockout to fetch a free task block. It loads the information into the free task block both from the process's \texttt{TIT} and its registers. The reference variable \texttt{TASKHANDED} references this task block before calling the procedure \texttt{HANDTASKANDSETSTATE}. If the called process is suspended by the procedure, then \texttt{SYSCHED} is called to re-schedule the whole system in case the process just suspended is of a higher priority than a running process. The reference \texttt{TASKHANDED} is reset to none and the activation of the next process to run is identical to that of the \texttt{FETCH(N)} call.

A \texttt{SELF(N)} call (1675 - 1712) is issued by a process wishing the task to be inserted in its own input queue, with priority \( N \). The process allocator entity fetches a free task block - after engaging \texttt{FREELO} lockout. The \texttt{OGTI} is copied straight as the \texttt{ICTI} (Incoming Task Index). The Incoming Task Index is used by a process to determine the course of action in respond to the task received.
The reference CALLEDPROCESS is made to refer to the calling process entity. Thus there is no need to access the entity's TIT. The contents of G(0) - G(7) are copied to the task block and HANDTASKANDSETSTATE called in followed by SELECTPROC as before.

The initialisation section of the simulator creates the process models (entities), hardware models, resources, queues etc. in a configuration. It then initialises the TITs of the process models and starts the simulation running by scheduling CLINT. The simulator progresses in simulated time where the simulator components act and interact reproducing the system dynamic behaviour. At the end of the simulated period a summary report is outputted.

5.4.11 Activity Durations

In the majority of cases, the duration of an activity is of a fixed length of time. In such a case the duration of an activity is a function of the number of program steps that constitute the activity and the speed of the computer. This activity is modelled by modelling the events that constitute the cause and effect of the activity plus a statement that models the passage of the activity time. As mentioned fore, this corresponds in SIMULA to the statement HOLD(T), where T is the activity time. This procedure was followed in calculating the durations of such fixed time activities.

Another type of activity will depend on some other additional parameters. For example, in searching the suspended state map for a suitable suspended process, the time of a search will depend also on the location of the process index in the suspended state map.
In this case the time $T$ of this activity may be expressed as

$$T = 16.6 + 4 \times \text{PI} / 16 \ \mu\text{sec.}$$

where

- $\text{PI}$ is the process index
- $/ /$ is the integer division.

The constant 16.6 represents the fixed time overhead. The denominator 16 stems from the fact that Mark II BL is a 16-bit machine, and that the position of the bit representing the process (Bit = 1 if process in map, = 0 if not) reflects the process index and hence its priority. The number 4 is the time to scan a whole word (16 bits).

Another example is the time required to insert a task in a process's input queue. This will depend on factors such as the length of the queue, whether the task is inserted as the last task, the process is in a blocked state and the task is unblocking. Denoting these parameters by $X$, $L$, $B$ and $Y$ respectively, then

$$T = 28.6 + 10.8L + B + 7.4X + 16.2Y \ \mu\text{sec}$$

This formula is used in conjunction with a HAND call. In a SELF(N) call, the expression reduces to

$$T = 16.8 + 10.8L + 7.4X \ \mu\text{sec}$$

In a SEEK(N) call, the seek time depends on the location of the required task in the input queue. Thus the time taken in a successful seek is

$$T = 33.2 + 10.6K \ \mu\text{sec}$$

and for an unsuccessful seek

$$T = 8.6 + 10.6K$$

where $K$ is the ranking of the task in the input queue. For an unsuccessful seek, $K$ is the queue length.
The duration of servicing a HAND call depends, also, on whether the process handed the task is suspended as a result and whether a suspended queue interrupt is triggered consequently. Denoting these two factors by say, \( Y \) and \( Z \), where \( Y \) and \( Z \) are INTEGER quantities that take the values 0 or 1, this activity duration can be expressed as

\[
T = Y(18.6 + 4.0 \text{ PI} / 16 + 4.8Z) \, \mu\text{sec.}
\]

Within the model of the process allocator, the activity of servicing a call or an interrupt may be considered to be made up of a number of smaller activities. When developing the model, the HOLD statements representing the time of each individual 'mini-activity' is located immediately before, within or after the 'mini-activity' model as appropriate. Hence the overall duration of the activity is split up into a number of smaller durations scattered throughout the activity. Although that may entail a smaller increase in the model run time, it is found that by so doing, the model behaviour is more consistent with that of the real system. The detailed dynamic model behaviour is revealed by a trace incorporated in the simulator (see Chapter 6).

5.5. CONCLUSIONS

The description of the simulator program presented in this Chapter represents the basic model of Mark II BL system, i.e. the models of the relevant hardware and operating system processes. In addition to these, models of application processes and application hardware have to be added to make up the overall model of a particular exchange configuration. The above basic model of Mark II BL can be grouped in a SIMULA class and named say MARK II BL e.g.
CLASS MARK II BL;
BEGIN
ENTITY CLASS CPU; ---;
ENTITY CLASS TASKBLOCK; ---;
ENTITY CLASS INTRIPICATES: ---;
ENTITY CLASS PROCESS ALLOCATOR: ---;
END ** MARK II BL;

To run a particular exchange model, it is required to prefix the block containing the description of the application models by CLASS MARK II BL. Hence, the application model will run on top of the Mark II BL model.

Moreover, if the SIMULA compiler supports EXTERNAL COMPILATION, the package (CLASS MARK II BL) is only needed to be compiled once and used as an object-code module to prefix a particular exchange model. This will result in considerable saving in computer time if the model is to be run frequently. Chapters 6, and 7 discuss models of specific applications.
CHAPTER 6

THE VERIFICATION AND VALIDATION OF MARK II BL SYSTEM SIMULATOR

6.1 THE VERIFICATION PROBLEM

One of the best definitions of simulation is by Sayre and Crosson in a discussion of a model of the human mind (MIHR 72):

"A simulation model is a symbolic (as opposed to physical or material) representation of a phenomena or system, yet in contradistinction to mathematical models, the symbols of a simulation model are not all manipulated by a well-formed discipline, such as algebra, the integral calculus, numerical analysis or mathematical logic. Indeed, it is becoming more apparent that such simular models are of a more general nature than those restricted to mathematical operations for their solution and/or evaluation."

The development of such a simulation model of a real or proposed system is a purposeful orderly activity, which is in essence an extension of the scientific method of enquiry (MIHR 71). Purposeful, in the sense that modelling goals are first well defined and understood. It is orderly and planned because a well defined, partially-iterative procedure is available for model development and hence the realisation of the stated goals.

But the credibility of any modelling effort rests on the firm demonstration that the simulator represents reality.

Unfortunately, the problem of verifying and validating computer simulation models has received little attention in the literature. Simulation analysts usually have very little to say about the way one goes about using a simulation model or the data generated by
such a model on a digital computer. Most of those who made
the attempt restricted themselves to purely graphical (as
opposed to statistical) techniques to prove the "goodness of fit"
of their simulation model (NAYL 67). An explanation of this
phenomena, due to NAYLOR et al., is that "In part, the reason for
avoiding the subject of verification stems from the fact that
the problem of verifying or validating computer models remains
today perhaps the most elusive of all the unresolved methodological
problems associated with computer simulation techniques." As
a solution to the problem, they suggest a multi-stage verification.
The first stage of the methodology calls for the formulation
of a set of postulates or hypotheses that describe the behaviour
of the system of interest. This set of postulates is normally
derived from the analysts' already-acquired knowledge of the system
under study or of similar systems which have already been success-
fully simulated.

The second stage of this multi-stage verification calls for the
verification of the postulates adopted, subject to the limitations
of existing statistical tests.

The third stage consists of testing the models' ability to predict
the behaviour of the system under study. It envisages
two approaches here to accomplish that goal; namely, historical
verification and verification by forecasting. The essence of these
approaches is prediction; for historical verification is
concerned with retrospective predictions, while forecasting is
concerned with prospective predictions. Historical verification
in this sense involves the choice of one historical path along
which the system was or could be driven and subsequently comparing
the output data from the system with that outputted by the model
when it is driven along the same path, i.e. under the same environmental
conditions. This is actually the approach adopted in the validation of the Mark II BL simulation model, as will be explained later. Though in this case the output data generated by the system cannot be used as a check on whether the model did actually point out the best policy to follow, the actual outcome of an alternative policy, strategy or design selected can be compared with the outcome predicted by the simulation model on which basis the respective policy, strategy or design is chosen. This is the essence of verification by forecasting. On the other hand, Van Horn (HORN 72) depicts two stages to accomplish verification and validation. Firstly, we must understand the behaviour of the simulator itself in terms of relations that exist between inputs and results.

Secondly, and this is often the more difficult task, we have to translate 'learning' from the simulation to 'learning' about the actual system. These two stages are defined respectively as verification and validation. Alternatively, Fishman and Kiviat (FISH 68) divide simulation testing into three categories:

1) Verification: ensures that a simulation model behaves as an experimenter intends.

2) Validation: tests the agreement between the behaviour of the simulation model and a real system.

3) Problem Analysis: embraces statistical problems relating to the analysis of data generated by computer simulation.

A simulation project normally begins by stating clearly the objectives of the exercise in the form of the questions to be answered together with performance measures appropriate for answering them. Following this initial stage, model development is governed by five additional stages given by Mihram (MIHR 72) as follows (Fig. (6.1)):
FIGURE (6.1) : THE MODELLING PROCESS

Diagram showing the modelling process with stages including:
- Modelling Goals
- System Analysis
- System Synthesis
- Model Verification
- Model Validation
- Inference
1) **System Analysis**

The study of a system in order to ascertain its salient elements and to delineate their interactions, relationships and dynamic behaviour mechanisms. Here, the model entities are identified and their attributes specified. The state variables and transformational rules are enumerated.

2) **System Synthesis**

The construction of a complete logical structure of the system elements and their interactions to provide a symbolic model of the system. Since this stage realises the model and the computer language influences the realisation, the latter ought to be selected at this stage. Appropriate support data are also determined and collected.

3) **Verification**

Comparison of the model responses with those which would have been anticipated if the model algorithms structure were prepared as intended. Here, the model is debugged. Tracing routines are very helpful in the verification of the model logical structure by enabling the simulation analyst to analyse closely the dynamic sequence of events as the model components interact in simulated time. If the model fails to compare favourably with predicted theoretical values or known system performance, then a return to the previous stage is necessary (Fig 6.1). For stochastic models, one-sample statistical tests are appropriate.
4) **Validation**

The comparison of responses of the verified model with available information regarding the corresponding behaviour of the simulated system. Failure of the model to compare favourably with the real system necessitates a return to Stage 2 (model synthesis). An improper comparison may force the analyst to re-think the abstraction selected in Stage 1 (system analysis).

5) **Inference**

The contrasting of model responses under alternative input conditions. Experiments using the verified and validated model are designed and conducted. Comparisons are made and recommendations and conclusions drawn to satisfy the goals set at the outset of the simulation study.

Thus, the end result of a simulation model construction is the creation of a credible system representation from which inferences regarding the actual system's performance and behaviour can be made without resorting to costly and time-consuming experimentation with the actual system.

Most simulation analysis goals fall into two categories: to ascertain the static effects of the model's performance (which will be indicated by the value of one or more of its state variables or descriptors at the end of a specified period), or to determine the dynamic performance by observing the behaviour of the state variables during a simulated period of T units, say. The static effect of a model's performance may be represented by the multivariate function S(T), denoted as the simulator response at T. S(T) may be expressed as (MIHR 72):

\[ S(T) = R_T(X_1,X_2,\ldots,X_n) + E_T(X_1,X_2,\ldots,X_n ; S) \quad \ldots \ldots (6.1) \]
where

\[ R_T \] is the simulator response function observed at time \( T \).

\( X_1, X_2, \ldots, X_n \) are environmental conditions (input parameters).

\( E_T \) is a random response or 'error' of mean zero introduced by randomness in the model.

\( S \) is the random number seed or seeds employed explicitly or implicitly in stochastic models.

For deterministic models \( E_T(X_1, X_2, \ldots, X_n; S) = 0 \). The dynamic characteristics are measured by a finite set of values of the form

\[ S(t), t = 1, 2, \ldots, T \]

Where

\( S(t) \) is as described above.

A systematic verification procedure involves the determination of a specific set of environmental conditions \( X_1', X_2', \ldots, X_n' \) for which the model response could be predicted provided that the model is programmed in accordance with the analyst's intentions. The determination of such input/output relationships.

\[ S'(T) = R_{T}(X_1', X_2', \ldots, X_n') + E_{T}(X_1', X_2', \ldots, X_n'; S) \]

is easier to establish once the stochasticity in the model is eliminated. This is also true for the model validation as will be shown later.

6.1.1 The Verification Experiment

The only feasible way to verify the operating system model is to model the software of a typical computer-controlled telephone exchange and run the resulting model in conjunction with the operation system model. If the model is run in a controlled deterministic self-driven mode, then the model static response at time \( T \) reduces to

\[ S'(T) = R_{T}(X_1', X_2', \ldots, X_n') \quad \ldots \ldots (6.2) \]
This can be compared with the anticipated output for verification purposes. Also, the dynamic response \( S'(t), t = 1, 2, \ldots, T \) must be analysed and studied carefully to ascertain the credibility of the model's dynamic behaviour. For this purpose, an appropriate trace program must be imbedded in the model to disclose the model components interactions and the state descriptors values as simulated time progresses.

Unfortunately, a typical system X exchange software was not available at the time of model verification. So a model of a hypothetical suite of application programs of a particular system X exchange was developed by the author, based on the anticipated exchange architecture (LAWR 77). The application software modelled belonged to the Digital Main Network Switching Centre (DMNSC).

6.1.1.1 The Hypothetical DMNSC

The basic telephony requirements of application programs may be summarised as follows:

- Detect seizure
- Determine class of service
- Receive and store digits
- Translate code digits into routing and call charge information
- Set-up own exchange trunking
- Forward digits to set-up the rest of the network
- Detect called subscriber answer
- Detect clear forward or clear backward
- Release network
- Release trunking.

In addition to the above telephony functions there remains the functions of routing, diagnostics, traffic recording and maintenance control.
The application software is partitioned into a number of software modules with clearly defined interfaces. Each module, known as a process, takes care of one or more of the above-mentioned telephony functions. As long as the interfaces between the processes are strictly observed, they may be developed and tested separately by teams of software engineers, possibly in different physical locations. As an initial stage of partitioning, it may be observed that the above telephony functions could be grouped into three categories each served by an application process (Fig: (6.2)). These processes are:

a) Line Circuit Handler:
This process interfaces with the incoming and outgoing telephone lines and deals with all aspects of signalling.

b) CALL Control:
Responsible for processing telephone calls, establishing the network routing, supervising the call and upkeeping and updating of telephone traffic statistics.

c) Switch Handler:
This process interfaces with the exchange switch matrix and handles all call set-up and clear-down functions.

With only three processes, there is a limit to the maximum traffic carrying capacity set when all the three processes are running simultaneously in a multi-processor system. An obvious way to overcome this disadvantage and increase the traffic-carrying capacity is to split up the individual processes on a functional basis. For example, the line circuit handler process may be split into a number of processes; a process for each signalling system e.g. a loop-disconnect handler, DC2 handler etc. The disadvantage of this method is that an appreciable increase in telephone-traffic handling capacity is only possible if the circuits for each type of signalling are roughly equal. A better way of splitting is to provide a line circuit handler per group of lines, and to have separate handlers for incoming and outgoing groups of lines.
FIGURE (6.2): BASIC STRUCTURE OF HYPOTHETICAL DMNSC
The call control may be split into three smaller processes, e.g., call supervision, network routing, and traffic recording. Call supervision may be split further into incoming call supervision and outgoing call supervision. The processes may then be replicated to deal with different groups of circuits. For a switch handler, the time to set-up a path is so short in a digital exchange (of the order of a few hundreds of microseconds) that a single process is capable of carrying a few thousands of erlangs. However, for other types of exchanges it depends on the form of trunking, e.g., for a cross-bar exchange, a switch handler process may be allotted to each router control with the identity of the incoming circuit determining which router control and hence which switch handler is to be invoked. The resulting software structure is shown in Fig. (6.3).

Splitting the processes is not without its own problems. The increase in the number of processes imposes a heavier burden on the operating system to handle the inter-process communication by passing tasks of information or messages between them. This may well become the limiting factor as far as the total system traffic capacity is concerned. This problem may be eased to some extent by having some of the processes periodically activated rather than activated with the arrival of each individual task.

The DMNSC exchange handles only trunks and junctions. Generally, for a certain type of exchange there is no optimum design for its software structure and there is always the trade-off between the conflicting requirements of traffic carrying capacity, economy, efficiency, etc. The function of the register-selection process in Fig. (6.3) is to select a register if an MF sender or receiver is required and set-up the path between the line and register via the switch handler.
ENHANCED DMNSC STRUCTURE AND PERIODIC PROCESS TABLE
The processes are table-driven. The set of processes a process is allowed to communicate with is strictly defined in the process's task index table (TIT). The destination of a message and the response of the destination process are clearly defined for a process by the outgoing task index (OGTI) value a process passes to the real-time operating system when requesting a task to be handed. The operating system then uses this outgoing task index value to index down the process's task index table to obtain information such as the identity of the called process, the task priority and an incoming task index for the called process. The incoming task index (together with the task contents) is used by the called process to select the appropriate action and pass further messages to other processes and so on. Thus, if the different values for the incoming and outgoing task indices are defined for each process, the task index tables can then be compiled (with suitable choice of task priorities) to determine the sequence of operating the exchange.

If this sequence can be deterministically identified beforehand, then it may be used to compare with the dynamic interactions between the software processes and the operating system and hence constitute a powerful verification tool. For the sake of obtaining a detailed record of this dynamic interaction, a special trace program is written and incorporated in the model. The characteristics of this trace routine follows.

6.1.1.2 The Trace Program

The objective of the trace is to output dynamically values of the state variables and the transformational rules applied to change them. The contents of the operating system global tables, such
as the map of suspended processes ready to be selected to run 
(Suspended State Map) and the free task list are continuously 
outputted whenever they are accessed by one of the process 
allocators, together with an indication of the value of simulated 
time. The suspended state map trace gives the names and process 
indices of the processes in the suspended state map. Other state 
variables such as the suspended queue interrupt (SUSQINT), the clock 
interrupt (CLOCKINT) and the reference to the CPU running the 
lowest priority process (LCPU) are also outputted, whenever their 
values or references change. Moreover, when a suspended queue 
interrupt is triggered an indication is given as to the reason 
for that, in the form of the identity of the highest priority 
suspended process, the priority of the process running on LCPU 
(i.e. LCPU CURP) and the identity of LCPU.

Whenever a running process issues a call to the process allocator 
of the CPU in which it is running, the identities of the call, 
the process and the CPU are indicated. For a HAND call, the 
call index, the outgoing call index and the called process identity 
are also outputted. This is quite a useful piece of information 
to use in checking against the task index tables of the calling 
and called processes to see whether the process allocator extracts 
the right sort of information regarding the called process, the 
task priority and the incoming task index for the called process. 
It can also be used to check on the reaction of the called process 
to the task handed which is supposed to be governed by the value 
of the incoming task index in most cases. In other cases, the 
response of a process depends on the contents of one or more of 
the general registers (scratch pad) e.g. in the validation experiment, 
the storage allocator response to requests from RASH replicates
depends on the value of G2 which is indicative of the nature of the request (see Section 6.2.). For this reason, the contents of a task (Go - G7) are always outputted whenever the task is loaded in the simulated CPU general registers. The number of tasks in the input queue of the called process and the task priorities are also indicated and whether the process being handed the task changes state as a result of that and hence its insertion in the suspended state map.

For a FETCH or SEEK call, the identities of the calling process and its CPU are given and whether the call is a successful one or not. For a BLOCK call, in addition to that, the trace indicates for an unsuccessful BLOCK call, the identity of the newly selected process and its last state, it being suspended (Interrupted) or suspended (unblocked).

In addition, the trace program outputs error messages whenever a malfunction of the operating system occurs. Some error numbers lead to the abortion of the simulation run. A sample of the trace program output is shown in Appendix (C).

Since the process allocator is at the heart of the real-time operating system and handles the scheduling of processes, the interrupts servicing and the inter-process communication, the trace program is embedded in the process allocator. The trace can be switched on and off to run for certain durations of simulated time.

The detailed trace output makes it feasible to scrutinize the behaviour of the operating system model in a complex multi-processor environment and verify the different algorithms within the operating system.
6.1.1.3 The Verification

A great care was taken in the splitting of the software structure of Fig. (6.3) into periodic and aperiodic processes, in the choice of process indices and in the design of individual processes task index tables. This was to ensure an even distribution of load among the processes and the utilization of all types of services offered by the real-time operating system. The following software processes were selected to be periodic. The name of the process is followed by the process index number followed by the process periodicity i.e. how often a process is activated.

Periodic Processes:

<table>
<thead>
<tr>
<th>Process Name</th>
<th>Index</th>
<th>Periodicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line Circuit Handler 1 (LCH1)</td>
<td>11</td>
<td>11 msec</td>
</tr>
<tr>
<td>Line Circuit Handler 2 (LCH2)</td>
<td>12</td>
<td>22 &quot;</td>
</tr>
<tr>
<td>Switch Handler (SH)</td>
<td>13</td>
<td>22 &quot;</td>
</tr>
<tr>
<td>Incoming Call Supervision (ICS)</td>
<td>14</td>
<td>33 &quot;</td>
</tr>
<tr>
<td>Outgoing Call Supervision (OCS)</td>
<td>15</td>
<td>33 &quot;</td>
</tr>
<tr>
<td>Incoming Register Selection Process (IRSP)</td>
<td>16</td>
<td>44 &quot;</td>
</tr>
<tr>
<td>Incoming Line Circuit Routiner (ILCR)</td>
<td>19</td>
<td>66 &quot;</td>
</tr>
</tbody>
</table>

The minimum length of the periodic processes table (part of the interrupt triplicates unit data) is the least common multiple of the periodic processes periodicities, in our case that of 1, 2, 3, 4 and 6 i.e. 12. The periodic processes table is shown in Fig. (6.3). Its pointer is incremented with the arrival of each clock interrupt or reset to 1 whenever its value reaches 12.

As described in Chapter 5, INTIM process (Interrupt and Timing Process) accesses this table at the arrival of a clock interrupt, using the most recent pointer value to obtain the identity of the periodic processes to be activated at that time by handing them periodic unblocking tasks of high priority.
A process is to be activated at that particular clock interrupt if its corresponding entry in the row indicated by the pointer value is non-zero. On the other hand, the aperiodic (non-periodic) processes and their process indices were selected as follows:

<table>
<thead>
<tr>
<th>Process Description</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outgoing Register Selection Process (ORSP)</td>
<td>17</td>
</tr>
<tr>
<td>Network Routing (NR)</td>
<td>18</td>
</tr>
<tr>
<td>Outgoing Line Circuit Routing (OLCR)</td>
<td>20</td>
</tr>
<tr>
<td>Traffic Recording</td>
<td>21</td>
</tr>
</tbody>
</table>

All processes are table-driven. In most cases the response of an application process depends on the value of the CPU condition codes after a FETCH, SEEK or BLOCK call to the process allocator e.g. LCH1, OCS etc. When the condition codes are positive, the response of a process may be dictated by the value of the incoming task index stored in GO e.g. IRSP, ILCR, SH etc., or by the contents of one of the other general registers e.g. IRSP, LCH2.

The processes and their task index tables are shown in Fig. (6.7) - Fig. (6.17). At the end of its processing cycle, a process normally picks up any remaining tasks in the input queue using BLOCK(15) call; if any, otherwise it is blocked until activated by INTIM or the arrival of a task, depending on whether it is periodic or aperiodic. This last bit of process logic is coded in a separate procedure - LASTJOB.

By following closely the detailed trace output of the system dynamics as it progresses through simulated time, and taking into account the structure of the processes and their task index tables, the logic of the simulator is verified against the expected behaviour of the real system and the intentions of the simulation analyst (Author).
In many occasions, the coding of the simulator algorithms had to be changed, especially the interrupt handling and the interaction between the process allocators and the interrupt triplicates unit. For example, if a running process was interrupted and pre-empted during a HOLD statement, then the remaining activity time represented by the HOLD statement had to be calculated and stored. In another instance, it was noticed that the local sequence control (LCS) of a process would change its position if the process was interrupted while being selected to run. When it was selected again to run, its LCS skipped one or more statements of its coding. This unexplicable behaviour of the SIMULA system seems to have been caused by a combination of scheduling statements being executed in quasi-parallel. When the scheduling statements at the end of each branch of the process allocator were re-arranged, this phenomenon luckily disappeared. A third and rather tricky problem was the division of an activity time (i.e. time consumed by a particular activity which is represented by a particular code) into smaller time slices and the insertion of those in HOLD statements at the right positions within the SIMULA coding of that activity in order to obtain the correct dynamic interactions. This 'tuning' process was a delicate and time-consuming one that required a great deal of patience. This iterative process was carried out for different activities until the proper dynamic interactions were obtained.

After a large number of runs, the verification was completed.
6.2 THE VALIDATION PROBLEM

In spite of the relative importance of simulation model validation, very few papers in the literature are dedicated to the subject (HORN 72, UNGE 75). In line with Fishman and Kiviat (FISH 68), Van Horn defines validation as "the process of building an acceptable level of confidence that an inference about a simulated process is a correct or valid inference for the actual process". This definition implies that validation is not intended to be used as a proof that the simulator is a true model of the real system. As a simulator may be considered as a finite-state machine that transforms inputs into outputs, it cannot be proved that two machines are identical just by comparing input-output transformations however large a sample is used (HORN 72). Fortunately, simulation analysts are concerned with validating the insights produced by the simulator rather than proving the truth of the model in every respect. In other words, the validation objective is to validate a specific set of insights not necessarily the mechanism that generated the insights. In this respect, a large number of tools and techniques are at the disposal of the analyst. These range from various statistical tests to complementary studies and field tests (MIHR 72, FISH 67, HORN 72). In his excellent paper, Van Horn listed the following actions that could be taken to achieve the desirable confidence in a simulation model.

1. Find models with high face validity for comparison.
2. Supplement the model by knowledge available from existing or past research, experience or observation.
3. Conduct simple empirical tests of means, variances and distributions using available data.
4. Run 'Turing' type tests.
5. When adequate data is available, apply complex statistical tests. These include analysis of variance, regression, factor analysis, spectral analysis and autocorrelation, \( \chi^2 \) and non-parametric tests.


7. Run prototype and field tests.

8. Implement the results with little or no validation.

However, validation is characterised as problem-oriented and the real task of validation is finding the appropriate set of actions. For example, Unger (UNGE 75) proved the validity of his computer resource allocation simulation model by comparing the model output with that of a validated analytical model of a known computer configuration. Hence the first action listed above proved to be sufficient for the purpose.

As stated before, a digital computer simulator is a finite-state machine for transforming an input set into an output set. Since insight is gained from the observation and analysis of the transformation process, overall confidence in the insight depends clearly on confidence in the transformation process. One of the obvious ways to gain confidence is to compare the outputs of the simulator and the system using identical input parameters. In such situations, more often than not, simple comparison of means, ranges, variances, tabular and graphical comparisons of performance measures will suffice for the purpose (BELL 72, HORN 72).

Simulators may either be deterministic or stochastic. If stochasticity can be removed from the actual system, the validation is then a straight-forward process. One needs only to observe the output or performance of the actual system under a controlled set of operating conditions and compare these in a one-to-one basis with the responses emanating from the simulator under the same set of operating conditions, i.e. compare the simulator responses \( S(T) \)
of equation (6.1) with $Y(T)$, the response of the actual system.

With specific regard to the Mark II BL real-time operating system model, the stochasticity represented by the system workload generator (the arrival of telephone traffic) may be removed and the system/self-driven. This can be achieved by running the system with one or more instances of a specific operating system process representing the system workload generator by demanding service from the operating system in a continuous cyclical fashion, as will shortly be explained.

6.2.1 The Validation Experiment

The Mark II BL processes involved in the validation are shown in Fig. (6.4). The validation experiment consists of nine runs of the simulation model in which one, two and three RASH process instances represent the loading of the operating system. They execute a loop continuously for a pre-determined interval of time. Every time a RASH process instance traverses its loop, it increments an execution counter. The extent of the agreement between the contents of real RASH counters and their corresponding models counters is indicative of the degree of credibility of the simulator. Models of the operating system processes in Fig (6.4) have been described before (Chapter 5) except for the RASH process. An appropriate description follows.

6.2.1.1 RASH process

The Thrash and Crash suite of processes is collectively known as RASH and has been provided within Mark II BL system software to assist in the debugging and commissioning of the real-time operating system. It allow the programmer to write his own test modules to perform functional and performance tests and control their
KEY
PAP: Process Allocator Process
RP: RASH Process
SAP: Storage Allocator Process
IP: INTIM Process
CP: CPU Process
ITP: Interrupt Triplicates process
PAC: Process Allocator Call
CIP: Clock interrupt process
BP: Background process

FIGURE (6.4) SIMULATOR STRUCTURE FOR VALIDATION EXPERIMENT
running interactively using the man/machine interface provided by RASH.

A macro-level flow chart of RASH is shown in Fig. (6.5) and the corresponding model flow chart in Fig. (6.6). The striking similarity between the two figures is yet another demonstration of the main characteristic of this simulator package - namely, the ease with which a real process can be transformed into its equivalent model in a one-to-one translation process.

There can be up to seven RASH processes running concurrently. RASH0 is known as the command process and it interprets the commands of the man/machine language used by the programmer to control other individual RASH processes. A RASH process is also called a test process and a test module can be executed by up to six such test processes at the same time.

A programmer may pass data items to a test module before it starts running. A further facility is available to set a sequence of predefined test modules executing within specified processes. The aim is to allow multi-function tests to be initiated by a single command and to restart the tests after every system or process rollback. A complementary command is used to stop the execution of the processes.

There is a basic structure which is identical to all test processes. It is best to imagine each RASH process as containing a loop which it can be forced to enter by giving it a RUN command. The loop itself involves emptying the input queue and dealing with any tasks, followed by entry into the test module specified by the operator. When the module has been executed, a counter, call the Execution Counter is incremented by one to record the fact that the loop
FIGURE (6.5) : RASH FLOW CHART

Close file 579.2 micro-secs

HAND

----- (To S.A.

BLOCK(15)

Open file 459.2 micro-secs

HAND

----- (To S.A.

BLOCK(15)

House-keeping 109.2 micro-secs

FETCH(15)

Condition Codes

? = 0

Increment Counter

House-keeping 109.2 micro-secs

> 0

House-keeping 600.0 micro-secs

Increment Counter

INTIM periodic timing task
\[ G(0) = 0 \]
\[ G(2) = 5 \]
\[ G(7) = 0 \]

**Increment Counter**

**FIGURE (6.6)**

RASH PROCESS MODEL FLOW CHART
has been completed once more. The loop is then started again. Once the loop is entered it is executed repeatedly until the operator stops the test module by issuing it a FINISH command (Fig.(6.5)).

Another facility available to the programmer is the timing facility of RASH. The run-time of test programs can be measured by counting the number of times a process executes its loop containing the test module, in a specified period of time. The programmer uses the TIME command to specify which RASH process he wishes to time and over what time interval the loop should be counted. The value of the execution counter will, therefore, be the number of times the test process (RASH) executed the test module in its internal loop, during the specified interval of time. The timing so obtained will include the portion due to RASH overhead and other operating system overheads. An estimate of RASH overhead is easily obtained by repeating the measurements with the test process executing an empty test module.

6.2.1.2 The Experiment

The procedure adopted in the validation experiment is for RASH to run an empty test module and to issue requests to the storage allocator to open and close its files. It is also arranged that it receives a time task from INTIM every 100 msec and that the time interval during which the loop is executed is 7.2 secs (Fig. (6.5)). The system is run in this deterministic fashion to exclude the element of stochasticity or randomness that would otherwise be introduced by the input process (arrival of telephone calls). The simulator is initialised with the background processes running on the CPUs.
CLASS OUTGOING REGISTER
SELECTION PROCESS
PI=17

FIGURE (6.7)

CLASS NETWORK ROUTING
PI=18

FIGURE (6.8)
CLASS SWITCH HANDLER
PI=13

FIGURE (6.9)

<table>
<thead>
<tr>
<th>TI</th>
<th>TIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
</tr>
</tbody>
</table>

DE SCHEDULE THIS PROCESS

SCHEDULE AFTER 700.0 MICROSECONDS

SCHEDULE AFTER 500.0 MICROSECONDS

SCHEDULE AFTER 1100.0 MICROSECONDS

SCHEDULE AFTER 1200.0 MICROSECONDS

SCHEDULE AFTER 1300.0 MICROSECONDS
CLASS OUTGOING
CALL SUPERVISION
PI=15

FIGURE (6.10)

<table>
<thead>
<tr>
<th>TIT</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>4</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>7</td>
<td>8</td>
<td>13</td>
</tr>
</tbody>
</table>
OLCR

\[ TI = \, ? \]

\begin{align*}
&= 1 \\
&= 2 \\
&= 3 \\
&\quad \text{SCHEDULE AFTER } 1100.0 \text{ MICROSECONDS} \\
&\quad \text{SCHEDULE AFTER } 1300.0 \text{ MICROSECONDS} \\
&\quad \text{SCHEDULE AFTER } 1400.0 \text{ MICROSECONDS} \\
&\quad \text{SCHEDULE AFTER } 100.0 \text{ MICROSECONDS} \\
&\quad \text{BLOCK(15)} \\
&\quad \text{CC = ?} \\
&\quad > 0 \\
&\quad \text{DESCHEDULE THIS PROCESS} \\
&\quad \boxed{TIT} \\
&1 \quad 13 \quad 5 \\
&2 \quad 12 \quad 2 \\
&3 \quad 15 \quad 3 \\
&\text{TOTAL} \quad 6 \\
&\text{TOTAL} \quad 4 \\
&\text{TOTAL} \quad 7 \\
\end{align*}

CLASS OUTGOING LINE CIRCUIT ROUTINER
PI=20

FIGURE (6.11)
CLASS LINE
CIRCUIT HANDLER 2
PI=12

FIGURE (6.12)

<table>
<thead>
<tr>
<th>TI</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

LCH2

HAND

DESCHEDULE THIS PROCESS

SCHEDULE AFTER 200.0 MICROSECONDS

SCHEDULE AFTER 200.0 MICROSECONDS

BLOCK(15)

CI? = ?

TI? = ?

G(3)? = ?

SELF(7)

HAND

HAND

HAND

HAND

SCHEDULE AFTER 1000.0 MICROSECONDS

SCHEDULE AFTER 900.0 MICROSECONDS
CLASS INCOMING REGISTER
SELECTION PROCESS (IRSP)
PI=16
FIGURE (6.13)

CLASS TRAFFIC RECORDING
PI=21
FIGURE (6.17)
CLASS INCOMING LINE
CIRCUIT ROUTINER
(ILCR) Pi=19

FIGURE (6.14)
CLASS LINE CIRCUIT
HANDLER 1 (LCH1)
PI=11

FIGURE (6.16)

<table>
<thead>
<tr>
<th>OCTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>16</td>
</tr>
</tbody>
</table>

227
To enable RASH instances to run, INTIM (Interrupt and Timing Process) is arranged to hand them unblocking tasks when INTIM instance is first activated with the arrival of the first clock interrupt at simulated time zero. That is to say, RASH processes are considered to be periodic temporarily to enable them to start running. In reality they are set running by the software engineer typing the command RUN. This diversion from reality is insignificant and does not affect the subsequent dynamic behaviour of the simulator.

The highest priority process is that of the storage allocator (Priority = 14 i.e. the lower the priority number the higher the priority). This followed by INTIM process (Priority = 16) followed by the three RASH processes (Priority 41 - 43).

For a configuration of one CPU only, INTIM process has to wait for the storage allocator to finish any pending tasks before it can run to service the clock interrupt. The maximum delay that INTIM can experience is 4.3 msec which is the time required by the storage allocator to deal with a 'closefile' request from RASH. For a configuration of one CPU this delay is irrespective of the number of RASH instances whereas for a configuration of more than one CPU, it will depend on the number of requests pending and thus can be more than 4.3 msec. Of course INTIM does not wait for the storage allocator to finish if one of the RASH processes is running on another CPU as it will be pre-empted by INTIM on a priority basis. The storage allocator distinguishes between an open and a close file request by examining the value of G2 of the request task. A value of 4 indicates a close-file request whereas a value of 5 an open-file request.
A value of $G2 = 8$ will indicate a part-file read request. Although this is allowed for in the model, the request is not issued by the RASH processes. The time required by the storage allocator to honour a part-file request would have been drawn from an empirical distribution based on data collected from the real system.

Every time a RASH process issues a request it follows that by executing a BLOCK(3) call to the process allocator to be blocked until the storage allocator response of priority 3 arrives. The requests to the storage allocator have priority 10. Following the issue of the two requests RASH process checks for the arrival of the time task from INTIM which arrives every 100 msec. Whether it arrives or not the execution counter is incremented by one to indicate that the loop is completed once more and the loop is started again. The storage allocator picks up the tasks in its input queue using a BLOCK(10) instruction. When the simulated time expires a report is outputted displaying among other things the contents of the execution counters. A number of these reports for different configurations are included in Appendix (D).

6.2.2 Discussion of Validation Experiment Results

The results of the nine runs of the validation experiment are tabulated in Table (6.1). The figures are the values of RASH processes execution counters. The columns indicate the number of CPUs in a configuration and the rows between consecutive horizontal double lines, the number of RASH processes. For a configuration of one RASH process and one, two and three CPUs, we note an increase in the number of times RASH is executed when the number of CPUs is increased from one to two. An increase from 521 to 609 is noted in the real system and from 559 to 615 in the simulator.
<table>
<thead>
<tr>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>RASH 1</td>
<td>521</td>
<td>609</td>
<td>529</td>
<td>559</td>
<td>615</td>
</tr>
<tr>
<td>RASH 1</td>
<td>524</td>
<td>485</td>
<td>489</td>
<td>560</td>
<td>405</td>
</tr>
<tr>
<td>RASH 2</td>
<td>0</td>
<td>475</td>
<td>482</td>
<td>0</td>
<td>405</td>
</tr>
<tr>
<td>RASH 1</td>
<td>522</td>
<td>355</td>
<td>293</td>
<td>556</td>
<td>270</td>
</tr>
<tr>
<td>RASH 2</td>
<td>0</td>
<td>350</td>
<td>292</td>
<td>0</td>
<td>270</td>
</tr>
<tr>
<td>RASH 3</td>
<td>0</td>
<td>315</td>
<td>289</td>
<td>0</td>
<td>270</td>
</tr>
</tbody>
</table>

**TABLE 6.1** TIMES RASH PROCESSES EXECUTED FOR A CONFIGURATION OF 1, 2 and 3 CPUs with 1, 2 and 3 RASH PROCESSES.
This increase is attributed to the fact that, with two CPUs available, the storage allocator will always occupy one of them, leaving the other one to be contested by INTIM and RASH processes. INTIM is activated once very 10 msec with the arrival of a clock interrupt, and since there are no periodic processes for it to activate it occupies a CPU for a relatively shorter time. Hence, a RASH process is guaranteed to run (possibly with minor delays) every time it is handed a response task by the storage allocator. This is in contrast to the case of one CPU only in a configuration; here RASH is always pre-empted on priority basis by the storage allocator and can only be run when both the storage allocator and INTIM are idle.

It is worthwhile to note that when the number of CPUs is increased to 3, with only one RASH process running in the configuration, the value of the execution counter drops for the real system whereas it remains constant for the simulator. This difference is due to some remaining randomness in the real system such as that due to store contention. The store contention effect is expected to decrease appreciably with the application of faster and higher-capacity storage modules.

When two or three RASH processes are running in a configuration of one CPU the execution counter value for RASH2 or RASH3 is zero Table (6.1). This is because as soon as RASH1 hands a request task to the storage allocator, it is pre-empted by the latter. It is only allowed to run again when the storage allocator processes the request, hands back the response task and executes a BLOCK(10) call which is blocked as there are no further request tasks pending in the input queue, (429 - 558).
Since RASH1 is the first of the RASH processes to be selected to run on the basis of its priority, the other RASH processes will have no chance at all of being run throughout this particular configuration. We note that the request tasks from RASH have a priority of 10 whereas their responses from the storage allocator have a priority of 3. Periodic unblocking tasks from INTIM have priority of 5. There is a notable drop in the execution counters values when three RASH processes are run in a configuration of 3 CPUs instead of 2 CPUs in the real system. This drop is inexplicable in terms of the system scheduling algorithms and is only attributable to residual randomness due to factors such as the store contention. What is expected is that at least no drop should be experienced in the values of execution counters since more CPUs are available now. This expectation is more fulfilled in the simulator than in the real system.

The remainder of this section will be devoted to a statistical assessment of the credibility of "goodness of fit" of the Mark II BL simulator. The execution counters values of Table (6.1) are rearranged in a descending order of value in Table (6.2) and the table is arranged in such a way as to facilitate the calculation of different statistical parameters. The random variable X denotes the execution counter values for the real system whereas the random variable Y denotes those for the simulator.
<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$x = X - \overline{X}$</th>
<th>$y = Y - \overline{Y}$</th>
<th>$x^2$</th>
<th>$xy$</th>
<th>$y^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-408.44</td>
<td>-386.94</td>
<td>166823.11</td>
<td>158041.77</td>
<td>149722.50</td>
</tr>
<tr>
<td>289</td>
<td>270</td>
<td>-119.44</td>
<td>-116.94</td>
<td>14265.89</td>
<td>13967.31</td>
<td>13674.95</td>
</tr>
<tr>
<td>292</td>
<td>270</td>
<td>-116.44</td>
<td>-116.94</td>
<td>13558.27</td>
<td>13616.49</td>
<td>13674.95</td>
</tr>
<tr>
<td>293</td>
<td>270</td>
<td>-115.44</td>
<td>-116.94</td>
<td>13326.35</td>
<td>13499.55</td>
<td>13674.95</td>
</tr>
<tr>
<td>315</td>
<td>270</td>
<td>-93.44</td>
<td>-116.94</td>
<td>8731.02</td>
<td>10926.87</td>
<td>13674.95</td>
</tr>
<tr>
<td>350</td>
<td>270</td>
<td>-58.44</td>
<td>-116.94</td>
<td>3415.23</td>
<td>6833.97</td>
<td>13674.95</td>
</tr>
<tr>
<td>355</td>
<td>270</td>
<td>-53.44</td>
<td>-116.94</td>
<td>2855.84</td>
<td>6249.27</td>
<td>13674.95</td>
</tr>
<tr>
<td>475</td>
<td>405</td>
<td>66.56</td>
<td>18.06</td>
<td>4430.23</td>
<td>1202.07</td>
<td>326.16</td>
</tr>
<tr>
<td>482</td>
<td>405</td>
<td>73.56</td>
<td>18.06</td>
<td>5411.07</td>
<td>1328.49</td>
<td>326.16</td>
</tr>
<tr>
<td>485</td>
<td>405</td>
<td>76.56</td>
<td>18.06</td>
<td>5861.43</td>
<td>1382.67</td>
<td>326.16</td>
</tr>
<tr>
<td>489</td>
<td>405</td>
<td>80.56</td>
<td>18.06</td>
<td>6489.91</td>
<td>1454.91</td>
<td>326.16</td>
</tr>
<tr>
<td>521</td>
<td>550</td>
<td>112.56</td>
<td>163.06</td>
<td>12669.73</td>
<td>18354.03</td>
<td>26588.51</td>
</tr>
<tr>
<td>522</td>
<td>556</td>
<td>113.56</td>
<td>169.06</td>
<td>12895.85</td>
<td>19198.45</td>
<td>28581.21</td>
</tr>
<tr>
<td>524</td>
<td>560</td>
<td>115.56</td>
<td>173.06</td>
<td>13354.07</td>
<td>19998.81</td>
<td>29949.71</td>
</tr>
<tr>
<td>529</td>
<td>615</td>
<td>120.56</td>
<td>228.06</td>
<td>14534.68</td>
<td>27494.91</td>
<td>52011.23</td>
</tr>
<tr>
<td>609</td>
<td>615</td>
<td>200.56</td>
<td>228.06</td>
<td>40224.24</td>
<td>45739.71</td>
<td>52011.23</td>
</tr>
</tbody>
</table>

$\Sigma X = 6535$  $\Sigma Y = 6191$

$\Sigma x^2 = \Sigma xy = \Sigma y^2 =$

$\overline{X} = 408.44$  $\overline{Y} = 386.94$

$338846.92$  $359288.57$  $422218.73$

**TABLE (6.2)**: Values for Calculation of Regression of $Y$ on $X$ and Correlation Coefficient

233
With reference to Table (6.2).

For the real system:

\[
\text{The mean } = \bar{X} = \frac{1}{N} \sum_{i=1}^{N} X_i = 408.44
\]

\[
\text{The variance } = \sigma_X^2 = \frac{1}{N-1} \sum (X_i - \bar{X})^2 = 22589.79
\]

\[
\text{The standard deviation } = \sigma_X = 150.30
\]

The corresponding values for the simulator are as follows:

\[
\text{The mean } = \bar{Y} = \frac{1}{N} \sum_{i=1}^{N} Y_i = 386.94
\]

\[
\text{The variance } = \sigma_Y^2 = \frac{1}{N-1} \sum (Y_i - \bar{Y})^2 = 28147.92
\]

\[
\text{The standard deviation } = \sigma_Y = 167.77
\]

The two means of the real system \(\bar{X}\) and simulator \(\bar{Y}\) differ by

\[
\frac{|\bar{X} - \bar{Y}| \times 100}{\bar{X}} = \frac{408.44 - 386.94}{408.44} = 5.26\%
\]

Expressed in a different way, the two means agree to within

\[
\frac{|\bar{X} - \bar{Y}|}{\sigma_X} = 0.143 \sigma_X
\]

which is considered to be a very good agreement. (HORN 72).

We will now test the output of both systems for positive correlation, determine the correlation coefficient value and derive the least square linear regression equation of \(Y\) on \(X\).

The correlation coefficient \(r\) is defined as App.(E).

\[
r = \frac{\rho_{xy}}{\sqrt{(\rho_x^2)(\rho_y^2)}} \quad ...........(6.3)
\]

where the quantities \(xy\), \(x^2\) and \(y^2\) are as described in Table (6.2).

Substituting for \(xy\), \(x^2\) and \(y^2\)
\[ r = \pm \frac{359288.57}{\sqrt{338846.92 \times 422218.73}} = +0.9498 \]

The positive value is taken because \( y \) increases as \( x \) increases.

The linear least square regression line of \( Y \) on \( X \) is defined by the equation

\[ Y = a_0 + a_1X \tag{6.4} \]

where \( a_0 \) and \( a_1 \) are obtained from the normal equations

\[
\begin{align*}
\Sigma y &= a_0 N + a_1 \Sigma x \tag{6.5} \\
\Sigma xy &= a_0 \Sigma x + a_1 \Sigma x^2 \tag{6.6}
\end{align*}
\]

When these two equations are solved, they result in \( a_0 \) and \( a_1 \) as having the values

\[
\begin{align*}
a_0 &= \frac{(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma xy)}{N \Sigma x^2 - (\Sigma x)^2} \tag{6.7} \\
a_1 &= \frac{N \Sigma xy - (\Sigma x)(\Sigma y)}{N \Sigma x^2 - (\Sigma x)^2} \tag{6.8}
\end{align*}
\]

Using the transformation \( x = X - \bar{X}, y = Y - \bar{Y} \)

where \( \bar{X} \) and \( \bar{Y} \) are the mean values, equation (6.4) becomes

\[ y = \left( \frac{\Sigma xy}{\Sigma x^2} \right) x \tag{6.9} \]

substituting the values of \( \Sigma xy \) and \( \Sigma x^2 \) from table (6.2)

\[ y = \frac{359288.57}{338846.92} \cdot x = 1.06 \cdot x \]

or

\[ Y - \bar{Y} = 1.06(X - \bar{X}) \]

\[ Y = 1.06X + \bar{Y} - 1.06\bar{X} \]

\[ = 1.06X + 386.94 - 1.06\times408.44 \]

\[ Y = 1.06X - 46.01 \tag{6.10} \]

This is the least-square regression line of \( Y \) on \( X \), which is also plotted in Graph (6.1). In the verification by regression analysis the "goodness of fit" criteria of a simulation model is that the slope of the regression line is nearly unity and the intercepts are not
GRAPH (5.1)

REGRESSION LINE OF Y (SIMULATOR) ON X (REAL SYSTEM)

\[ Y = 1.66X + 46.01 \]
significantly different from zero (COHE 61). From the high positive correlation coefficient value (+0.9498), the values of \( \bar{Y}, \bar{X} \) which differ by 0.1430 only and the form of equation (6.10) where the slope is almost unity one can conclude that the credibility of the simulator or its "goodness of fit" to the real system is quite good.

The process described in this chapter seems deceptively simple. Validating a complex system such as the Mark II BL multi-processor system proved to be both a complex and time-consuming process (NOE 74). The situation is best described in Macdougal's words (MACD 74): "While the simulation model represents a reduction in size of several orders of magnitude relative to the actual system, the reduction in complexity is not equally great. The complexity of the actual system results from the interactions of its parts and the representation of these interactions is a major part of the modelling effort. There is no substitute for care in program design and implementation, and all the other prosaic aspects of the programming art".

This validation exercise, apart from building confidence in the model, enhances greatly the understanding of the system dynamic behaviour and helps to identify areas where potential performance improvements are possible as will be shown in Section 6.3. The verification and validation process is depicted in Figure (6.18).
FIGURE (6.18)  Verification And Validation Process
6.3 Experiments Conducted Using the Mark II BL System Simulator

In this section two experiments will be reported. They both make use of the Mark II BL Simulation package developed and are meant to give a flavour of the type of problems that can be tackled and to demonstrate the flexibility and ease of use of the package in the System X sub-systems design - optimisation process.

6.3.1 Investigation of a New Call FBLOCK(P) to the Process Allocator

With reference to Section (4.3.2.1), FBLOCK(P) call to the process allocator is equivalent to:

FETCH(15)
JNZ(C+Z)
BLOCK(P)

It is proposed to be used largely in conjunction with periodic processes which normally pick up all the tasks in their input queues when periodically activated by INTIM process (4.3.2.2), and then block awaiting the next activation. Thus, when the process allocator is issued an FBLOCK(P) call, it services a FETCH(15) call routine first. If a task is found in the calling process's input queue, the condition codes are set positive and the process allocator exits back to the process. On the other hand, if no task is found in the input queue, the process allocator sets the process blocked with a priority level P and schedules on that CPU. P is the priority of INTIM's periodic unblocking task.

The saving in the process allocator is that if FBLOCK(P) call is negative that is no task is found in the input queue, the process allocator does not exit back to the process which will
then execute a BLOCK(P) call, but rather sets it blocked and schedules the CPU without the need to engage the process's PROLO Lockout and inspect the input queue (Figure (6.19)).

A simulation experiment is designed to investigate this new call to the process allocator and quantify the reduction in its overhead. The periodic processes NOFBLCK (Figure (6.20)) and WITHFBLCK (Figure (6.21)) are an adaptation of the RASH processes. They are used in this experiment with a periodicity of 10 msec.

When activated by INTIM, the test processes check their input queues. If a task is present, they will either issue an 'Open-File' or 'Close-File' request to the storage allocator. This will be determined by the value of G(2) of the task in the input queue, which is usually a response task from the storage allocator to a previous request.

To enable the storage allocator to service up to five periodic test processes during the 10 msec clock period, the storage allocator times are reduced from 4050.0 and 4300.0 microseconds to 650.0 and 700.0 microseconds respectively. Despite that, it is noted from the output trace that the storage allocator over ran the clock period in servicing five test processes in a configuration of 2 CPUs. This particular case resulted in a rather over-optimistic figure of 23.88% in the process allocator overhead reduction. Therefore, this particular overshoot value is discarded in the final analysis.

However, if a larger number of periodic processes are required this can be attained by increasing the periodicity of the test processes and/or reducing the storage allocator service times.
FIG (G.19): FBLOCK(P) CALL AND FBLOCK(P) PROCEDURE

1. **FBLOCK(P)**
   - CIX := 6
   - G := P

2. **HOLD(G, 0)** μs
   - R - TIME

3. **PROCIG** (CALLING PROCESS)

4. **Q>EQ** (QUEU EMPTY)
   - N

5. **HOLD(G, 0)** μs
   - P (TIME)

6. **HOLD(G, 0)** μs
   - PA - TIME

7. **OF FIG(S)**

FIG (G.24): INITIATOR PROCESS MODEL

1. **INITIATOR**
   - GI := 1
   - (G2) := 4
   - G1 := 0

2. **HAND**

3. **END**

RETURN
To initialise the experiment, the INITIATOR process is incorporated in the model. Its function is to hand 'Close-File' response tasks to the test process, on behalf of the storage allocator (Figure 6.22), at the start of a run. Thus when they are activated by INTIM and have checked their input queues, response tasks will be found to which they will respond by sending further request tasks to the storage allocator. The contents of a request task is defined as follows:

\[
\begin{align*}
G(0) & = 1 \\
G(1) & = \text{PI} - \text{the process index of a test process} \\
G(2) & = 4 \text{ or } 5 - \text{ for open or close file request} \\
G(3) - G(7) & = \text{not used};
\end{align*}
\]

Two sets of experiments are carried. In the first set up to five instances of NOFBLCK process are used. When this process is activated it executes a FETCH(15) as in Figure (6.20). If a task is available in the input queue, it hands a close or open file request to the storage allocator according to the value of the general register G(2). A counter is then incremented and the above mentioned sequence of actions repeated for positive FETCH(15) calls. If, however, the fetch call is negative, the process instance calls to be blocked until the arrival of the next INTIM's periodic unblocking task of priority level 5.

In the second series of experiments, up to five instances of the periodic test process WITHFBLCK (Figure 6.21) are used. This process is similar to NOEBLCK process, the only difference is that it uses the new call, FBLOCK(P). In both process types, since the storage allocator services all the test processes once during a clock period, the counter of each process instance is incremented once during that period.
The process allocator overhead portions due to FETCH(l5), JNZ(C+Z), BLOCK(P) and the equivalent FBLOCK(P) are tabulated in Table (6.3) and Table (6.4) respectively. The percentage reduction in this overhead portion, calculated on the basis of the average overhead per CPU in a configuration is tabulated in Table (6.5) and plotted in Graph (6.2).

The reduction in the overhead due to the use of FBLOCK(P) call is in the range 19% to 23.59%. All the experiments are run for a simulated time of 7.2 secs, thus all process instances counters read 720. If we assume that the process allocator overhead for FETCH(l5), JNZ(C+Z), BLOCK(P) is X μsecs and that due to FBLOCK(P) is X - δμsecs, then

\[
\% \text{ reduction in overhead} = \frac{100{\delta}}{X}
\]

There appears to be a variable element in the value of δ according to the number of events of finding and not finding a task in the input queue when executing FBLOCK(P) call. The variation in the number of events is dependent on the configuration of a particular run. This fact explains the variation in the overhead reduction between 19% and 23.59%. Nevertheless, the reduction in the process allocator overhead by using FBLOCK(P) call is significant. Further, the incorporation of this call into the microprogrammed process allocator does not entail a major modification.

**6.3.2 Tuning of Interrupts Handling in Mark II BL System**

During the model verification phase, when the output trace of the system dynamic behaviour is closely examined (Chapter 6), it is felt that one of the areas where the system performance improvement is possible is the interrupts handling mechanism of the real-time operating system. For example the Suspended Queue
<table>
<thead>
<tr>
<th>PROCESSES</th>
<th>NOFBLCK 1</th>
<th>NOFBLCK 2</th>
<th>NOFBLCK 3</th>
<th>NOFBLCK 4</th>
<th>NOFBLCK 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>161505.12</td>
<td>281770.75</td>
<td>589584.69</td>
<td>336687.94</td>
<td>602675.12</td>
</tr>
<tr>
<td>2</td>
<td>161666.62</td>
<td>383558.69</td>
<td>370890.37</td>
<td>1033042.00</td>
<td>601201.75</td>
</tr>
<tr>
<td>1</td>
<td>161486.94</td>
<td>210251.19</td>
<td>315323.62</td>
<td>315652.81</td>
<td>527104.50</td>
</tr>
<tr>
<td>2</td>
<td>0.00</td>
<td>210355.69</td>
<td>294672.19</td>
<td>315929.12</td>
<td>527062.56</td>
</tr>
<tr>
<td>3</td>
<td>161666.62</td>
<td>210143.94</td>
<td>315377.69</td>
<td>623333.00</td>
<td>527477.62</td>
</tr>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>232481.06</td>
<td>314629.37</td>
<td>623927.25</td>
</tr>
<tr>
<td>2</td>
<td>161486.94</td>
<td>211922.25</td>
<td>232500.06</td>
<td>314641.31</td>
<td>321115.12</td>
</tr>
<tr>
<td>3</td>
<td>0.00</td>
<td>211963.50</td>
<td>232069.75</td>
<td>314606.50</td>
<td>321119.56</td>
</tr>
<tr>
<td>4</td>
<td>161666.62</td>
<td>211953.62</td>
<td>232071.37</td>
<td>314607.69</td>
<td>321113.00</td>
</tr>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>298655.81</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>297376.37</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>161486.94</td>
<td>211922.25</td>
<td>233201.62</td>
<td>307523.69</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.00</td>
<td>211963.50</td>
<td>232069.75</td>
<td>300642.81</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>161666.62</td>
<td>211963.62</td>
<td>232071.37</td>
<td>1633.86</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE (6.3)**  
PROCESS ALLOCATOR OVERHEAD (μs) DUE TO FETCH(15)  
\( \text{JNZ(C+Z), BLOCK(P)} \)
<table>
<thead>
<tr>
<th>PROCESSES</th>
<th>WITHFBLCK 1</th>
<th>WITHFBLCK 2</th>
<th>WITHFBLCK 3</th>
<th>WITHFBLCK 4</th>
<th>WITHFBLCK 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUS</td>
<td>(~s)</td>
<td>(~s)</td>
<td>(~s)</td>
<td>(~s)</td>
<td>(~s)</td>
</tr>
<tr>
<td>1</td>
<td>130898.56</td>
<td>301845.50</td>
<td>452905.19</td>
<td>261743.87</td>
<td>458820.12</td>
</tr>
<tr>
<td>2</td>
<td>130839.94</td>
<td>227131.00</td>
<td>309479.87</td>
<td>845151.31</td>
<td>457532.62</td>
</tr>
<tr>
<td>3</td>
<td>130839.94</td>
<td>165027.19</td>
<td>247841.19</td>
<td>247556.94</td>
<td>422438.94</td>
</tr>
<tr>
<td>4</td>
<td>130839.94</td>
<td>0.00</td>
<td>233274.19</td>
<td>248001.37</td>
<td>423194.37</td>
</tr>
<tr>
<td>5</td>
<td>130839.94</td>
<td>165131.62</td>
<td>245558.69</td>
<td>502040.50</td>
<td>422817.12</td>
</tr>
<tr>
<td>1</td>
<td>0.00</td>
<td>165598.94</td>
<td>182369.44</td>
<td>165598.94</td>
<td>316083.69</td>
</tr>
<tr>
<td>2</td>
<td>130898.56</td>
<td>165071.12</td>
<td>182372.75</td>
<td>165230.06</td>
<td>316086.94</td>
</tr>
<tr>
<td>3</td>
<td>0.00</td>
<td>165069.81</td>
<td>182273.81</td>
<td>500110.19</td>
<td>316093.81</td>
</tr>
<tr>
<td>4</td>
<td>130839.94</td>
<td>165069.94</td>
<td>182274.75</td>
<td>165229.25</td>
<td>316085.94</td>
</tr>
<tr>
<td>5</td>
<td>0.00</td>
<td>0.00</td>
<td>2351.38</td>
<td>232780.56</td>
<td>232780.56</td>
</tr>
<tr>
<td>1</td>
<td>0.00</td>
<td>182339.81</td>
<td>182339.81</td>
<td>229696.37</td>
<td>229984.06</td>
</tr>
<tr>
<td>2</td>
<td>0.00</td>
<td>182339.81</td>
<td>230294.25</td>
<td>230816.37</td>
<td>230816.37</td>
</tr>
</tbody>
</table>

**TABLE (6.4)**: PROCESS ALLOCATOR OVERHEAD (μs) DUE TO FBLOCK(P)

246
<table>
<thead>
<tr>
<th>PROCESS TYPE &amp; NO. OF PROCESSES</th>
<th>NK NOFBLCK PROCESS</th>
<th>WL WITHFBLCK PROCESS</th>
<th>OVERHEAD REDUCTION (%)</th>
<th>NK NOFBLCK PROCESS</th>
<th>WL WITHFBLCK PROCESS</th>
<th>OVERHEAD REDUCTION (%)</th>
<th>NK NOFBLCK PROCESS</th>
<th>WL WITHFBLCK PROCESS</th>
<th>OVERHEAD REDUCTION (%)</th>
<th>NK NOFBLCK PROCESS</th>
<th>WL WITHFBLCK PROCESS</th>
<th>OVERHEAD REDUCTION (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>161585.87</td>
<td>130869.25</td>
<td>19.00</td>
<td>107717.85</td>
<td>87246.12</td>
<td>19.00</td>
<td>80788.39</td>
<td>65434.59</td>
<td>19.00</td>
<td>64630.71</td>
<td>52347.67</td>
<td>19.00</td>
</tr>
<tr>
<td>2</td>
<td>332664.72</td>
<td>264488.15</td>
<td>20.50</td>
<td>210230.27</td>
<td>165026.22</td>
<td>21.50</td>
<td>158962.34</td>
<td>123802.71</td>
<td>22.12</td>
<td>127169.87</td>
<td>101735.89</td>
<td>20.00</td>
</tr>
<tr>
<td>3</td>
<td>480237.53</td>
<td>381192.53</td>
<td>20.64</td>
<td>308457.83</td>
<td>242224.69</td>
<td>21.47</td>
<td>232280.54</td>
<td>182322.62</td>
<td>21.51</td>
<td>186115.86</td>
<td>144133.66</td>
<td>22.56</td>
</tr>
<tr>
<td>4</td>
<td>684864.95</td>
<td>553447.55</td>
<td>19.19</td>
<td>418304.96</td>
<td>332532.93</td>
<td>20.51</td>
<td>314621.20</td>
<td>249042.11</td>
<td>20.84</td>
<td>241166.48</td>
<td>184283.71</td>
<td>23.59</td>
</tr>
<tr>
<td>5</td>
<td>601938.40</td>
<td>458176.37</td>
<td>23.88</td>
<td>527214.86</td>
<td>422816.80</td>
<td>19.80</td>
<td>399068.85</td>
<td>316087.50</td>
<td>20.79</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**KEY**

NK: NOFBLCK PROCESS

WL: WITHFBLCK PROCESS

Table (6.5): AVERAGE PROCESS ALLOCATORS OVERHEAD AND THEIR PERCENTAGE REDUCTIONS
GRAPH (6.2)

% REDUCTION IN USING FBLOCK(P)
WITH PERIODIC PROCESSES

% REDUCTION

0  2  4  6  8  10
NO. OF PERIODIC PROCESSES
Interrupt is found to be triggered unnecessarily in certain cases, with the subsequent delay to the pre-empted process. Examining the runs traces closely, it is found that the interrupt triplicates reference to the CPU running the lowest priority process (LCPU) is not updated as early as it ought to be in the scheduling sequence of a CPU. Let us explain this by a simple example.

Suppose that LCPU (Sec. 5.4.9) is CPU1 and its current running process is process X. Suppose a CPU schedule 'CPUSCHED' (Sec. 5.4.10) executed for LCPU and another process Y is selected from the suspended state map to run on CPU1. If, however, a third process, process Z, say, is suspended in another CPU, CPU2, before updating LCPU and the further process Z is of a priority higher than process X but lower than process Y, then the suspended queue interrupt will be triggered by CPU2 and steered to CPU1. It is steered to CPU1 because the LCPU reference is not yet updated. Thus-process Y will be pre-empted from CPU1 when the suspended queue interrupt is serviced but will be selected again for CPU1 because it is of a higher priority than process Z. This unnecessary nest and de-nest of a process and the subsequent delay is avoidable if LCPU reference is updated as soon as a process is selected to run on a CPU. This is confirmed by trial runs on the model where LCPU is updated immediately after SUSPLO2 in CPUSCHED (Sec. 5.4.10).

Another aspect of tuning the interrupt handling mechanism of the Mark II BL multiprocessor system is to interrupt a running process only if it is a background process. The modification to effect this change is a minor one both in the real system and the simulator. This is an example of the ease in incorporating design modifications and extensions in the simulator brought about by its process based nature and its multi-level hierarchical structure as explained in
Chapter 1. In the system scheduling procedure SYSCHED of section 5.4.10 instead of:

IF( I < INTRIP. LCPU. CURP. PI AND I NE 0 AND I NE 127) we include:

IF( I < INTRIP. LCPU. CURP. PI AND INTRIP. LCPU. CURP. PI > 116)

where 116 is the process index of the highest priority background process.

The two proposed modifications to the interrupt handling mechanism, namely, updating LCPU as soon as a new process is selected and interrupting LCPU only if it is running a background process are incorporated in the Mark II BL simulator. Two sets of three experiments are carried out using RASH processes as in Chapter 6. In the first set, for a configuration of 2 CPUs and 2 background processes, 3 runs of the simulator are made with 3, 4 and 5 RASH instances respectively and using the existing interrupt handling mechanism. The same experiments are then repeated with the two proposed modifications incorporated in the interrupt handling mechanism. The statistics of interest are summarised in Table (6.6). It is evident that the modifications resulted in an increase in the throughput and an appreciable decrease in the percentage of total process allocators interrupts to total process allocators calls. The increase in the throughput is due to the fact that the process allocator overhead in servicing the interrupts is now being used to do useful processing. However, the advantages of not interrupting a running process unless it is a background one are offset at times of high traffic, when low priority processes are still running while high priority ones are waiting, with a possible loss of traffic. Nevertheless the proposal of updating LCPU as soon as possible is easy to implement in the microprogrammed process allocator without any side effects. The proposal has been
### Table (6.6) Summary of Statistics for the Interrupt Handling Mechanism Modifications

<table>
<thead>
<tr>
<th>ITEM</th>
<th>MODIFIED INTERRUPT HANDLING</th>
<th></th>
<th>EXISTING INTERRUPT HANDLING</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3 RASHES</td>
<td>4 RASHES</td>
<td>5 RASHES</td>
<td>3 RASHES</td>
</tr>
<tr>
<td>No. of CPUs</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Throughput (RASH CYCLES)</td>
<td>359</td>
<td>359</td>
<td>353</td>
<td>246</td>
</tr>
<tr>
<td>Total PA Calls</td>
<td>3316</td>
<td>3332</td>
<td>3275</td>
<td>2300</td>
</tr>
<tr>
<td>Total PA Interrupts</td>
<td>54</td>
<td>60</td>
<td>55</td>
<td>435</td>
</tr>
<tr>
<td>Background Processes</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Interrupts Calls</td>
<td>1.63</td>
<td>1.8</td>
<td>1.68</td>
<td>18.9</td>
</tr>
</tbody>
</table>
welcomed by the software design engineer in charge of the process allocator and may well be implemented in the process allocator for the System X Release 2.
7.1 THE DIGITAL MAIN NETWORK-SWITCHING CENTRE (DMNSC)

7.1.1 Introduction

The Digital Main Network Switching Centre is a digital trunk exchange, a member of the System X family of exchanges. It operates in a mixed analogue and digital environment and comes in a wide range of sizes, with the large trunk exchange anticipated to have a termination capacity of 85,000, a switch capacity of 20,000 erlangs and a processing capacity of 500,000 busy-hour call attempts (TIPP 79). A schematic block diagram of the DMNSC is shown in Figure (3.9). The exchange is assembled from a common set of System X sub-systems. The sub-systems are either hardware or software sub-systems, with most hardware sub-systems having their own software handlers which are stored and run on the processor sub-system. Examples of such sub-systems are SIS (Signal Interworking Subsystem), DSS (Digital Switching Subsystem) and MTS (Message Transmission Subsystem). The functions of these and other sub-systems have been explained in (3.6.3). The status of the DMNSC in the network hierarchy, as a trunk exchange or a regional switching centre serving a number of trunk exchanges, can be changed by software modifications, provided the junction and trunk routes can cope with the new pattern of traffic flow. All existing channel associated signalling systems are provided over analogue and digital transmission systems plus digital common channel signalling. The DMNSC is controlled by Mark II BL multiprocessor systems with the number of CPUs being dependent on the traffic carried, with pre-processing employing microprocessors. The software sub-systems have secured message interfaces. The DMNSC operates as a mutually synchronised
centre at any assigned level in the synchronised network.
The overall management of the DMNSC is exercised by the Local Administration Centre together with a local man/machine control point using a man/machine high level language. The exchange itself is constructed from general purpose sub-systems allowing a wide range of applications in the existing and proposed network configurations, and allowing an evolutionary updating of the network. Another important advantage of this modular construction is that it allows new technology to be introduced as it becomes viable and proven.

Apart from the design and development of the processor sub-system, GEC is also responsible for the design and development of the Call Processing Sub-system (CPS). This is a totally software sub-system responsible for all telephony functions. This sub-system is central to the efficient functioning of the System X family of exchanges. Figure (7.1) is a typical message sequence chart of an incoming (junction) decadic to an outgoing (junction) decadic call through the DMNSC. The message sequence chart indicates the most important sub-systems with which CPS communicates. In Figure (7.1) there are eight pairs of instances in a call processing sequence marked by crosses and numbered 1 to 8. The delay distributions between each two events in a pair is indicative of the performance of the DMNSC and consequently whether that performance meets the British Post Office requirements or not.

The objectives of simulating the DMNSC are twofold. On the one hand, it is essential to evaluate the performance of the DMNSC by quantifying the delays between particular instances in the call processing sequence as mentioned above. On the other hand, the simulation of the DMNSC is required to tune the call processing sub-system. In the following paragraphs the DMNSC is described from the point of view of the CPS, as it is our main concern here.
FIG 7.1 MESSAGE SEQUENCE CHART OF AN Incoming (JUNCTION) DECADIC TO AN OUTGOING (JUNCTION)
The Call Processing Sub-system (CPS) is responsible for supervising all the control functions relating to the passage of telephone calls through the exchange in which it resides. It is a purely software sub-system and does not interface directly with the hardware necessary for setting up the call. Other sub-systems, namely, SIS MTS and DSS have this responsibility, CPS interworking with them by means of task transfers when required. A modular approach has been adopted in the design of CPS to enable as much commonality as possible to be achieved between the various variants of CPS for the various applications such as the DMNSC and the local exchange. It was also necessary to establish clearly the true essentials of the call processing function, with the objective that the basic structure of the sub-system should be as general purpose as possible. Any differences in the call processing functions associated with the various signalling systems and circuit types should not be reflected into the foundations of the CPS design. This has resulted in the adoption of the strategy that, in order to set up a call, the call processing entities in each exchange, through which the call passes, require to communicate with each other in the best possible way. For calls originating in the new network, the medium by which this is achieved is the MTS sub-system which is effectively transparent to CPS. On the other hand, the SIS sub-system is the medium for calls originating in the old network with in-band signalling over the actual speech path. The interfaces between CPS and MTS and between CPS and SIS are designed to be as similar as possible using a common repertoire of call protocol messages with any peculiarities of the existing signalling systems being effectively hidden in SIS.
CPS communicates through a defined interface with the DSS sub-system to physically set up and clear down the switch paths between the incoming and outgoing circuits. It also provides features for dealing with line circuit maintenance and the on-line updating of certain data areas under the control of MCS. It also sends statistical data related to calls it processes to the MSS sub-system and co-operates with the OCS sub-system in controlling overload situations. The above mentioned interfaces of CPS with other sub-systems will be elaborated upon later. First it is worthwhile to consider the sequence of events in processing a call in the DMNSC in more detail.

7.1.2 Basic Sequence of Events in Call Processing in the DMNSC

With reference to the message sequence chart of Figure (7.1), the basic sequence of events is as follows:-

The start of a call is indicated by CPS receiving an 'Initial Address Message' (IAM). This may be either a simple seizure message or a message containing a number of dialled digits. This message, as with all other messages, contains the identity of the appropriate line circuit. Further dialled digits are received in subsequent address messages from SIS. According to the line signalling type, the digits may be sent as soon as possible to this exchange, each digit being received in a separate address message, or if the interworking with the previous exchange can be more interactive as in the new network, then the digits may be sent on request by CPS at this exchange. In this latter case, several digits may be sent in one address message, this is en bloc working.
When sufficient digits have been received by CPS, it sends a part-file read request message to the storage allocator. This is a message to translate the digits received to network routing information which provides the details of up to five possible routings for the call in priority order. The details for each routing consist of the outgoing route identity plus the particulars on how and what digits should be sent on to the next exchange. It may be that a new set of digits has to be injected, and if so, these would be provided in the routing information.

An attempt is made to select a free circuit on the first choice outgoing route. If no such free circuit is available, the attempt is repeated for the remaining subsequent routes until all have been attempted. This is called Automatic Alternative Routing. The selected circuit is busied in software to prevent it being selected on subsequent calls using that route. The circuit selection algorithm uses two modes of working, a cyclical selection followed by a sequential selection (DENT 78A). On a successful outgoing circuit selection, the CPS instance in charge of the outgoing route sends back the message "Start Sending" with the outgoing circuit identity to the incoming CPS instance. Whether or not the incoming CPS instance is the same as the outgoing CPS instance depends on the outgoing route identity in the routing information, that is whether the incoming and outgoing routes are the responsibility of the same CPS instance or not. A CPS instance can have up to 256 routes, each route having up to 256 bands with 16 circuits in a band (DENT 78B).

Having received the "Start Sending" message, the incoming CPS instance sends an initial address message (IAM) to SIS on the outgoing circuit, containing the appropriate digits. At this point in time, a message could be received from the next exchange via SIS conveying an unsuccessful set-up such as 'Congestion' or 'Subscriber Engaged'.
If that message does come through, CPS can either abort the current set up to the outgoing circuit and repeat on the same route, attempt to re-route on an alternative route or abort the call completely applying tone to the incoming circuit via DSS if necessary (DENT 78A).

Assuming a successful set-up as in Figure (7.1), the remaining digits received on the incoming circuit via incoming SIS are passed on in subsequent address messages to SIS on the outgoing circuit. The last digit is sent in a 'Final Address Message', FAM, to which SIS on the outgoing circuit responds back by a 'Number Receiver' message indicating that no further address messages are required in order to set-up the call. This message is sent end-to-end to the SIS on the incoming circuit side.

On receiving the 'Number Received' message the incoming CPS instance sends a 'set-up switch path' message to the DSS software handler to set-up the switch path between the incoming and outgoing circuits.

When DSS responds back with a 'Switch Response' message, the incoming CPS instance sends the 'Set-up Complete' message to the outgoing CPS which will have control of the call from here onwards.

When the called subscriber answers, the outgoing SIS sends the 'Answer' message which is sent end-to-end to the incoming SIS which is also responsible for starting the call charging by sending the 'Meter Over Junction' message to the originating local exchange. A similar message is sent when the subscriber clears down. This also initiates the call clear and circuit release sequences. Usually, the exact sequence of messages depends on the order in which the calling and called subscribers clear down, and whether the particular DMNSC is in control of the release. Assuming the calling party clears first, a 'Release' message is sent end-to-end. This message is sent when a network switching node has started to release its switch connection. On receipt of this message CPS initiates the
release of its switch connection. An SIS equivalent is a clear forward from an incoming SIS circuit. The outgoing CPS instructs the DSS to 'Clear Switch Path' and when DSS responds back with a 'Switch Response' message, usually within a comparatively short time, the outgoing CPS sends a 'Call Released' message to the incoming CPS instance. The 'Circuit Free' message is sent when a switch connection has successfully released and a circuit is free to accept a new call. On receipt of this message, CPS frees the circuit in software and hence makes the circuit available for the selection of a new call if and when the switch connection has been successfully released.

The messages that are passed between CPS and SIS/MTS and relate to telephone calls are collectively known as call protocol messages. One of the design objectives is to maximise the commonality between the sequencing of messages on both the new and old networks, that is between CPS and SIS for the old network and between CPS and MTS for the new one. A detailed description of this common repertoire of messages is given in (DENT 78C). Each message is described in terms of the contents of the eight general registers of Mk. II BL, that is G0-G7. The messages sent between CPS and SIS or MTS are further classified into five categories according to the role which they play in the progress of a call. The first of these categories is the Forward Address Messages and these are used to forward dialled digits and service information through the network. The second category is the Forward Set-up Messages which are involved in the set-up phase of a call but do not carry any dialled digits. In general, they tend to be associated with rather particular functions such as the 'Receiver/Sender Forward Release' message which is only employed on a call incoming on a multi-frequency (MF) circuit, in which SIS has to inform CPS that it has released the receiver/sender. The third category is the Backward Set-up Request
Messages which are requests for forward set-up information from the previous exchange in the form of forward address or set-up messages such as 'Send n Digits' message. The fourth category is the Backward Set-up messages used to indicate a condition to the previous exchange such as 'Number Received' message. The fifth category is that of 'Call Supervision Messages' and these are mainly concerned with the supervision of a call. They can be forward, backward or both forward and backward messages such as 'Answer' and 'Clear'.

7.1.3 Sequencing of Messages

The message sequence chart of Figure (7.1) is one of several different types of call sequencing through the DMNSC. Sequencing of the other types of calls are covered by similar sequence charts. A considerable amount of detail on the sequencing of the call protocol messages is presented in the system design description. (DENT 78B).

Certain factors have influenced the design of the message sequence charts. The design of the sequencing of messages in the set-up phase is largely controlled by the need to minimise post dialling delay, and to minimise the number of messages carried by MTS in the new network so as to keep the processor sub-system load and MTS resources to a minimum. The requirement to minimise the post dialling delay is particularly critical in the old network, such as in the interchange of messages between CPS and SIS, because of the slow step-by-step nature of the path set-up. Therefore, for the old network, it is important to forward the dialed digits, each in a separate message, as soon as possible. This is in contrast with the new network, where since digital switching at each exchange consists of a single operation, the minimising of post dialling delay is
far easier to achieve. Thus the number of messages on MTS may be reduced by packing up the digits into a smaller number of messages.

To resolve the conflict when a call is switched both through the old and new sections of the network en route, while the network is evolving, the set-up protocols are divided into incoming and outgoing set-up protocols which are allocated to the circuit types. The flexibility inherent in the set-up protocols on the new network enables the problems of interworking old and new network signalling systems on the same telephone call to be overcome. Here the identity of the outgoing set-up protocol influences the mode of working of the incoming set-up protocol that is to effectively tune the flexibility built in the incoming set-up protocol to the call advantage. For example if the call is incoming on a new network, the outgoing set-up protocol must be established whether new or old before deciding to request the digits en bloc or individually from the incoming side.

For a symmetrical call supervision and clear sequence, the concepts of calling, called, first and last party clear, and of circuit selection at the calling, called or both ends of a circuit are introduced. For example on receipt of a 'Clear' message, the action taken will depend on the 'clear program' stored at the receiving switching node. The clear program is loosely defined as the action to be taken by CPS on receipt of the 'Clear' message. The clear program would have been set-up as a result of path of entry information, routing information or service instructions received (DENT 78A). The messages that appear in Figure (7.1) are by no means comprehensive. Figure (7.1) covers only the messages interchanged in one typical type of call. For example one such message that does not appear in Figure (7.1) is the 're-answer' message. This message is sent in either direction when a network terminal re-establishes its hold
condition before the switch path started to release, that is it cancels out a previous 'Clear' message. It is acknowledged end-to-end on the new network.

7.1.4 Circuit Selection

There are three basic modes of circuit selection of outgoing calls which are the responsibility of CPS. These are: single-way working, in which a circuit is available for selection at one end only (forward circuit selection); both-way working, where the selection is possible at both ends and backward circuit selection, in which it is the responsibility of the incoming exchange to select the circuit. CPS retains information per circuit on the mode of selection to be applied for each circuit.

To make a circuit available for selection on the incoming side the concept of pseudo circuits is introduced (DENT 78A). Pseudo circuits are provided on the route and the circuit is made available for selection here. On the arrival of an incoming seizure message on a pseudo circuit, all the circuits available for outgoing calls will be available for selection for incoming calls. Pseudo circuits are treated in the same fashion as all other circuits from which they are distinguished by having a different 'line circuit type'. They are chosen by the circuit selection algorithm only when all other circuits available for selection are busy. If a pseudo circuit is selected for an outgoing call, the switch connection will be inhibited until the next exchange selects a real circuit in place of the pseudo one. This is indicated by a 'Change Circuit' backward message to communicate the identity of the real circuit to this exchange. This exchange then checks the validity of the selection by ensuring that the circuit selected is a real free one, in which case a message 'Circuit Changed' will be sent on the pseudo circuit.
Otherwise, if it is another pseudo circuit, then it can be
assumed that no real circuits are available and the 'Release'
message is sent forward on the pseudo circuit.

7.1.5 The Virtual Circuit Concept

A prerequisite to the incorporation of alternative routing in a
network is the ability to prevent traffic from overflowing from a
particular route to other routes when the route planned limits
are exceeded. These limits are determined by the maximum traffic
expected to be offered to the route and the number of circuits the
route is expected to have. The most complex routing situation is
where a call may be offered to up to five routes in turn, the
primary, three intermediate routes and a final route. The simplest
solution and that which is being adopted in CPS is to incorporate a
simple cut-off of overflow traffic if the planned limits of a route
are exceeded.

When the planned limits of a route are determined, a margin of
capacity is allowed for limited failures or surges of traffic
within the network. Now, consider a route of p circuits, let
the planned traffic offered, after all alternative routes have been
tried, have a grade of service g. This same grade of service may be
obtained for a fully provided route with p+q circuits. Therefore,
the route of p circuits, with alternative routing, has
virtually p+q circuits. Use is made of this concept in setting
the limit of cut-off overflow traffic. Thus once p+q calls have
been offered to that route and are connected to that route and alternative
routes, the planned limit will have been reached and further calls
on that route will meet congestion irrespective of whether there are free
alternative paths available. q is therefore, a measure of the capacity
of the network to carry the overflow traffic.
Another reason for excessive overflow is the loss of circuits in a route. For small routes, the number of virtual circuits is decremented for each circuit lost. For large routes 'stepped integer adjustment' is implemented. Below a 'step threshold x' each circuit lost is subtracted from the virtual circuit allocation. At the step threshold value, the virtual circuits are decremented by the step value n plus the normal single circuit decrement. Above the step threshold an equal number of any circuits lost will be subtracted from the virtual circuits allocation. The above relationships may be expressed as follows:

Let

- $RA$ = no. of circuits allocated with O/G access
- $RI$ = " " " in service " " "
- $VA$ = " " virtual circuits allocated
- $VI$ = " " " available.
- $x$ = step threshold
- $n$ = step value.

a) when $RA - RI < x$, then $VI = VA - (RA - RI)$

$$ (VI - RI) = (VA - RA) $$

b) when $RA - RI \geq x$, then $VI = (VA - RA) - n$

$$ (VI - RI) = (VA - RA) - n $$

$(VI - RI)$ is the allowed overflow from a route partially in service and $(VA - RA)$ is the allowed overflow from a route fully in service.

7.1.6 Communication Between CPS and Other Sub-systems

DSS is the means by which CPS carries out physically any switch operation. Information between the two sub-systems is interchanged by task transfers; a request task from CPS to DSS, followed by a response task in the reverse direction. Five different requests can be issued by CPS. These are:
i) A request to allocate a path between two terminations A and B.

ii) A request to change a reserved path between A and B.

iii) A request to clear a path between A and B.

iv) A request to clear all paths connected to A.

v) A request to trace paths connected to A.

Requests (iv) and (v) are only employed after a CPS rollback (OWEN 73) that is a CPS re-initialisation after a hardware or software fault. The current DMNSC design does not make use of request (ii) as the probability of blocking in DSS is thought to be extremely low. When a trace request task is issued by CPS, DSS responds with a task containing the identity of a termination to which A is connected plus an indication of whether more paths exist. The trace/response sequence is repeated until a response task indicates that no more connections to A exist. When setting up and clearing paths for a call, CPS specifies the identities of the two terminations in the task request.

CPS also employs DSS for the connection of tones such as busy, number unobtainable and recorded announcements. The request task from CPS to DSS contains the identity of the tone circuit of the appropriate type. DSS always connects the tone circuit in the backward direction that is to the calling party. This fact facilitates the connection of a tone circuit simultaneously to a number of speech circuits. The tone circuits are cleared using the usual 'Clear Path' request.

The 'Trace Path' message is issued by CPS to DSS in an attempt to establish calls in progress at the time of rollback together with the identity of other calls involved. This request is issued whenever any of the following messages are received by CPS after a rollback:
a message that starts a call such as 'IAM' and 'Seizure', a message that ends a call such as 'Release' and 'Circuit Free' or a message that changes the maintenance state of a circuit such as 'Maintenance Busy' and 'Free' from the Maintenance Control Sub-system. At the end of the first call which occurs on every circuit affected by the CPS rollback, a 'Clear All' request is sent to DSS instead of 'Clear Path' to ensure that all possible connections to the circuit are indeed cleared out before the circuit is returned to the free state.

A software sub-system with which CPS communicates is the Management Statistics Sub-system (MSS). CPS has the responsibility to provide this sub-system with the necessary information upon which traffic and related measurements can be based. The basic philosophy associated with the provision of call processing statistical information is that CPS should not keep any data of a statistical nature to itself, but rather provide this information in its own format to MSS. MSS then processes that information and incorporates it into the required measurements (DENT 78A,B,C).

The maintenance and update aspects of DMNSC are undertaken by the Maintenance Control Sub-system (MCS) in co-operation with CPS. A comprehensive range of resources such as circuits, routes and digit decodes are provided which can be updated by the maintenance staff using the online update interface with MCS through an interactive user-oriented language. Hardware and software faults generated by CPS are passed to MCS which pass them in the proper format to the maintenance staff.

Overload control in the DMNSC is the responsibility of the Overload Control Sub-system (OCS). This sub-system interrogates target processes
periodically for information such as the amount of work rejected and accepted in the last monitoring period and the current workload limit. The current workload limit for each process is set by OCS and represents the maximum number of simultaneous calls in the set-up phase. When this limit is exceeded, a process simply rejects any new calls by reducing its workload limit to zero and sending a 'Congestion' message if backward signalling allows or by applying a congestion tone via the digital switch. When the workload condition disappears, OCS instructs the process to raise its workload limit to a reasonable level which is roughly seven-eight of the workload for the process at the time of the overload (DENT 78A, B, C.)

7.2 THE DESIGN OF THE DMNSC SIMULATION MODEL

7.2.1 INTRODUCTION

The reasons for simulating the DMNSC, namely, to tune the CPS and assess the overall performance of the DMNSC are considered in greater detail in Section 7.1.1. In the following sections we will be concerned with the design of the DMNSC simulation model based on the message sequence chart of Figure (7.1). The square boxes in Figure (7.1) represent processing activities by the software sub-systems of the DMNSC as a result of a message or task arrival. The numbers in the boxes indicate the order in time in which messages concerning a call are processed. A message processing time is calculated by counting the number of instructions in the piece of code corresponding to a particular box and multiplying that by the average execution time per instruction. This information is supplied by the design team. A relevant term which has been coined recently in this respect is a 'mission' (BEAR 79).
A mission is a process instance run initiated by a message arrival (after a delay if no CPU is available). Before or at the end of a mission a process instance may send tasks or messages to other process instances.

The columns of boxes in Figure (7.1) represent the application software processes of interest in the DMNSC. These are the incoming SIS software, the incoming CPS software, the outgoing CPS software, the digital switching sub-system and the outgoing SIS software. The left and right boundaries of Figure (7.1) represent the hardware associated with the incoming and outgoing SIS software with which it communicates by passing hardware messages.

The message sequence chart of Figure (7.1) is transformed to Figure (7.2) which shows the components of the DMNSC simulation. The hardware and software entities of the DMNSC are translated in a one-to-one fashion to their corresponding model processes indicated by the big circles. The messages interchanged for each call through the DMNSC are indicated with their appropriate sequencing numbers. The Mark II BL system Model which controls the DMNSC model is transparent to the individual simulation processes as is the case with the real system.

The DMNSC model is built for an exchange of 1060 incoming and outgoing circuits and trial runs are made with a traffic of 0.6 Erlang per circuit. DSS and SIS are periodic processes activated once every 10 msec by INTIM. CPS is an aperiodic process activated by the arrival of any task of an appropriate priority (Section (4.3)).

The process index of DSS is 25 while the first instance of SIS process has a process index of 26-and the nth instance a process index of \( 26 + (n-1) \). Likewise, the first CPS instance has a process index of 49 and the nth instance a process index of \( 49 + (n-1) \).
Components of the DMNC model and signals interchanged per call

**Figure (7.2)**

**Key**
- **SIS** = Signalling Interworking Subsystem
- **CPS** = Call Processing Subsystem
- **DSS** = Digital Switching Subsystem
- **MSS** = Management Statistics Subsystem
- **IAM** = Initial Address Message
- **SAM** = Subsequent Address Message
- **FAM** = Final Address Message
- **SOC** = State of Call
As mentioned before in Chapters 4 and 5, a process index reflects its priority, that is the lower the index, the higher is the priority. It is worthwhile mentioning here that the term instance is used in the same sense as replication or replicate, the former being the jargon of the simulation analysts world-while the latter lies in the domain of SPC software engineering. The two terms will be used interchangeably. The strategy adopted when starting or stopping the time measurements of any one of the eight response times of interest of Figure (7.1) is as follows: the starting and stopping of a particular time measurement is always performed before sending a message. For example, the time measurement of the 'Outgoing Circuit Selection Time (Task Space to Task Space)' of Figure (7.1) is stopped before sending the 'IAM' message to the appropriate SIS process instance. This strategy is adopted because the delay in sending the message to its destination through the operating system may be appreciable enough to distort the time measurement.

The number of SIS and CPS replicates in a DMNSC depends on the size of the exchange and the number of circuits allocated per replicate. In the DMNSC, the circuits are allocated to the replicate in a staggered fashion as in Figure (7.3). Hence there are 2 SIS instances or replicates and 4 CPS instances in this particular model. The model is highly parameterised with parameters such as the total number of circuits in an exchange, the number of CPUs in the controlling multiprocessor system and parameters of various distributions are passed as input data to the model in the initialisation stage. We will now consider, in more detail, the design of the individual simulation processes in the DMNSC model.
FIGURE (7.3) : CIRCUIT ASSIGNMENT TO SIS AND CPS REPLICATIONS
7.2.2 The SIS Simulation (2337 - 2702)

Although SIS is depicted as two separate software processes that is incoming SIS and outgoing SIS in Figure (7.1), in reality it is one software sub-system and indeed it is a requirement of the software design engineers that the model reflects this fact. Figure (7.4) is the SIS simulation process model flow chart. The process task index table is designed as shown in Table (7.1). The table depicts the software processes with which SIS is allowed to communicate the incoming task indices and the tasks priorities.

Since the SIS process communicates with its hardware a message table is defined as in Table (7.2). The table has twenty entries, each entry representing the nature of the hardware signal, and the associated incoming and outgoing circuits identities.

A common interface specification is adopted for all the software processes in the model. This interface consists of the general registers contents of the CPU on which an instance is running. The contents of the general registers in the DMNSC simulation are defined as follows:

\[
\begin{align*}
G(0) & : \text{The outgoing task index} \\
G(1) & : \text{The outgoing message identity} \\
G(2) & : \text{not used} \\
G(3) & : \text{The incoming circuit identity} \\
G(4) & : \text{The outgoing circuit identity} \\
G(5)-G(7) & : \text{not used}
\end{align*}
\]

Each SIS instance has a table indicating the incoming messages identities corresponding to the numerical values of G(1) of the tasks received as in Table (7.3). From the above mentioned interface specification,
every task interchanged between the processes is self-contained.

SIS is a periodic process, activated periodically by an unblocking task of a high priority from INTIM every 10 msec. When it is selected to run, it issues a FETCH(15) call to the process allocator to pick up the first task in its input queue if any. If a task is found which is indicated by positive condition codes, SIS checks the value of G(1) to establish the identity of the received message. Using the value of G(1), it branches off to the appropriate code representing the response of SIS to that particular message. This response always incorporates a 'Hold' or a delay and possibly the sending of a hardware message to an incoming or outgoing SIS hardware process or the stopping of the time measurement of one of the response times of interest or both. This process of fetching a task from the input queue is repeated until the input queue is exhausted which is indicated by the condition codes being zero. The process then scans its message array to see if there are any hardware messages pending. If a hardware message is detected the process branches off, using the message number, to the appropriate code servicing that particular hardware message as in Figure (7.4). A typical signal or hardware message servicing will always start by resetting the message array subscript representing the message index so that an identical hardware message may be sent immediately afterwards. The process then holds or delays itself for a simulated time equal to that required to carry out the same servicing activity in the real system. Then the SIS model may send a task to the appropriate CPS instance to communicate the arrival of the hardware message. The hardware message in this case may be a seizure that is an arrival of a new telephone call, a pulsed-out digit, a subscriber answering or clearing down or a circuit freed.
The appropriate CPS instance is indicated to the operating system by writing the equivalent outgoing task index in register G(0). This is achieved by calling the global integer procedure CPSREP (CCTNUM), Line (231), to calculate the replication number of the CPS instance. CCTNUM is either the incoming circuit or outgoing circuit number as appropriate. The value of CCTNUM is obtained from the array MESSAGE and would have been written to the appropriate array subscript when the hardware message first arrives, as in Table (7.2). As the first CPS instance is indicated to the operating system by G(0) = 2, the appropriate CPS replication will thus be indicated to the operating system by the general register G(0) having the value:

$$G(0) = 2 + \text{CPSREP(CCTNUM)}.$$ 

As we mentioned before, the other information passed in this task is a numerical indication of the nature of the outgoing task to the called CPS replication in G(1), the incoming circuit identity in G(3) and the outgoing circuit identity, if known at that instance in time, in G(4). The hardware message servicing activity may also involve sending a hardware message to the incoming or outgoing SIS hardware as well as starting or stopping a response time measurement.

This process of checking for the arrival of hardware messages and the subsequent servicing is repeated until no more hardware messages are found. The process will have completed its servicing epoch of both software messages (tasks) and hardware messages. It will then call the process allocator to be blocked until periodically activated again 10 msec later by INTIM to go through the above mentioned routine. The hardware messages are usually exchanged between SIS instances and the incoming and outgoing SIS hardware processes. The software messages or tasks are exchanged between

276
SIS and CPS instances. An SIS instance receives a total of 10 software messages or tasks and 20 hardware messages for each telephone call.

7.2.3 The CPS Simulation (2706 - 3074)

As with SIS, although the CPS process is initially modelled as two separate processes to model the incoming and outgoing sides of the DMNSC, the models are later modified and combined in a single CPS model, on the request of the DMNSC software engineers. The CPS process model flow chart is depicted in Figure (7.5). The CPS simulation task index table is shown in Table (7.4) whereas the identities of the incoming tasks are shown in Table (7.5). A total of 23 tasks per telephone call is received by a CPS instance. The CPS process, as we have mentioned, is an aperiodic process that is, it services an incoming task as soon as possible depending on the instance priority, the task priority and the availability of a CFU. (Chapter 4).

When a CPS simulation instance, referred to subsequently as CPS instance, is activated and run as a result of an arrival of a task, it checks its CPU general register G(1) for the nature of the incoming task. It is interesting to note here that a CPS instance does not first execute a 'FETCH(15)' like an SIS instance. This is because while for SIS, the task contents in the CPU general registers are those of INTIM unblocking periodic task, the general registers contents when a CPS instance is activated are those of a proper task or software message from another software process model. The arrival of the task initiates a particular mission or servicing activity to which the CPS instance branches using the numerical value stored in G(1).
A servicing activity may be a simple or a compound one. A simple servicing activity is typically represented by the holding or delaying of a process instance for the equivalent of the actual activity time such as the servicing activity resulting from the arrival of an 'IAM' or 'SAML' task from SIS as in figure (7.5). A compound mission, on the other hand, may entail the sending of one or more tasks to other process instances, stopping the timing of a response time of interest and/or searching for particular incoming tasks using the SEEK(N) call to the process allocator such as in the mission initiated by the 'Start Sending' message arrival in Figure (7.5).

If an outgoing task from a CPS instance is destined for another SIS instance, the identity of the called instance has got to be established, that is its process index, and the proper outgoing task index has to be written in the general register \(G(\sigma)\). In other words an identical procedure followed by an SIS instance and described in 7.2.2 is followed. The global procedure used here to establish the replication number of the outgoing SIS instance is the integer procedure SIS(CCTNUM), Line (224). 'CCTNUM', passed as an actual parameter, is the incoming or outgoing circuit number as appropriate. As the SIS process instance of highest priority is equivalent to an outgoing task index of 3, the outgoing task index of any other SIS instance stored in \(G(1)\) will have the value

\[
G(1) = 3 + \text{SISREP(CCTNUM)}
\]

Certain CPS service activities are of particular interest. The service activity initiated by the arrival of a 'SAM 3' task from an SIS process instance, sends a part-file read request task to the storage allocator, (2801 - 2807) for route translation. The details of the task are as follows:
\[ G(0) = 1 \]
\[ G(1) = \pi \]
\[ G(2) = 8 \]
\[ G(3) - G(7) \text{ not used} \]

G(1) contains the identity or the process index of the CPS instance sending the request. This information is used by the storage allocator, later, to route back the response task to the appropriate CPS instance. The value of G(2) is an indication to the storage allocator of the service requested that is a part-file read. The storage allocator response task is usually processed fast by the CPS replicate because it is of the highest priority and hence is always inserted at the top of the input queue. The CPS instance responds to the storage allocator response task by sending the task 'Outgoing Route Seize' to the CPS instance in charge of the outgoing route (2809 - 2818). When this task is encountered in the input queue, the called CPS instance responds by selecting randomly a free outgoing circuit. It, also, creates a call record process instance for this outgoing circuit. A 'Start Sending' task is also sent to the CPS instance in charge of the incoming circuit communicating the identity of the selected free outgoing circuit.

Until the message 'Set Up Complete' is generated the CPS instance responsible for the incoming circuit has the control of the telephone call. From the 'Set Up Complete' message until the 'Call Released' message the CPS instance in charge of the outgoing circuit has the control of the call. After 'Call Released', the call as such has ended; all that remains is for the circuits to be fully released that is for the call record, the incoming SIS hardware and the outgoing SIS hardware instances, related to the released call, to terminate. These three process instances are described in section 7.2.5.
7.2.4 The DSS Simulation Model:

This sub-system is modelled by two simulation processes. DSSSW (3078-3161) and DSSHW (3165-3230) modelling the DSS handler and the DSS hardware respectively.

The DSSSW process is a periodic process activated once every 10 msec. Figure (7.6) depicts the process model flow chart and its task index table. When the process is activated periodically, it picks up any incoming tasks in the input queue by a FETCH(15) call to the operating system which is interpreted as a request to fetch any task of any priority.

The DSSSW process expects two types of tasks only and both from CPS process instances. The two tasks are 'Set Up Switch Path' and 'Clear Switch Path'. In each case the identities of the incoming and outgoing circuits in question are sent in G(3) and G(4) respectively, as usual.

If DSSSW detects the presence of an incoming task, it checks the value of G(1) to establish whether the request task is for setting up or clearing down a switch connection. A value of G(1) = 1 indicates the former, whereas G(1) = 2 indicates the latter. In both cases, the process holds for the appropriate activity time sends a hardware message to set up or clear the path, together with the identities of the incoming and outgoing circuits to DSSHW process and schedules DSSHW to run immediately after itself.

If no incoming task is detected in the input queue, DSSSW checks for any hardware messages from DSSHW as a response to previous messages from itself for setting or clearing switch connections. The hardware messages from DSSHW are indicated by DSSHW setting one or both of the indicators RESPONSE 1 and RESPONSE 2. If the presence of a hardware
message is detected, the appropriate indicator is reset so that identical hardware messages may be sent. DSSSW then holds for the activity time, stops the appropriate time measurement and sends a response task to the appropriate CPS instance. If the task to be sent is a response task to a 'Set Up Switch Pass' request, the identity of the CPS replication is derived from the identity of the incoming circuit as this is the process instance which sends the request in the first place. Similarly, the identity of the CPS replication to which the response task to a 'Clear Switch Path' is to be sent is derived from the identity of the outgoing circuit. Both the incoming and outgoing circuits identities will have been sent by DSSH along with the hardware message. The values will be written to INCCTS and OUTCCTS or INCCTC and OUTCCTC.

After servicing a hardware message from DSSH, a fresh look is taken at the process input queue to see if any request tasks have arrived in the meantime. After servicing any request task which may have arrived, in the manner described, a check is made on the presence of hardware response messages from DSSH. Any such request is also serviced in the manner described above. This sequence is repeated until all the software and hardware messages are serviced. The process then blocks awaiting the next periodic activation 10 msecs later by INTIM.

Figure (7.7) shows the flow chart of the DSSHW process (3165-3230) which simulates the digital switch. DSSHW is activated by DSSSW after sending a hardware request. The identities of the incoming and outgoing circuits are sent usually by DSSSW as part of the hardware message. When activated, DSSHW first checks for a 'Set Up Path' request. If it finds one, it first resets this
DSS H/W MODEL
FIGURE (7.7)

SET UP PATH?

N

CLEARPATH?

ERROR(20)

RESET FLAG

HOLD (SETUPTIME) MICROSECONDS

SEND IDENTITIES OF I/C & O/C CCTS TO SET & RESPONSE1 TO DSS S/W

PASSIVATE

Y

Y

RESET FLAG

HOLD (CLEAR TIME) MICROSECONDS

SEND IDENTITIES OF I/C & O/C CCTS TO CLEAR & RESPONSE2 TO DSS S/W

PASSIVATE

END RESPONSE TIMES

Y

284
hardware message indicator, which is a boolean variable. It then holds for the set up time, derived from a uniform distribution of maximum value of 10 msec, the maximum time required by the digital switch to switch through a path between two terminations. Next it sends the identities of the terminations connected to DSSSW process and the appropriate hardware message, that is setting RESPONSE 1. It also, stops the timing measurement of the 'Switching through time (Hardware)' before passivating.

A similar action is taken if the hardware request message is to clear an existing connection. In this case the hardware response message to the DSSSW process will be indicated by setting RESPONSE 2. If when DSSHW is activated it finds no hardware message request pending, it outputs an error message.

7.2.5 Telephone Calls Generation Simulation

The strategy adopted in the generation of telephone calls for the DMNSC model aims at the maximum flexibility in the generation process and call progression through the exchange. For each call generated, three processes instances are created and exist in the model throughout the lifetime of a call. These are the call record process instance, CALLRECORD(CIRUITNUM) (2029-2114), the call incoming SIS hardware, INCSISHW(CCTNUM) (2166-2333), and the call outgoing SIS hardware, OUTGSISHW(CCTNUM) (3234-3379). These three processes instances interact and co-operate among themselves to produce the desired realistic modelling of a call progression through the DMNSC. The following is a detailed explanation of this.

The call generation is controlled centrally by a single call generator process instance, CALLGENERATOR (2118-2162), depicted in Figure (7.8).
SELECT RANDOMLY A FREE INCOMING CIRCUIT

GENERATE A CALL RECORD & I/C SIS H/W INSTANCES FOR THE SELECTED CCT

ACTIVATE CALL RECORD AFTER THIS INSTANCE

CALL GENERATOR MODEL FLOWCHART

FIGURE (7.8)
The call generator creates instances of CALLRECORD and INCSISHW processes at exponentially distributed time intervals to represent the arrival of telephone calls. The inter-arrival time is determined by the telephone traffic offered to the exchange in erlangs, \( A \), say. Suppose that

\[
\begin{align*}
C &= \text{the number of calls originating during a long period } T \\
\tau &= \text{the average duration of a call.} \\
A &= \text{The average number of simultaneous calls in progress during the period } T \text{ that is}
\end{align*}
\]

\[
A = \text{'traffic flow' or 'traffic intensity' in erlangs,}
\]

Now, the volume of traffic may be defined simultaneously as:

\[
\begin{align*}
\text{Volume of Traffic} &= C \times \tau \\
&= A \times T \\
\therefore A \times T &= C \times \tau \\
\text{or } A &= \frac{C \times \tau}{T}
\end{align*}
\]

Since \( C \) calls originate during the period \( T \), and the average holding time as a fraction of \( T \) is \( \bar{\tau} \), then the average number of calls which originate during \( \tau = C \bar{\tau} = A = \text{traffic intensity} \) in erlangs.

Taking a traffic intensity of 600 erlangs, an absolute value of \( \tau \) of 3 minutes and \( T \) as being 1 hour then

\[
600 = \frac{C \times 3}{60}
\]

\[
\therefore C = \text{calls generated in 1 hour} = 12,000
\]

\[
\therefore \text{Average call inter-arrival time} = \frac{3600}{12000} = 0.3 \text{ sec.}
\]

Since the traffic is considered as originating from an infinite number of sources, that is a Poisson call arrival, the inter-arrival distribution is sampled from a negative exponential distribution with the mean

\[
\frac{1}{3.33 \times 10^{-6} \mu \text{sec}^{-1}}.
\]
When a telephone call is due to be generated, CALLGENERATOR will select, randomly, a free incoming circuit. An incoming circuit is free if no CALLRECORD process instance with the identity of the circuit exists in the model. When a free circuit is selected, CALLGENERATOR creates a CALLRECORD instance and an INCSISHW instance for the call, passing the circuit identity selected to both instances as the call identity. It also, inter links the two instances together using reference variables local to each process instance.

The call generator generates a few calls at the start of the simulation run in order to reduce the transient period. Other measures to reduce this transient period, or 'warm-up' period, are discussed in connection with the CALLRECORD process. The call generator then holds for an inter arrival time sampled from a negative exponential distribution whose mean is determined by the traffic value before going through the above described process of a call generation.

When a CALLRECORD process instance is created, (Figure (7.9)) it is scheduled to be active after the CALLGENERATOR. To reduce further the transient period, the concept of the 'state of Call' is introduced in CALLRECORD (2029-2114). A newly generated call may be in one of three states marked as A, B and C in Figure (7.1). The particular state is randomly selected by the CALLRECORD process instance as its first action when activated by the CALLGENERATOR process. The concept of the state of call is introduced to reduce the length of the transient period in the simulator. This is very useful in view of the fact that if all calls injected into the system are generated starting with a 'Seizure' hardware message arrival, that is at state A, then a minimum of three minutes of simulated time is required to obtain a reasonable mix of calls to justify the
CALL RECORD MODEL

FIGURE (7.9)

CALL RECORD

SAMPLE SOC & PASS VALUE TO 1/C SIS H/W

SOC=?

1=a

SEIZURE TO 1/C SIS H/W

2=b

generate randomly an 0/G SIS H/W & link it to this call record & its 1/C SIS H/W

3=c

'subs. clears down' to 1/C SIS H/W instance

SOC=?

2

'SUBS. ANSWERS' to 0/G SIS H/W

3

activate 0/G SIS H/W instance after this call record

THIS CALLRECORD IS RE-ACTIVATED BY IT'S INCISIW WHEN IT RECEIVES 'CCT FREE' FROM SIS S/W

'activates 1/C SIS instance after this call record

PASSIVATE

reset references to this call record & 1/C & 0/G SIS H/W

TERMINATE

THIS CALLRECORD IS RE-ACTIVATED BY INCISIW WHEN CANCELLING T/O B
collection of statistical data. The reason for this is that the post dialling delay takes six seconds on average and the speech phase three minutes on average, which results in a longer run because of the fine time grain of the Mark II BL System simulator and the fact that it is not doing much useful work, apart from housekeeping, while holding for those respective times. By generating the individual calls to start at A, B or C that is with a 'Seizure' or 'Subscriber Answers' or 'Subscriber Clears Down' event, the post dialling delay and the speech phase are eliminated. To conserve the flow of traffic through the system, a call generated at state A has got to terminate before the start of state B. Likewise, a call generated at state B has got to terminate before the start of state C. The simulation processes modelling the telephone calls that is CALLRECORD, INCSISHW and OUTGSISHW have been designed and synchronized to ensure this. The net result of this strategy of call generation is that we are able to get a relatively fast simulation for the DMNSC with simulated time to real time ratio of about 7 to 1 on average.

When activated, a CALLRECORD instance determines randomly the state of its call and passes that information to its INCSISHW instance. If the state of call sampled is state A, the CALLRECORD instance will indicate a 'Seizure' to the calling subscriber line circuit that is the call INCSISHW instance. It will then schedule the INCSISHW instance and passivates. On the other hand, if the state of call generated is state B or C, then judging by Figure (7.1), an OUTGSISHW process instance must be created for the call to proceed. The CALLRECORD instance selects randomly a free outgoing circuit number, creates an instance of OUTGSISHW with that circuit identity and links it both ways to its INCSISHW and CALLRECORD instances. If the state of call generated is state B the CALLRECORD instance
will indicate to the OUTGSISHW that the called subscriber has answered before scheduling the OUTGSISHW instance to run after this CALLRECORD instance when it passivates. If the state of call is that of state C, the the INCSISHW instance is notified that the calling subscriber has cleared down before scheduling it to run when this CALLRECORD passivates. For a type A call this CALLRECORD instance is re-scheduled by the OUTGSISHW instance when resetting the timeout after the last digit. If it is of type B or C then the CALLRECORD instance will be re-scheduled by the INCSISHW instance when cancelling the timeout after the arrival of 'Answer' or 'Circuit Free' message from SISSW instance respectively as shown in Figure (7.1). When re-activated, the CALLRECORD instance resets the references to itself, the INCSISHW and OUTGSISHW instances to NONE and terminates.

The INCSISHW process flow chart is shown in Figure (7.10) and the listing on (2166-2333). For each call generated an instance of this process is created at the time of creation of a CALLRECORD instance by the CALLGENERATOR process. This process instance is first activated by its CALLRECORD process instance if the state of call is A or C and by the respective SIS instance for a type B call.

If it is a state A, then a 'Seizure' message will have been sent to this instance by its CALLRECORD. The first action for INCSISHW instance is to check for the presence of this message. If it is not there, an error message is printed and the simulation stopped. This is a serious error which will throw the model out of synchronism and distort the measurements if the run is allowed to continue. Assuming that the 'Seizure' message is found, it will be passed together with the incoming circuit identity to the SIS replication. INCSISHW instance then alternately holds for an interdigital pause then sends
I/C SIS H/W MODEL

FIGURE (7.10)

CALL ARRIVAL?

ERROR(21)

ABORT RUN

'SEIZURE' & CI TO SIS INSTANCE

HOLD(IDP) mSEC

1ST DIGIT & CI TO SIS INSTANCE

HOLD(IDP) mSEC

2ND DIGIT & CI TO SIS INSTANCE

HOLD(IDP) mSEC

3RD DIGIT & CI TO SIS INSTANCE

HOLD(IDP) mSEC

4TH DIGIT & CI TO SIS INSTANCE

HOLD(IDP) mSEC

5TH DIGIT & CI TO SIS INSTANCE

HOLD(IDP) mSEC

6TH DIGIT & CI TO SIS INSTANCE

HOLD(IDP) mSEC

7TH DIGIT & CI TO SIS INSTANCE

HOLD(IDP) mSEC

8TH DIGIT & CI TO SIS INSTANCE

PASSIVATE

RESET REFERENCES TO CALL RECORD & THIS INSTANCE

TERM INATE

ANSWER?

ERROR(22)

PASSIVATE

PRES S

HOLD (TIMEOUT) mSEC

'TIMEOUT' & CI TO SIS INSTANCE

ACTIVATE CALL RECORD AFTER THIS INSTANCE

ERROR(23)

'SILENT' & CI TO SIS INSTANCE

START TIMING RESPONSE & RESPONSE

PASSIVATE

CT FREE?

ERROR(24)

KEY
CI=CALL INDEX
IDP=INTER DIGITAL PAUSE
a digit and the circuit identity to SIS replication until the eighth
digit is sent after which the instance passivates. When reactivated
later by its CPS replication on receiving the set-up switch response
from DSS, it resets its reference variables to NONE and terminates.

For a state B call, INCSISHW instance first checks for the presence
of an 'Answer' message from the SIS replication and prints an error
message if the message is not found. It resets a covering time
out started by its SIS replication after a delay sampled from a
uniform distribution. It schedules its CALLRECORD instance, resets
the references to NONE and terminates. When the CALLRECORD
instance is activated, it resets all its references to NONE and
terminates. Special care has been taken in the DMNSC model to
reset the reference variables to transient processes to NONE when
they terminate to reduce the time used for garbage collection in
SIMULA and consequently increase the run efficiency of the simulator.

For a state C call, INCSISHW instance checks for the presence
of a 'Subscriber Clears Down' hardware message from its CALLRECORD
instance and outputs an error message if not present. It sends a
hardware message 'Idle' and the call identity to the SIS replication,
starts 'Release Time' and 'Clear Forward Transfer Time' time
measurements and passivates. When reactivated by the SIS replication
it checks for the presence of a 'Circuit Free' message, schedules
its CALLRECORD, sets the reference variables to NONE and terminates
marking the end of its telephone call.

A third process, that together with CALLRECORD and INCSISHW complete
the modelling of a telephone call is the OUTGSISHW process (3234-3379).
It models the call scenario at the outgoing end of the exchange by
modelling the hardware messages exchanged with SIS replications and
indicating the states through which a call passes at the outgoing end
as shown in Figure (7.11).

If the state of call generated is A, that is a call starting at the beginning of its life scenario, then the responsibility of the selection of a free outgoing circuit is vested on the appropriate CPS replication process. This selection is made sometime later after the call generation. When a free outgoing circuit is selected, an OUTGSISHW process instance is created with the outgoing circuit identity passed as an actual parameter. It is linked both ways to its corresponding CALLRECORD and INCSISHW instances.

On the other hand, if the state of call sampled is B or C then the OUTGSISHW process instance is created by the CALLRECORD instance and inter-linked at the time of sampling of the state of call.

Whenever an OUTGSISHW instance is activated, it always checks for the arrival of a hardware message as in Figure (7.11) and if no such message is pending it outputs an error message and terminates, otherwise, if a message is found it is serviced and then the process passivates. The process instance is always activated by its SIS replication or its CALLRECORD instance after sending a hardware message. For the messages sent by the SIS replication, a covering time-out is started in SIS and the OUTGSISHW has to send back a timeout reset signal within the specified timeout limits. The reset signals are sent after a delay sampled from uniform distributions (TIMEOUT1 to TIMEOUT7). After resetting the covering timeout for the 6th digit, OUTGSISHW starts the timing measurement of the 'Switch Through Time (Hardware)' response time, schedules its CALLRECORD, resets its reference to NONE and terminates.

If the call is generated to start at label B in Figure (7.1) that is state of call B, then after servicing the 'Subscriber Answer' message
from its CALLRECORD, OUTGSISHW immediately resets its reference
variables to NONE and terminates.

For calls with state of call C, the OUTGSISHW instance services
the 'Idle' message by sending back 'Busy' and 'Free' messages to
its SIS instance, resets its references to NONE and terminates.

7.2.6 Quantification of the Delays in the DMNSC

To validate the design of the DMNSC it is necessary to quantify
particular delays between certain instances in the call processing
sequence in the DMNSC. Such information cannot be obtained from
the DMNSC prototype and the only feasible way is through simulation.

With reference to Figure (7.1), the encircled numbers indicate
the start and finish of response times in which there is particular
interest, as follows:

1) Outgoing circuit selection time (hardware to hardware)
2) " " " (task space to task space)
3) Signal transfer time (hardware to hardware)
4) " " " (task space to task space)
5) Switch through time (hardware)
6) Switch through time (software)
7) Release time
8) Clear forward transfer time.

The DMNSC configuration tested consists of 1061 circuits, 2 CPUs,
2 instances of SIS, 4 instances of CPS, 1 instance of DSS and a
traffic of 0.6 erlang per circuit. For each quantity of interest
(1) to (8) the average value, the standard deviation, the minimum
and the maximum are calculated. These values are tabulated in Table
(7.6).
<table>
<thead>
<tr>
<th>Measure</th>
<th>Average Value (μs)</th>
<th>Standard Deviation (μs)</th>
<th>Minimum Value (μs)</th>
<th>Maximum Value (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>O/G CCT. Selection Time (H/W to H/W)</td>
<td>37120.0</td>
<td>2642.2</td>
<td>34176.1</td>
<td>40591.0</td>
</tr>
<tr>
<td>O/G CCT. Selection Time (Task space to Task space)</td>
<td>24925.7</td>
<td>65.6</td>
<td>24868.0</td>
<td>25024.0</td>
</tr>
<tr>
<td>Signal Transfer Time (H/W to H/W)</td>
<td>16055.5</td>
<td>2630.0</td>
<td>13091.2</td>
<td>19478.0</td>
</tr>
<tr>
<td>Signal Transfer Time (Task space to Task space)</td>
<td>6114.8</td>
<td>10.3</td>
<td>6100.0</td>
<td>6121.0</td>
</tr>
<tr>
<td>Switch through time (H/W)</td>
<td>25088.0</td>
<td>2093.2</td>
<td>23602.0</td>
<td>27482.0</td>
</tr>
<tr>
<td>Switch through time (S/W)</td>
<td>27720.3</td>
<td>1162.3</td>
<td>25735.0</td>
<td>29002.0</td>
</tr>
<tr>
<td>Release Time</td>
<td>33527.5</td>
<td>11928.8</td>
<td>21619.3</td>
<td>46519.0</td>
</tr>
<tr>
<td>Clear forward Transfer Time</td>
<td>18451.6</td>
<td>7770.1</td>
<td>11749.0</td>
<td>27349.0</td>
</tr>
</tbody>
</table>

TABLE (7.6.): QUANTIFICATION OF THE DELAYS OF INTEREST IN THE DMNSC
The values are found to be within the DMNSC specifications. Experiments for larger sizes of the DMNSC handling up to 20,000 erlangs of traffic may be conducted if required to validate the DMNSC design. As a by-product of this experiment, the lockouts occupancies in the DMNSC, namely, FREELO, SUSPLO and INTLO are examined. As expected they are found to be extremely low ranging from 1.36% to 2.47%.
8.1 ACHIEVEMENTS

In this thesis we have developed a philosophy for simulating stored program control switching systems and have presented a methodology for a computer-aided design of such systems through simulation. The multi-level process simulation proposed and presented eliminates the limitations of the real-time environment simulation and flat-level simulation methodologies used at present. A real-time environment simulator is costly to develop and is limited in its application to the testing and debugging of the application software. There is no means of evaluating the performance of the real-time operating system and/or proposed future modifications and extensions. On the other hand, flat-level software simulators are monolithic in nature, difficult to verify and validate and costly to run. The resulting lack of credibility manifests itself in the suspicion with which the simulator results are treated.

We have developed an alternative simulation methodology of SPC systems where the complex SPC system is divided into its natural sub-systems. The hardware and software sub-systems in a system are transformed in a 1 to 1 transformation process to their corresponding models. The sub-system models are placed at different levels in a hierarchy according to the role they play in the real system. For example, the model of the real-time operating system upon which the other sub-systems depend for their functioning is placed at the bottom level of the simulator, with the applications models one level higher. The adoption of the philosophy of a multi-level process-based simulation enabled us to develop a simulator in which the interfaces and driving
tables of the real system are preserved. The resulting simulator looks very similar to the high-level structure of the real system. Moreover, the detailed trace incorporated in the simulator traces the interaction of the sub-systems models in a manner readily recognizable to the software design engineers. These factors helped to sell the simulator to the community of users of software design engineers.

The importance of verification and validation in the simulation process has been stressed in Chapter 6. This aspect of model building has sometimes been ignored by some simulation analysts with a consequent lack of credibility and confidence in the simulator. In contrast to the flat-level simulator of System X developed by the British Telecommunications (formerly the British Post Office), our simulator has been thoroughly verified and validated. We feel that, for a complex and highly-interactive system such as the Mark II BL multi-processor system, the verification of the logic of individual modules in the simulator as well as the sequences of interactions between the modules is extremely important. Indeed, getting the quasi-parallel serial execution of SIMULA to reproduce faithfully the parallel interactions of the process allocators, the interrupt triplicates and the individual processes running on different CPUs proved to be the most time-consuming phase in the development of the simulator.

For example, on one occasion it was noted from the trace that the local sequence control of a process changes its position and skips a statement when a process is re-activated after being interrupted and pre-empted. No obvious reason was apparent and the proper sequence was obtained by a re-arrangement of the sequencing statements in the process allocator and the interrupt triplicates. On another occasion during the validation phase, the storage allocator was not activated, because when it is loaded and had a higher priority, the CPU in which it was running was still...
referred to as LPA by INTRIP (Chapter 6) which was waiting in a queue for its process allocator to finish executing its present instruction. LPA references the process allocator on LCPU, the CPU running the lowest priority process. The process allocator on which the storage allocator is to run, senses that it is LPA and that INTIM is waiting for it. Therefore, instead of activating the storage allocator, it activates INTIM. When activated, INTIM comes out of its waiting queue and updates LPA as

\[ \text{LPA} := \text{LCPU.MYPA} \]

and hence points to a different CPU. INTRIP will then instruct LPA to re-schedule and the storage allocator is never activated again.

To cope with such situations additional pre-cautionary logic is incorporated in INTRIP thus whenever INTRIP is activated to service an interrupt, it checks whether LPA is changed since it was last activated and whether the suspended queue interrupt servicing is still required. If not, as in this case, then the interrupt is not serviced and the appropriate process such as the storage allocator here is re-activated. Other similar situations are corrected by taking appropriate actions until the simulator is able to cope with expected and unexpected combinations of events. The effort spent on the verification and validation proved to be worthwhile, judging by the validation experiment results.

The hierarchical multi-level process simulator proved to be capable of meeting its objectives which were; firstly, to provide a tool for the designers of the real-time operating system of the Mark II BL multiprocessor system which could be used for the performance evaluation of the prototype design and the evaluation of possible modifications and extensions; Secondly, to provide a tool for the performance evaluations of the members of System X family of exchanges. The ability of the simulator in
this respect is demonstrated by the simulation of the Digital Main Network Switching Centre (DMNSC). Valuable information to the processor utility and call processing sub-systems designers was obtained using the simulator as described in Chapter 7. Such information is not possible to obtain in this detail otherwise.

By adopting our philosophy of hierarchical multi-level process simulation for SPC systems, we were able to develop a simulator package which is open-ended, flexible in its level of detail of individual modules, adaptable, relatively fast and smaller in size than the corresponding flat-level simulator developed for System X before. Our simulator is about 3.5K statements and runs at a speed of 7:1 to 10:1 relative to real-time, depending on the configuration being modelled and the traffic intensity.

Aside from the original objectives and owing to the credibility demonstrated by the simulator, it is now being used to guide the analytical modelling process of System X family of exchanges. The simulator is being used to test the simplifying assumptions, such as the assumption of nodal independence, necessary for the approximate analytical modelling of System X. It will also be used for validating the analytical model when fully developed. Another use of the simulator is the evaluation of the proposed processor sub-system architecture for System X Release 2. To reduce the store contention in a tightly-coupled system such as the Mark II BL multiprocessor system, it is now proposed that clusters of a few CPUs each will be more appropriate. The simulator will be adapted to evaluate the performance of clusters of CPUs relative to the existing Mark II BL multiprocessor system architecture.

The simulator developed fits nicely with the existing emulator, which emulates a single CPU, to form a powerful integrated computer-aided design.
package for SPC systems design. The emulator is used for the verification of applications software and the simulator for the performance evaluation of SPC exchanges.

8.2 SUGGESTIONS FOR FURTHER WORK

Design modifications to the Overload Control Sub-system (OCS) have been proposed recently in an attempt to optimize its performance (GREE 80). To evaluate these modifications a fairly detailed simulation of the Overload Control Sub-system has been requested. The model of the overload control sub-system will be incorporated in the DMNSC model. It will then be possible to optimize the performance of the Overload Control Sub-system through conducting experiments on the DMNSC model.

The problems of determining the length of a transient period in a run and the sample size to achieve a specific confidence level in the results have been discussed by many simulation analysts before (FISH 67, KLEI 74, MIHR 72). These problems arise when the data is correlated such as in queuing systems. An original approach, the regenerative approach, has been proposed recently (CHAN 78, IGLE 78). This approach is based on the philosophy that many stochastic systems have the property of 'starting afresh' probabilistically from time to time. This enables the observation of independent and identically distributed blocks of data in the course of the simulation.

It will be nicer to incorporate the regenerative process feature in the simulator such that a PROCESS CLASS REGENERATIVE will determine the run length according to the required confidence level passed as a parameter at run time.
Interactive simulation packages are flexible and more user-oriented. Features such as the ability to stop the simulator during a run, interrogate and/or change the variable values and assemble a configuration to be run by a questionnaire type of conversation between the user and the package at the start of a session will make the simulator more attractive.
APPENDIX A

MK II BL MULTIPROCESSOR SYSTEM SIMULATOR IN CSL
**TABLE (A-1): PERIODIC PROCESSES TASK TABLE (PTM)**

<table>
<thead>
<tr>
<th>PROCESS INDEX</th>
<th>INCOMING TASK INDEX</th>
<th>TASK INDEX</th>
<th>PRIORITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**APPENDIX (A-1): COMMUNICATION BETWEEN PERIODIC APPLICATION PROCESSES**

**TABLE (A-2): PERIODIC PROCESS TABLES**

**TABLE A: PROCESS 2**

<table>
<thead>
<tr>
<th>PROCESS TI</th>
<th>CALLED PI</th>
<th>CALLED PROCESS TI</th>
<th>TASK INDEX</th>
<th>PRIORITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE B: PROCESS 3**

<table>
<thead>
<tr>
<th>PROCESS TI</th>
<th>CALLED PI</th>
<th>CALLED PROCESS TI</th>
<th>TASK INDEX</th>
<th>PRIORITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>0</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>0</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE C: PROCESS 6**

<table>
<thead>
<tr>
<th>PROCESS TI</th>
<th>CALLED PI</th>
<th>CALLED PROCESS TI</th>
<th>TASK INDEX</th>
<th>PRIORITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE D: PROCESS 8**

<table>
<thead>
<tr>
<th>PROCESS TI</th>
<th>CALLED PI</th>
<th>CALLED PROCESS TI</th>
<th>TASK INDEX</th>
<th>PRIORITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
LISTING 1
SOURCE(CHR, S1)
OBJECT(ED, COMMENT FILEA, ABCD)

CLASS MASTER
   CLASS LOCKOUT, 3(2)
   CLASS TIME PROCESS,11(10) SET SUSPENDED, INQUEUE, 11, RUNNING, BLOCKED
   CLASS TIME CPU,1(3) SET BUSY, FREE
   CLASS TASK, 200(5) SET FRATSKLIST
   FLAT OCCUPANCY, SIM
   ARRAY PTBM(4,10), PPTSK(10,3), PROC(11,2)
   DTBM IS THE PERIODIC TASK BIT MAP
   DPTSK IS PERIODIC PROCESS TASK TABLE, STORES FOR EACH ACTIVATED PROCESS, I/C TASK INDEX AND ITS PRIORITY
   A,B,C,D STORERS I/C.TIX, CALLED PRI, CALLED PROCESS TIX R TASK PRIORITY
   FOR EACH CORRESPONDING PROCESS
   HIST SUSPAINED, 11(20, 40, 60)
   ARRAY PROC(1,J) STORES TIMES INITIATED AND EXECUTION TIMES/PROCESS
   COMMON/AA/Lockout/BR/PROCESS, SUSPENDED, RUNNING, BLOCKED
   COMMON/CC/CPU, BUSY, FREE
   COMMON/DD/TASK, FRATSKLIST
   COMMON/DD/PROC
   COMMON/FF/AR, J, Y, INTRIP, SUSPOINT, N, K, M, PA, TX / GG/SUSPWAIT
   COMMON/HH/T, PROCESS, T, CPU, T, TIME, T, CLINT
   COMMON/HH/IVQUE
   SUSPWAIT HISTOGRAM MEASURES THE TIME FROM A PROCESS BEING SUSPENDED UNTIL IT IS SELECTED TO RUN AGAIN

   INITIALIZATION
   ==
   ==
   ==
   ==
   Z IS THE N, OF CPUS IN SYSTEM
   CYCLE=0
   INTRIP=<)
   LOAD FREE, FRATSKLIST, BLOCKED
   FOR I = FRATSKLIST
      FOR L = 1, 5
         TASK, I(L)=)
   DUMMY
   FOR I = BLOCKED
      FOR L = 1, 7
         PROCESS, I(L)=0
   DUMMY
   FOR I = FREE
      T, CPU, I=-1
   FOR L = 1, 3
      CPU, I(L)=0
   DUMMY
   FOR W = BLOCKED
      PROCESS, I(A)=4
   DUMMY

   DATA PTBM/0, 1, 7, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 4, 0, 1, 0, 1, 0, 1, 2, 0, 1, 2, 0, 1, 0,
   
   DATA PPTSK/1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 6, 10, 7, 4, 9, 8, 7, 6, 4, 5/
   ORLEV=0

   COMMON LEVELS INTERRUPT
C EXTERNAL TASK LEVELS INT.
SITIMIME=44000
SIM=44000.0
C STORES SIMULATED TIME IN SECS
INT=0
CIX=0
POINTER=1
WRITE(2,6)
6 FORMAT(1H1,120X,38HSIMULATION OF MK 28L PROCESS ALLOCATOR///H0,10X,3
/RAN==:=====================================/1)
WRITE(2,7)SIMTIME,Z
7 FORMAT(1H0,16HSIMULATED TIME =,18,2X,12H(MICRO-SECS)//1H0,13HNO,
/OFPUPUS =1,4,2X///1H0,51HLANGUAGE USED,CONTROL AND SIMULATION LANG
/UAGE:(CSL)///)
C TRIV=0
C FLAG TO INDICATE STATE OF INT. TRIPLECTES
T,TIME=SITIME
C SIMULATED TIME
T,CLINT=10000
C FIRST CLOCK INTERRUPT AFTER 10 MILSECS
CIX=0
OCCUPANCY=9.00
C COUNTS INSUCCESSFUL ATTEMPTS TO FIND A FREE CPU
WRITE(2,4010)
4000 FORMAT(1H0,3H)K1)
FOR S =1,4
FOR I =1,10
CHECK PTHSI(S,I)
PRRSET(2) FREE,FRTSKLIST,BLOCKED
WRITE(2,5070)
5000 FORMAT(1H0,3H)K2)
C ACTIVITIES
C BEGIN CLOCK INTERRUPT
C >>>>>>>>>>><<<<<<<<<<<
T,TIME GT 0
T,CLINT EQ 0
FOR I =1,11
PROCESS.I(4)=0
PROCESS.I(7)=0
PROCESS.I(8)=4
PROCESS.I(10)=0
endif
C RESET ATTRIBUTES FOR NEXT CLOCK INT
CIX=1
INT+1
T,CLINT=10000
PA+1
C BEGIN CLOCK INT. SERVED
C >>>>>>>>>>><<<<<<<<<<<
CIX EQ 1 614=613
613 CYCLE EQ 1 614,
614 T,TIME GT 0
CYCLE EQ 1 33615
615 ORLEV=1
ORLEV=1
C SET ORDINARY LEVEL INTERRUPT
WRITE(2,6070)
4000 FORMAT(1H0,3H)K3)
CIX=0
3 FREE EMPTY 92
126 FIND I BUSY MIN(CPU,I(1))
127 INTRIP=1
128 GO TO 15
129 C 2 FIND Y FREE FIRST
130 CPU,Y FROM FREE
131 CPU,Y INTO BUSY
132 RAR=2
133 C TO SHOW THAT AFFREE CPU IS SEIZED
134 GO TO 13
15 VCCPU+1
136 SUSPINT EQ 1
137 X=CPU,INTRIP(1)
138 C SELECT CPU STORED IN INTRIP,CURP OF LPCU IN CPU,X(1)
139 PROCESS,X(3) EQ 2
140 C SET CURP OF LPCU SUSP(INT)
141 PROCESS,X(10) = 1
142 C TO INDICATE LATOR THAT PROCESS WAS SUSP(INT)
143 RAR = 3
144 C TO SHOW THAT A BUSY CPU IS INTERRUPTED
145 LOCKOUT,2(1)+1
146 C ENGAGE SUSP(INT)
147 LOCKOUT,2(2)+1
148 C STORE TIME SUSPLO WAS USED
149 PROCESS,X FROM RUNNING
150 PROCESS,X INTO SUSPENDED
151 C TRIP = 1
152 C PRSET(2) SUSPENDED
153 C ENGAGE INT, TRIPLECT L,3.
154 UPLEV EQ 1 55
155 C TEST IF ORDINARY LEVELS ARE TRIGERRED
156 FOR I=1,10
157 PTM(POLINTER,1) EQ 1 B30
158 C CHECK IF THE PERIODIC PROCESS IS TO BE ACTIVATED
159 LOCK:IT.1(1)+1
160 LOCK:IT.1(2)+12
161 C FIND L FRSKLIST FIRST R30
162 FOR X=1E,13
163 C TASK.L(X) = PTASK(I/X)
164 C COPY TASK DETAILS OF ACTIVATED PROCESS FROM PTASK
165 TASK.X FRO1 FRASKLIST
166 TASK.X INTO IMQUEUE,11
167 C INSERT TASK.J INTIM WIQUEUE
168 DUMMY 3,3
169 FOR I =1,13
170 C CHECK PTM((POLINTER,1)
171 C PRSET(?1) IMQUEUE,11,FRASKLIST
172 POLINTER EQ 4 H35
173 POLINTER = 1
174 GO TO 40
35 POLINTER+1
175 C SET POLINTER FOR NEXT CLOCK INTERRUPT
176 40 FIND J FRASKLIST FIRST R1900
177 LOCKOUT,1(1)+1
178 LOCKOUT,1(2)+12
179 TASK,J(1) = 11
180 TASK,J(1) = 0
181 C THIS IS THE TASK FOR INTIM WITH HIGHEST PRIORITY
182 GO TO 135
183 10 WRITE(2,50)
184 50 FORMAT(1X,14A1 NO FREE TASKS)
185 C EXIT
186 133 DUMMY =11
187 C TASK HANDLED TO INTIM
188 TASK,J FROM FRASKLIST
FOR L = 1, 11
CHECK PROCESS.L(8)
CALL EHNTSK
PRISET(2) SUSPENDED
CALL HAND TASK AND SET STATE SUBROUTINE
ORLEV=0
CALL ECPUSCHED
SUSPINT = 0
CALL ESYSCF
IF(PROCESS.W(1)) 130, 130, 135
C PREVIOUS STATE SUSP(UNAL) OR SUSP(INT)?
133 LOCKOUT.3(1)+1
LOCKOUT.3(2)+2
FINJ I INQUEUE.I MIN(TASK.1(3)) = 135
FOR L=1,5
PROCESS.W(L)=TASK.1(1)
COPY TASK DETAILS INTO P.O.
PROCESS.W(L)=TASK.1(1)=0
CLEAR TASK BLOCK
LOCKOUT.1(1)+1
LOCKOUT.1(2)+2
TASK.I FRU1 TINQUEUE.N
TASK.I INT=1 PRIKTAIL
PROCESS.W(7)=1
C SET CONDITION CODES POSITIVE
SUSPENDED EMPTY 95
BEGIN INITIATING
C >>>>>>>>>>>>>>>>>>>>
C PRISET(2) SUSPENDED, INQUEUE.11
PROCESS.11(4) = 1
CYCLE=0
W=11
PROCESS.W(6)=4
C RECORD TIME SPENT IN MAIN PROGRAM
PROC.11(1)+1
PROC.11(2)+2
LOCKOUT.3(1)+1
LOCKOUT.3(2)+2
FINJ J INQUEUE.11 MIN(TASK.J(3))
TASK.J FRU1 TINQUEUE.11
CALL EHNTSK
INQUEUE.11 EMPTY =154
C ANY MORE TASKS? IF SO CALL EHNTSK
W=15
C SET PARAMETERS FOR EBLOCK(P)
CALL EBLOCK(P)
PROC.11(2)+PROCCESS.W(6)
PRISET(2) SUSPENDED
FINJ I BUSY FIRST 1634165
CPU.1(1) = 11
163 CPU.1(2)+PROCCESS.W(6)+10
164 CPU.1(1) TO 166
165 WRITE(2,401)
400 FORIAF(1HV=19H)esy CPU NOT FOUND
166 SUSPENDED EMPTY 304167?
C ANY PROCESS SUSPENDED?
167 FREE EMPTY 304168
168 CYCLE=1
RECYCLE CLICK INT, SERVED
300 DUMMY
C BEGIN PROCESS1 RUNNING
C  >>>>>>>>>>>>>>>>>>>>>>>>>>>
254  PRISET(2) RUNNING
255  FOR I = RUNNING
256  CHECK PROCESS(I,4)
257  PROCESS,1(4) = 1
258  PROCESS,1(6) = 10
259  M=1
260  N=13
261
262  171 CALL EBLOCK(P)
263  INQUEUE,1 EMPTY 3171
264  CALL EBJSYCPU
265  DUMMY
266  C
267  BEGIN PROCESS2 RUNNING.
268  C  >>>>>>>>>>>>>>>>>>>><<<<<<<<<<<
269  PROCESS,2(3) = 1
270  WRITE(2,707)
271  FOR(I=15; SHOK4)
272  PROCESS,2(4) = 300
273  C THIS IS PROCESSING TIME OF PROCESS2
274  TX=3
275  M=2
276  CALL EHAND(TP)
277  TX=2
278  CALL EHAND(TP)
279  N=6
280  CALL EFLTC1(P)
281  TX=1
282  CALL EHAND(TP)
283  N=15
284
285  190 CALL EBLOCK(P)
286  INQUEUE,2 EMPTY 3190
287  CALL EBJSYCPU
288  DUMMY
289  C
290  BEGIN PROCESS3 RUNNING
291  PROCESS,3(1) = 1
292  PROCESS,3(6) = 310
293  TX=3
294  M=3
295  CALL EHAND(TP)
296  TX=4
297  CALL EHAND(TP)
298  TX=2
299  CALL EHAND(TP)
300  TX=1
301  CALL EHAND(TP)
302  N=15
303
304  200 CALL EBLOCK(P)
305  INQUEUE,3 EMPTY 3200
306  CALL EBJSYCPU
307  DUMMY
308  C
309  BEGIN PROCESS6 RUNNING
310  C  >>>>>>>>>>>>>>>>>>>><<<<<<<<<<<
311  PROCESS,6(7) = 1
312  PROCESS,6(6) = 350
313  M=6
314  N=6
315  CALL ESEEK(P)
316  TX=2
317  CALL EHAND(TP)
318  TX=1
319  CALL EHAND(TP)
320  N=15
320  202  CALL  EBLOCK(P)
321                     INQUEUE,6  EMPTY  0202
322                     CALL  ID'SYCPUI
323                     DISHY
324
325                     C  BEGIN  PROCESS4  RUNNING
326                     G>>>>>>>><><><><><><><><><><><><<<<
327                     PROCESS,4(3)  =  1
328                     PROCESS,4(6)=190
329                     N=4
330                     N=10
331                     CALL  EMSELF(P)
332                     N=15
333                     205  CALL  EBLOCK(P)
334                     INQUEUE,4  EMPTY  0205
335                     CALL  ID'SYCPUI
336                     DISHY
337                     C
338                     C  BEGIN  PROCESS5  RUNNING
339                     G>>>>>>>><><><><><><><><><><><><<<<
340                     PROCESS,5(3)  =  1
341                     PROCESS,5(6)=290
342                     M=5
343                     N=15
344                     310  CALL  EBLOCK(P)
345                     INQUEUE,5  EMPTY  0310
346                     CALL  ID'SYCPUI
347                     DISHY
348                     C  BEGIN  PROCESS6  RUNNING
349                     G>>>>>>>><><><><><><><><><><><><<<<
350                     PROCESS,6(3)  =  1
351                     PROCESS,6(6)=270
352                     M=7
353                     N=2
354                     CALL  EMSELF(P)
355                     N=15
356                     331  CALL  EBLOCK(P)
357                     INQUEUE,7  EMPTY  0331
358                     CALL  ID'SYCPUI
359                     DISHY
360                     C  BEGIN  PROCESS7  RUNNING
361                     G>>>>>>>><><><><><><><><><><><><<<<
362                     PROCESS,7(3)  =  1
363                     PROCESS,7(6)=270
364                     M=7
365                     N=2
366                     CALL  EMSELF(P)
367                     N=15
368                     350  CALL  EBLOCK(P)
369                     INQUEUE,8  EMPTY  0350
370                     CALL  ID'SYCPUI
371                     DISHY
372                     C  BEGIN  PROCESS8  RUNNING
373                     G>>>>>>>><><><><><><><><><><><><<<<
374                     PROCESS,8(3)  =  1
375                     PROCESS,8(6)=150
376                     M=8
377                     TX=1
378                     CALL  E'AND(TP)
379                     TX=1
380                     CALL  E'AND(TP)
381                     N=15
382                     350  CALL  EBLOCK(P)
383                     INQUEUE,8  EMPTY  0350
384                     CALL  ID'SYCPUI
385                     DISHY
386                     C  BEGIN  PROCESS9  RUNNING
387                     G>>>>>>>><><><><><><><><><><><><<<<
388                     PROCESS,9(3)  =  1
389                     PROCESS,9(6)=290
386 \text{m=9} \\
387 \text{N=2} \\
388 \text{CALL EHSELF(P)} \\
389 \text{N=6} \\
389 \text{CALL EFETCH(p)} \\
391 \text{N=15} \\
392 \text{370 CALL EBLOCK(P)} \\
393 \text{INQIEUE,9 EMPTY} \#370 \\
394 \text{CALL EBISYCPU} \\
395 \text{DUMMY} \\
396 \\
397 \text{C BEGIN PROCESS 1) RUNNING} \\
398 \text{C >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>} \\
399 \text{PROCESS,1U(3) EQ 1} \\
400 \text{PROCESS,1U(6)=500} \\
401 \text{N=15} \\
402 \text{403 \#380 CALL EBLOCK(P)} \\
404 \text{INQIEUE,1O EMPTY \#380} \\
405 \text{CALL EBISYCPU} \\
406 \text{FOR I =1,11} \\
407 \text{T,PROCESS,1 EQ 0 \#381\#382} \\
408 \text{408 \text{DUMMY} \\
409 \text{409 \text{DUMMY}} \\
410 \text{C BEGIN CPU FINISHES PROCESSING} \\
411 \text{C >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>} \\
412 \text{PRES3T(2) BUSY,BLOCKED,FREE} \\
413 \text{CYCLE=0} \\
414 \text{FOR I = BUSY} \\
415 \text{CHECK T,CPU,I,CPU.I(1),CPU.I(2)} \\
416 \text{FOR I=BUSY} \\
417 \text{T,CPU,I EQ 0 \#390} \\
418 \text{CPU.I FREE BUSY} \\
419 \text{CPU.I HEAD FREE} \\
420 \text{391 \text{DUMMY} \\
421 \text{392 \text{DUMMY}} \\
422 \text{PRES3T(2) BUSY,FREE} \\
423 \text{FREE EMPTY \#391\#392} \\
424 \text{392 SUSPENDED EMPTY \#391\#392} \\
425 \text{393 CYCLE=1} \\
426 \text{393 CYCLE=1} \\
427 \text{394 \text{RECYLE CLOCK INIT, SERVED} \\
428 \text{395 \text{DUMMY}}} \\
429 \text{C BEGIN SIMULATION ENDS} \\
430 \text{C >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>} \\
431 \text{T,TIME EQ 0 \#400} \\
432 \text{WRITE(2,400)} \\
433 \text{433 4000 FORMAT(1H0,3H-XS) \\
434 \text{WRITE(2,714)} \\
435 \text{435 714 FORMAT(1H0,2X,3H-SIMULATION OF MK 2AL PROCESS ALLOCATOR/1H0,10X,3} \\
436 \text{/Kh=))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))) \\
437 \text{715 FORMAT(1H0,1H-SIMULATED TIME =,13H-2X,12H(MICRO-SECS)/1H0,13HNO,} \\
438 \text{OF CPU =,13H-2X,1H0,51H LANGUAGE USED:CONTROL AND SIMULATION LANG} \\
439 \text{/UAGL;[CSL}//} \\
440 \text{WRITE(2,593) IUT} \\
441 \text{598 FORMAT(1H0,25H4), OF CLOCK INTERRUPTS =,18//} \\
442 \text{WRITE(2,600)} \\
443 \text{600 FORMAT(1H0,16H-MICROSECS,5X,15HTIMES INITIATED,5X,21HEXEXECI} \\
444 \text{SON(MICRO-SECS))} \\
445 \text{FOR I=1,11} \\
446 \text{WRITE(2,602) I,PROC(I,1),PROC(I,2) \\
447 \text{602 FORMAT(1H0,5X,14,10X,18,10X,18) \\
448 \text{WRITE(2,603) PA}} \\
449 \text{603 FORMAT(1H0,2H4), OF PROCESS ALLOCATOR CALLS =,18//} \\
450 \text{WRITE(2,604) LOCKOUT,1(1),LOCKOUT,1(2)
604 FORMAT(1HU, 12HLOCKOUT NAME, 5X, 13HTIMES ENGAGED, 5X, 21HEXECUTION(MIC
/RO-SECS), 1HU, 3X, 6HFREELEN, 9X, 18, 7X, 18)
C
605 WRITE(2, 605) LOCKOUT, 2(1), LOCKOUT, 2(2)
C
606 WRITE(2, 606) LOCKOUT, 3(1), LOCKOUT, 3(2)
C
WRITE(2, 607)
607 FORMAT(1HU, 7HCPU NO., 7X, 25HTIME OCCUPIED(MICRO-SECS), 7X, 21HTIME ID
/LE(MICRO-SECS), 7X, 10XxOCCUPANCY)
FOR I = 1, Z
   IDLE=SIMTIME-CPU, I(2)
   OCCUPANCY=CPU, I(2)/SIM*100, 0
   WRITE(2, 608) I, CPU, I(2), IDLE, OCCUPANCY
608 FORMAT(1HU, 3X, 14, 15X, 14, 10X, 18, 10X, 8, 4, 1H%)
C
WRITE(2, 609)
609 FORMAT(1HU, 53HWAITING-TIME HISTORGRAM FOR PROCESS IN SUSPENDED STAT
/F//)
FOR I = 1, 11
   OUTPUT(1, SUSP, I)
   717 FORMAT(1HU, 19X, 14HPROCESS NUMBER, 14//)
WRITE(2, 611)
611 FORMAT(1HU, 14H THIS IS A TRUE)
EXIT
503 NISHIY
END
BLOCK DATA  

POSITION BEFORE MAIN PROGRAM

THIS SEGMENT GIVES INITIAL VALUES TO ITEMS IN COMMON BLOCKS

ARRAY PROC(11,2), A(3,4), B(4,4), C(2,4), D(2,4)

COMMON/FF/BAR, J, Y, INTRIP, SUSPOINT, N, K, M, PA, TX
COMMON/EE/PROC, A, B, C, D
DATA PROC/22*0/
DATA A/1,2,3,7,3,4,0,3,0,6,4,8/
DATA B/1,2,5,6,2,8,5,3,1,0,6,3,13,12/
DATA C/2,3,1,3,0,4,2,10/
DATA D/0,1,3,2,10,3,0,3,5/
DATA BAR, J, Y, INTRIP, SUSPOINT, N, K, M, PA, TX/10*0/
END
SUBROUTINE CPU_SCHED

TAKES APPROX. 70 MICRO-SECS
CLASS LOCKOUT, 3(2)
CLASS TIME PROCESS, 11(10) SET SUSPENDED, INQUEUE, 11, RUNNING
CLASS TIME CPU, 1(3) SET BUSY, FREE

WIST SUSPWAIT, 11(20, 40, 60)

COMMON/AA/Lockout/BB/PROCESS, SUSPENDED, RUNNING
COMMON/CC/CPU, BUSY, FREE
COMMON/FF/Barrier, Y, INTRIP, SUSPEND, N, K, M/GB/SUSPWAIT
COMMON/HH/T, PROCESS, T, CPU, T, TIME, T, CLINT
COMMON/NN/INQUEUE

LOCKOUT, 2(1)+1
LOCKOUT, 2(2)+2

FUNCTION SUSPENDED MIN(11-L) <50
SEARCH FOR THE HIGHEST PRIORITY SUSPENDED PROCESS
PROCESS.L FROM SUSPENDED
PROCESS.L INTO RUNNING
PROCESS.L 11+1

SET STATE 12+1 11 IN P, D, TO RUNNING
AND -T, PROCESS.L, SUSPEND, L
ADD TIME PROCESS SUSPENDED TO WIST SUSPWAIT

N=1
BAR EQ 2, 90395
T, CPU, Y(1) = L
T, CPU, Y EQ 1 91392
91 DUMMY
92 DUMMY
GO TO 110
95 CPU, INTRIP(1) = L
FIND I BUSY MIN(CPU, I(1)) 40370
FIND NEW VALUE OF LCPU
70 WRITE(2, 80)
80 FORMAT(1HU, 27H NEW VALUE OF LCPU NOT FOUND)
GO TO 110
40 INTRIP=I
GO TO 110
C SEND NEW VALUE OF LCPU TO INT. TRIP. LN
50 WRITE(2, 60)
60 FORMAT(1HU, 34H SUSP PROCESS FOUND IN CPU_SCHED)
70 TIME GE 9 2100
T, CLINT GE 0 120X130
120 DUMMY
130 DUMMY
100 WRITE(2, 147)
140 FORMAT(1HU, 17H-ECPU_SCHED ENTERED)
RETURN
TIME SPENT IN CPU_SCHED IS ADDED IN BUSYCPU
END
SUBROUTINE SYSCHED TAKES APPROX. 50 MICRO-SECS

SUBROUTINE SYSCHED
CLASS PROCESS,11(10) SET SUSPENDED
CLASS CPU,1(3) SET BUSY,FREE

COMMON/10/B/P/R/SU/P/E,F,B/A/J/I
/SUSP/INT

FREE EMPTY 450
FINI 1 SUSPENDED MIN(11-I) 340
SEARCH FOR HIGHEST PRIORITY SUSPENDED PROCESS
GO TO 70
WRITE(2,60)
FORMAT(1HU,J2:11) SUSP PROCESS FOUND IN SYSCHED
GO TO 50

70 IF CPU.INTRIP(1)+50
C SUSPENDED PROCESS HIGHER THAN LCPU CURP?

SUSP/INT=1
SET THE SUSPENDED QUEUE INTERRUPT
WRITE(2,80)
FORMAT(1HU,16)SYSCHED ENTERED
RETURN
END
SUBROUTINE EHAND(TP)
IT HANDS A TASK TO THE CALLED PROCESS
IT TAKES APPROX. 60 MICRO-SECS
CLASS LOCKOUT.3(2)
CLASS TASK.20(5) SET FRTSKLST
CLASS PROCESS.11(10) SET SUSPENDED, INQUEUE.11, RUNNING
ARRAY PROC(11,2),A(3,4),B(4,4),C(2,4),D(2,4)
COMMON/AA/LOCKOUT/DR/PROCESS/SUSPENDED/RUNNING
COMMON/EE/PROC,A,B,C,D
COMMON/FF/BAI,J,Y,INTRP,SUSPQNT,N,K,M,PA,TX
COMMON/DD/TASK,FRSKLST
COMMON/QQ/INQUEUE
PA+1
C RECORD CALLING OF PA
LOCKOUT.1(1)+1
C ENGAGE FREELD
LOCKOUT.1(2)+13
C FREELD TAKES 13 MICRO-SECS
FIND I FRSKLST FIRST =600
TASK.I FROM FRSKLST
FETCH FREE TASK BLOCK
M EN 2 = 100
FOR L=2,4
    TASK.I(L-1) = A(TX,L)
100 M EN 3 = 200
C IS CALLING PROCESS INDEX =3?
FOR L=2,4
    TASK.I(L-1) = B(TX,L)
200 M EN 5 = 300
FOR L=2,4
    TASK.I(L-1) = C(TX,L)
300 M EN 6 = 400
FOR L=2,4
    TASK.I(L-1) = D(TX,L)
400 NUNIMY
C STORE TASK DETAILS FROM APPROPRIATE PROCESS ARRAY
P=TASK.I(1)
C TX IS CALLING PROCESS TASK INDEX
C M IS CALLING PROCESS INDEX
C LINK TASK BLOCK IN PROCESS INPUT Q
LOCKOUT.3(1)+1
LOCKOUT.3(2)+25
PROCESS,M(6)+6
C RECORD TIME SPENT
GO TO 700
600 WRITE(2,7)
C 7 FORMAT(1HU,17H40) FREE TASK LEFT
700 WRITE(2,A80)
800 FORMAT(1HU,17HEHAND(TP) ENTERED)
RETURN
END
SUBROUTINE BLOCK(P)

TAKES APPROX 40 MICRO-SECS

CLASS LOCKOUT, 3(2)
CLASS PROCESS, 11(10) SET SUSPENDED, INQUEUE, 11, RUNNING, BLOCKED
CLASS TASK, 20(5) SET FRSTKLIST

COMI:ION/AA/LOCKOUT/BA/PROCESS, SUSPENDED, RUNNING, BLOCKED
COMI:ION/FF/BAR, J, Y, INTRIP, SUSPQINT, N, K, M, PA
COMI:ION/JO/TASK, FRSTKLIST
COMI:ION/JO/INQUEUE

PA+1
LOCKOUT, 3(1)+1
LOCKOUT, 3(2)+1
FINI: I INQUEUE, M MIN(TASK, I(3)) @ 20
GET FIRST TASK IN Q OF HIGHEST PRIORITY
TASK, I(3) LE 4 @ 20
TASK IN I/P Q WITH PRIORITY > OR = N?
M IS PROCESS INDEX CALLING EBLOCK(N)
N IS PRIORITY IF REQUIRED TASK
FOR X=1,3
PROCESS, I(X)= TASK, I(X)
LOAD TASK IN PROCESS DESCRIPTOR
LOCKOUT, 1(1)*1
LOCKOUT, 1(2)*13
TASK, I FROM INQUEUE, M
TASK, I INTO FRSTKLIST
LINK TASK BACK INTO FREE SPACE LIST

PROCESS, M(?) = 1
SET CONDITION CODES POSITIVE
INQUEUE, M EMPTY 2030

SET STATE TO BLOCKED IN P,D.
PROCESS, M FOR I RUNNING
PROCESS, M (VT) BLOCKED
PROCESS, M(?) = 1
RECORD VALUE OF N IN BLOCKED(N) IN P,D.
PROCESS, M(6)+47
STORcE TIME SPENT IN THE CALL
WRITE(2, 40)
FORMAT(1HU, 17HE) BLOCK(P) ENTERED
RETURN
END
SUBROUTINE EFETCH(P)

>>>

TAKES APPROX 20 MICRO-SECS

CLASS LOCKOUT,3(2)

CLASS PROCESS,11(10) SET SUSPENDED,INQUEUE,11

CLASS TASK,200(5) SET FRSTKLIST

COMMON/AA/LOCKOUT/BB/PROCESS,SUSPENDED

COMMON/DTASK,FRSTKLIST

COMMON/FF/BAR,J,V,INTRIP,SUSPQT,N,K,M,PA

COMMON/GQ/INJUEIE

PA+1

LOCKOUT,3(1)+1

LOCKOUT,3(2)+2

FIND INQUEUE,N MIN(TASK,1(3)) •> 20

GET FIRST TASK IN Q OF HIGHEST PRIORITY

TASK,1(3) = N @ 20

TASK IN I/P Q WITH PRIORITY <.)M= N?

N IS THE PROCESS INDEX CALLING EBLOCK(4)

N IS STATED BEFORE THE CALL IN THE CALLING PROCESS

FOR X=1,3

PROCESS,M(X)=TASK,1(X)

LOAD TASK IN P.D.

LOCKOUT,1(1)+1

LOCKOUT,1(2)+3

TASK,1 FROM INQUEUE,M

TASK,1 INTO FRSTKLIST

LINK TASK BLOCK BACK INTO FREE SPACE LIST

PROCESS,M(?)=1

SET CONDITION CODES POSITIVE

GO TO 31

20 PROCESS,M(9)=N

STORES VALUE OF REQUIRED PRIORITY

PROCESS,M(?)=0

C.

SET CONDITION CODES ZERO

30 PROCESS,M(?)+2

WRITE(2,40)

40 FORMAT(14H0,17HEFETCH(P) ENTERED)

RETURN

END
SUBROUTINE SEEK(P)

TAKES APPRX. 20 MICRO-SECS

CLASS LOCK:1T,3(2)
CLASS PROCESS,11(10) SET SUSPENDED,INQUEUE,11
CLASS TASK,290(5) SET FRTSKLIST

COM:ION/AA/Lockout/BO/Process,Suspended
COM:ION/BO/TASK,FRTSKLIST
COM:ION/FF/Bar,J,J,Y,INTRIP,SUSPRINT,N,K,T,PA
COM:ION/NI/INQUEUE

PA+1
LOCKOUT,3(1)*1
LOCKOUT,3(2)+2

FIND I INQUEUE,H MIN(TASK,I(3)) @40

GET FIRST TASK IN Q OF HIGHEST PRIORITY

TASK,1(3) EQ 4 @20

C TASK IN I/P Q WITH PRIORITY EQUAL N?

M IS THE PROCESS INDEX CALLING SEEK(N)

N IS STATE BEFORE THE CALL IN THE CALLING PROCESS

FOR X=1,3

PROCESS,1(X)=TASK,1(X)

LOAD TASK IN P,0.

LOCKOUT,1(1)+1
LOCKOUT,1(2)+13

TASK,1 FROM INQUEUE,M

TASK,1 INTO FRTSKLIST

LINK TASK BLOCK BACK TO FRTSKLIST

PROCESS,M(7)=1

C SET CONDITION CODES POSITIVE

GO TO 30

20 PROCESS,M(7) = N

STORE VALUE IF N IN SEEK(N)

PROCESS,M(7) = 0

C SET CONDITION CODES ZERO

40 WRITE(2,50)

50 FORMAT(1HU,1SH) TASK IN I/P Q)

30 PROCESS,M(3)+27

WRITE(2,60)

60 FORMAT(1HU,1SH)SEEK(P) ENTERED)

RETURN

END
SUBROUTINE ESELF(P)

TAKES APPROX. 10 MICRO-SECS

CLASS PROCESS,1(10) SET SUSPENDED,INQUEUE

CLASS TASK,200(5)

COMMON/AB/PROCESS,SUSPENDED
COMMON/AD/TASK,FRTSKLIST
COMMON/FF/BAR,J,Y,INTREP,SUSQINT,N,K,H,PA
COMMON/AQ/INQUEUE

PA+1

FIND I INIQUE,MIN(TASK,1(3)) = 40

GET FIRST TASK IN Q OF HIGHEST PRIORITY

TASK,1(1) = 4

SET CALLED PROCESS EQUAL CALLING PROCESS

TASK,1(5) = N

SET PRIORITY EQUAL N

GO TO 60

WRITE(2,50)

FORMAT(1HU,16:14) TASK IN I/P Q

60 FORMAT(M(5),H17)

WRITE(2,70)

70 FORMAT(1HU,17HSELF(P ENTERED))

RETURN

END
SUBROUTINE EHNTS

SUBROUTINE EHNTS AND SET STATE, TAKES APPROX. 40 MICRO-SECS
CLASS LOCKOUT, 3(2)
CLASS TIME PROCESS, 11(10) SET SUSPENDED, INQUEUE, 11, RUNNING, BLOCKED
CLASS TIME CPU, 1(3) SET BUSY, FREE
CLASS TASK, 200(5) SET FRSTKLST

9 C COMMON/AA/LCKOUT/B4/PROCESS, SUSPENDED, RUNNING, BLOCKED
10 C COMMON/CC/CPU, BUSY, FREE
11 C COMMON/DD/TASK, FRSTKLST
12 C COMMON/FF/BAR,J,Y, INTERRUPT, SUSPINT, NEXT, PA, TX
13 C COMMON/H/T, PROCESS, CPU, TIME, T, CLINT
14 C COMMON/QQ/INQUEUE
15 C LOCKOUT, 3(1)+1
16 C LOCKOUT, 3(2)+25
17 W= TASK.J(1)
18 TASK.J(1) INQUE W
19 C LINK TASK BLOCK IN PROCESS I/P Q
20 D LET (2) INQUE W
21 TASK.J(1) LT 64 FREE
22 C TO TEST FOR PERIPHERAL PROCESS, TEST WHETHER PI>4
23 FOR L =1, 11
24 C CHECK PROCESS, L(3) EQ 4 9100
25 C CALLEw PROCESS STATE BLOCKED(N) FOR SOME N
26 C LOCKOUT, 3(1)+1
27 C LOCKOUT, 3(2)+25
28 FINT INQUE W= MIN(TASK.I(3)) 9120
29 PROCESS.W(9) GE TASK.I(3) 9140
30 C RIGHT OF WAIVED TASK AT LEAST 4?
31 PROCESS.W(1)=3
32 C CHANGE STATE WORD IN P.D. TO SUSP(NRAL)
33 T.PROCESS.W=0
34 C START SUSP TIME COUNT
35 LOCKOUT, 2(1)+1
36 C LOCKOUT, 2(2)+14
37 PROCESS.W= CPU1 BLOCKED
38 PROCESS.W= INT3 SUSPENDED
39 C INCLUKE PROCESS IN SUSPENDED STATE MAP
40 GO TO 40
41 60 WRITE(2,90)
42 90 FORMAT(1HU, 4HIP GE 64)
43 GO TO 30
44 100 WRITE(2,111)
45 110 FORMAT(1HU, 17HPROCESS, W(8) NE 4)
46 GO TO 30
47 120 WRITE(2,130)
48 130 FORMAT(1HU, 16H: TASK IN INQ, W)
49 GO TO 30
50 140 WRITE(2,150)
51 150 FORMAT(1HU, 25HPROCESS, W(9) LT TASK.I(3))
52 GO TO 30
53 80 PROCESS.W(6)=4
54 C RECORD TIME SPENT
55 WRITE(2,70)
56 70 FORMAT(1HU, 15HEHNDTSK ENTERED)
57 FOR I =FREE
58 T.CPU.I END 1608170
59 160 DUMP
65  170 DUMIY
66  T.TIME GE 3180
67  T.CLINT GE 3180190
68  180 DUMIY
69  190 DUMIY
70  RETURN
71  END
SUBROUTINE BUSYCPU

IT HUSIES CPU ON WHICH PROCESS RUNNING
 BY TOTAL RUNNING TIME OF THE PROCESS

CLASS TIME PROCESS, 11(10)
CLASS TIME CPU, 1(3) SET BUSY, FREE
ARRAY PROC(11,2)
COMMON/BB/PROCESS/CC/CPU,BUSY,FREE
COMMON/EE/PROC
COMMON/FF/BJ, Y, INTRIP, SUSPPOINT, N, K, M, PA, TX
COMMON/IH/T.PROCESS,T,CPU,T.TIME,T.CLINT

FIND I BUSY FIRST 2000 TO 2500

CPU, I(1) EQ M
2000 T,CPU,I = PROCESS,M(6) + 70
70 IS THE TIME SPENT IN CPU SCHED

CPU, I(2) = PROCESS.M(6)
STORES TIME PROCESS OCCUPIED THIS CPU

PROC(M,1) + 1
PROC(M,2) = PROCESS.M(6)

GO TO 50
2500 WRITE(2,3000)
3000 FOR FORMAT(1H4,1X,HUSBY CPU NOT FOUND)
31 FOR L = 1,11
32 T.PROCESS.L EQ 0 60370
33 60 DUMMY
34 70 DUMMY
35 T.TIME GE 4 180
36 T.CLINT GE 4 90280
37 80 DUMMY
38 90 DUMMY
39 50 WRITE(2,1U0)
40 100 FORMAT(1H4,1X,HUSBY CPU ENTERED)
41 RETURN
42 END
SIMULATION OF MK 2BL PROCESS ALLOCATOR

SCHOOL OF ELECTRICAL ENGINEERING-PLYMOUTH POLYTECHNIC-A.M.SALIH-JULY, 1977

SIMULATED TIME = 1000000 (MICRO-SECS)

NO. OF CPUS = 2

LANGUAGE USED: CONTROL AND SIMULATION LANGUAGE: [CSL]

NO. OF CLOCK INTERRUPTS = 99

<table>
<thead>
<tr>
<th>PERIODIC PROCESS</th>
<th>TIMES INITIATED</th>
<th>EXECUTION (MICRO-SECS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>6860</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>27000</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>29000</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>4710</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>6960</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>26501</td>
</tr>
<tr>
<td>7</td>
<td>25</td>
<td>7210</td>
</tr>
<tr>
<td>8</td>
<td>50</td>
<td>17460</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>6750</td>
</tr>
<tr>
<td>10</td>
<td>25</td>
<td>14460</td>
</tr>
<tr>
<td>11</td>
<td>90</td>
<td>26840</td>
</tr>
</tbody>
</table>

NO. OF PROCESS ALLOCATOR CALLS = 1467

<table>
<thead>
<tr>
<th>LOCKOUT NAME</th>
<th>TIMES ENGAGED</th>
<th>EXECUTION (MICRO-SECS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREELO</td>
<td>2038</td>
<td>26021</td>
</tr>
<tr>
<td>SUSPLO</td>
<td>946</td>
<td>35583</td>
</tr>
<tr>
<td>PROLU</td>
<td>2811</td>
<td>63700</td>
</tr>
<tr>
<td>CPU NO.</td>
<td>TOTAL TIME</td>
<td>TIME OCCUPIED</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>---------------</td>
</tr>
<tr>
<td>1</td>
<td>1000000</td>
<td>86020</td>
</tr>
<tr>
<td>2</td>
<td>1000000</td>
<td>63770</td>
</tr>
</tbody>
</table>

WAITING-TIME HISTOGRAM FOR PROCESS IN SUSPENDED STATE

**PROCESS NUMBER 1**

<table>
<thead>
<tr>
<th>COUNT</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>TO 10</td>
</tr>
<tr>
<td>0</td>
<td>11 TO 110</td>
</tr>
<tr>
<td>0</td>
<td>111 TO 210</td>
</tr>
<tr>
<td>0</td>
<td>211 TO 310</td>
</tr>
<tr>
<td>0</td>
<td>311 TO 410</td>
</tr>
<tr>
<td>0</td>
<td>411 TO 510</td>
</tr>
<tr>
<td>25</td>
<td>511 TO 610</td>
</tr>
<tr>
<td>0</td>
<td>611 TO 710</td>
</tr>
<tr>
<td>0</td>
<td>711 TO 810</td>
</tr>
<tr>
<td>0</td>
<td>811 TO 910</td>
</tr>
<tr>
<td>0</td>
<td>911 TO 1010</td>
</tr>
<tr>
<td>0</td>
<td>1011 TO 1110</td>
</tr>
<tr>
<td>0</td>
<td>1111 TO 1210</td>
</tr>
<tr>
<td>0</td>
<td>1211 TO 1310</td>
</tr>
<tr>
<td>0</td>
<td>1311 TO 1410</td>
</tr>
<tr>
<td>0</td>
<td>1411 TO 1510</td>
</tr>
<tr>
<td>0</td>
<td>1511 TO 1610</td>
</tr>
<tr>
<td>0</td>
<td>1611 TO 1710</td>
</tr>
<tr>
<td>0</td>
<td>1711 TO 1810</td>
</tr>
</tbody>
</table>

**PROCESS NUMBER 2**

<table>
<thead>
<tr>
<th>COUNT</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TO 13</td>
</tr>
<tr>
<td>0</td>
<td>11 TO 110</td>
</tr>
<tr>
<td>0</td>
<td>111 TO 210</td>
</tr>
<tr>
<td>0</td>
<td>211 TO 310</td>
</tr>
<tr>
<td>0</td>
<td>311 TO 410</td>
</tr>
<tr>
<td>0</td>
<td>411 TO 510</td>
</tr>
<tr>
<td>0</td>
<td>511 TO 610</td>
</tr>
<tr>
<td>0</td>
<td>611 TO 710</td>
</tr>
<tr>
<td>0</td>
<td>711 TO 810</td>
</tr>
<tr>
<td>25</td>
<td>811 TO 910</td>
</tr>
<tr>
<td>0</td>
<td>911 TO 1010</td>
</tr>
<tr>
<td>0</td>
<td>1011 TO 1110</td>
</tr>
<tr>
<td>24</td>
<td>1111 TO 1210</td>
</tr>
<tr>
<td>0</td>
<td>1211 TO 1310</td>
</tr>
<tr>
<td>0</td>
<td>1311 TO 1410</td>
</tr>
<tr>
<td>0</td>
<td>1411 TO 1510</td>
</tr>
<tr>
<td>0</td>
<td>1511 TO 1610</td>
</tr>
<tr>
<td>0</td>
<td>1611 TO 1710</td>
</tr>
<tr>
<td>0</td>
<td>1711 TO 1810</td>
</tr>
<tr>
<td>COUNT</td>
<td>RANGE</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>0</td>
<td>0 TO  9</td>
</tr>
<tr>
<td>0</td>
<td>10 TO 19</td>
</tr>
<tr>
<td>0</td>
<td>20 TO 29</td>
</tr>
<tr>
<td>1</td>
<td>30 TO 39</td>
</tr>
<tr>
<td>24</td>
<td>40 TO 59</td>
</tr>
<tr>
<td>0</td>
<td>60 TO 69</td>
</tr>
<tr>
<td>25</td>
<td>70 TO 79</td>
</tr>
<tr>
<td>0</td>
<td>80 TO 89</td>
</tr>
<tr>
<td>0</td>
<td>90 TO 99</td>
</tr>
<tr>
<td>0</td>
<td>100 TO 109</td>
</tr>
<tr>
<td>0</td>
<td>110 TO 119</td>
</tr>
<tr>
<td>0</td>
<td>120 TO 129</td>
</tr>
<tr>
<td>0</td>
<td>130 TO 139</td>
</tr>
<tr>
<td>0</td>
<td>140 TO 149</td>
</tr>
<tr>
<td>0</td>
<td>150 TO 159</td>
</tr>
<tr>
<td>0</td>
<td>160 TO 169</td>
</tr>
<tr>
<td>0</td>
<td>170 TO 179</td>
</tr>
<tr>
<td>0</td>
<td>180 TO 189</td>
</tr>
</tbody>
</table>

**PROCESS NUMBER 4**

<table>
<thead>
<tr>
<th>COUNT</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 TO  9</td>
</tr>
<tr>
<td>0</td>
<td>10 TO 19</td>
</tr>
<tr>
<td>0</td>
<td>20 TO 29</td>
</tr>
<tr>
<td>25</td>
<td>30 TO 39</td>
</tr>
<tr>
<td>0</td>
<td>40 TO 49</td>
</tr>
<tr>
<td>0</td>
<td>50 TO 59</td>
</tr>
<tr>
<td>0</td>
<td>60 TO 69</td>
</tr>
<tr>
<td>0</td>
<td>70 TO 79</td>
</tr>
<tr>
<td>0</td>
<td>80 TO 89</td>
</tr>
<tr>
<td>0</td>
<td>90 TO 99</td>
</tr>
<tr>
<td>0</td>
<td>100 TO 109</td>
</tr>
<tr>
<td>0</td>
<td>110 TO 119</td>
</tr>
<tr>
<td>0</td>
<td>120 TO 129</td>
</tr>
<tr>
<td>0</td>
<td>130 TO 139</td>
</tr>
<tr>
<td>0</td>
<td>140 TO 149</td>
</tr>
<tr>
<td>0</td>
<td>150 TO 159</td>
</tr>
<tr>
<td>0</td>
<td>160 TO 169</td>
</tr>
<tr>
<td>0</td>
<td>170 TO 179</td>
</tr>
<tr>
<td>0</td>
<td>180 TO 189</td>
</tr>
<tr>
<td>COUNT</td>
<td>RANGE</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>0</td>
<td>0 - 10</td>
</tr>
<tr>
<td>0</td>
<td>11 - 20</td>
</tr>
<tr>
<td>0</td>
<td>21 - 30</td>
</tr>
<tr>
<td>0</td>
<td>31 - 40</td>
</tr>
<tr>
<td>0</td>
<td>41 - 50</td>
</tr>
<tr>
<td>0</td>
<td>51 - 60</td>
</tr>
<tr>
<td>0</td>
<td>61 - 70</td>
</tr>
<tr>
<td>25</td>
<td>71 - 100</td>
</tr>
<tr>
<td>0</td>
<td>101 - 110</td>
</tr>
<tr>
<td>0</td>
<td>111 - 120</td>
</tr>
<tr>
<td>0</td>
<td>121 - 130</td>
</tr>
<tr>
<td>0</td>
<td>131 - 140</td>
</tr>
<tr>
<td>0</td>
<td>141 - 150</td>
</tr>
<tr>
<td>0</td>
<td>151 - 160</td>
</tr>
<tr>
<td>0</td>
<td>161 - 170</td>
</tr>
<tr>
<td>0</td>
<td>171 - 180</td>
</tr>
<tr>
<td>0</td>
<td>181 - 190</td>
</tr>
</tbody>
</table>

**PROCESS NUMBER 6**

<table>
<thead>
<tr>
<th>COUNT</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 - 10</td>
</tr>
<tr>
<td>0</td>
<td>11 - 20</td>
</tr>
<tr>
<td>0</td>
<td>21 - 30</td>
</tr>
<tr>
<td>25</td>
<td>31 - 50</td>
</tr>
<tr>
<td>0</td>
<td>51 - 60</td>
</tr>
<tr>
<td>1</td>
<td>61 - 70</td>
</tr>
<tr>
<td>24</td>
<td>71 - 90</td>
</tr>
<tr>
<td>0</td>
<td>91 - 100</td>
</tr>
<tr>
<td>0</td>
<td>101 - 110</td>
</tr>
<tr>
<td>0</td>
<td>111 - 120</td>
</tr>
<tr>
<td>0</td>
<td>121 - 130</td>
</tr>
<tr>
<td>0</td>
<td>131 - 140</td>
</tr>
<tr>
<td>0</td>
<td>141 - 150</td>
</tr>
<tr>
<td>0</td>
<td>151 - 160</td>
</tr>
<tr>
<td>0</td>
<td>161 - 170</td>
</tr>
<tr>
<td>0</td>
<td>171 - 180</td>
</tr>
<tr>
<td>0</td>
<td>181 - 190</td>
</tr>
<tr>
<td>COUNT</td>
<td>RANGE</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>0</td>
<td>TO 10</td>
</tr>
<tr>
<td>0</td>
<td>11 TO 110</td>
</tr>
<tr>
<td>0</td>
<td>111 TO 210</td>
</tr>
<tr>
<td>25</td>
<td>211 TO 310</td>
</tr>
<tr>
<td>0</td>
<td>311 TO 410</td>
</tr>
<tr>
<td>0</td>
<td>411 TO 510</td>
</tr>
<tr>
<td>0</td>
<td>511 TO 610</td>
</tr>
<tr>
<td>0</td>
<td>611 TO 710</td>
</tr>
<tr>
<td>0</td>
<td>711 TO 810</td>
</tr>
<tr>
<td>0</td>
<td>811 TO 910</td>
</tr>
<tr>
<td>0</td>
<td>911 TO 1010</td>
</tr>
<tr>
<td>0</td>
<td>1011 TO 1110</td>
</tr>
<tr>
<td>0</td>
<td>1111 TO 1210</td>
</tr>
<tr>
<td>0</td>
<td>1211 TO 1310</td>
</tr>
<tr>
<td>0</td>
<td>1311 TO 1410</td>
</tr>
<tr>
<td>0</td>
<td>1411 TO 1510</td>
</tr>
<tr>
<td>0</td>
<td>1511 TO 1610</td>
</tr>
<tr>
<td>0</td>
<td>1611 TO 1710</td>
</tr>
<tr>
<td>0</td>
<td>1711 TO 1810</td>
</tr>
<tr>
<td>0</td>
<td>1811 TO 1910</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COUNT</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>TO 10</td>
</tr>
<tr>
<td>0</td>
<td>11 TO 110</td>
</tr>
<tr>
<td>0</td>
<td>111 TO 210</td>
</tr>
<tr>
<td>0</td>
<td>211 TO 310</td>
</tr>
<tr>
<td>25</td>
<td>311 TO 410</td>
</tr>
<tr>
<td>0</td>
<td>411 TO 510</td>
</tr>
<tr>
<td>0</td>
<td>511 TO 610</td>
</tr>
<tr>
<td>0</td>
<td>611 TO 710</td>
</tr>
<tr>
<td>0</td>
<td>711 TO 810</td>
</tr>
<tr>
<td>0</td>
<td>811 TO 910</td>
</tr>
<tr>
<td>0</td>
<td>911 TO 1010</td>
</tr>
<tr>
<td>0</td>
<td>1011 TO 1110</td>
</tr>
<tr>
<td>0</td>
<td>1111 TO 1210</td>
</tr>
<tr>
<td>0</td>
<td>1211 TO 1310</td>
</tr>
<tr>
<td>0</td>
<td>1311 TO 1410</td>
</tr>
<tr>
<td>0</td>
<td>1411 TO 1510</td>
</tr>
<tr>
<td>0</td>
<td>1511 TO 1610</td>
</tr>
<tr>
<td>0</td>
<td>1611 TO 1710</td>
</tr>
<tr>
<td>0</td>
<td>1711 TO 1810</td>
</tr>
<tr>
<td>0</td>
<td>1811 TO 1910</td>
</tr>
<tr>
<td>PROCESS NUMBER</td>
<td>9</td>
</tr>
<tr>
<td>---------------</td>
<td>---</td>
</tr>
<tr>
<td>COUNT</td>
<td>RANGE</td>
</tr>
<tr>
<td>25</td>
<td>TO</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>111</td>
</tr>
<tr>
<td>0</td>
<td>211</td>
</tr>
<tr>
<td>0</td>
<td>311</td>
</tr>
<tr>
<td>0</td>
<td>411</td>
</tr>
<tr>
<td>0</td>
<td>511</td>
</tr>
<tr>
<td>0</td>
<td>611</td>
</tr>
<tr>
<td>0</td>
<td>711</td>
</tr>
<tr>
<td>0</td>
<td>811</td>
</tr>
<tr>
<td>0</td>
<td>911</td>
</tr>
<tr>
<td>0</td>
<td>1011</td>
</tr>
<tr>
<td>0</td>
<td>1111</td>
</tr>
<tr>
<td>0</td>
<td>1211</td>
</tr>
<tr>
<td>0</td>
<td>1311</td>
</tr>
<tr>
<td>0</td>
<td>1411</td>
</tr>
<tr>
<td>0</td>
<td>1511</td>
</tr>
<tr>
<td>0</td>
<td>1611</td>
</tr>
<tr>
<td>0</td>
<td>1711</td>
</tr>
<tr>
<td>0</td>
<td>1811</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROCESS NUMBER</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNT</td>
<td>RANGE</td>
</tr>
<tr>
<td>25</td>
<td>TO</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>111</td>
</tr>
<tr>
<td>0</td>
<td>211</td>
</tr>
<tr>
<td>0</td>
<td>311</td>
</tr>
<tr>
<td>0</td>
<td>411</td>
</tr>
<tr>
<td>0</td>
<td>511</td>
</tr>
<tr>
<td>0</td>
<td>611</td>
</tr>
<tr>
<td>0</td>
<td>711</td>
</tr>
<tr>
<td>0</td>
<td>811</td>
</tr>
<tr>
<td>0</td>
<td>911</td>
</tr>
<tr>
<td>0</td>
<td>1011</td>
</tr>
<tr>
<td>0</td>
<td>1111</td>
</tr>
<tr>
<td>0</td>
<td>1211</td>
</tr>
<tr>
<td>0</td>
<td>1311</td>
</tr>
<tr>
<td>0</td>
<td>1411</td>
</tr>
<tr>
<td>0</td>
<td>1511</td>
</tr>
<tr>
<td>0</td>
<td>1611</td>
</tr>
<tr>
<td>0</td>
<td>1711</td>
</tr>
<tr>
<td>0</td>
<td>1811</td>
</tr>
<tr>
<td>COUNT</td>
<td>RANGE</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>0</td>
<td>1 TO</td>
</tr>
<tr>
<td>0</td>
<td>11 TO</td>
</tr>
<tr>
<td>0</td>
<td>111 TO</td>
</tr>
<tr>
<td>0</td>
<td>21 TO</td>
</tr>
<tr>
<td>0</td>
<td>211 TO</td>
</tr>
<tr>
<td>0</td>
<td>31 TO</td>
</tr>
<tr>
<td>0</td>
<td>311 TO</td>
</tr>
<tr>
<td>0</td>
<td>41 TO</td>
</tr>
<tr>
<td>0</td>
<td>411 TO</td>
</tr>
<tr>
<td>0</td>
<td>51 TO</td>
</tr>
<tr>
<td>0</td>
<td>511 TO</td>
</tr>
<tr>
<td>0</td>
<td>61 TO</td>
</tr>
<tr>
<td>0</td>
<td>611 TO</td>
</tr>
<tr>
<td>0</td>
<td>71 TO</td>
</tr>
<tr>
<td>0</td>
<td>711 TO</td>
</tr>
<tr>
<td>0</td>
<td>81 TO</td>
</tr>
<tr>
<td>0</td>
<td>811 TO</td>
</tr>
<tr>
<td>0</td>
<td>91 TO</td>
</tr>
<tr>
<td>0</td>
<td>911 TO</td>
</tr>
<tr>
<td>0</td>
<td>101 TO</td>
</tr>
<tr>
<td>0</td>
<td>1011 TO</td>
</tr>
<tr>
<td>0</td>
<td>111 TO</td>
</tr>
<tr>
<td>0</td>
<td>1111 TO</td>
</tr>
<tr>
<td>0</td>
<td>121 TO</td>
</tr>
<tr>
<td>0</td>
<td>1211 TO</td>
</tr>
<tr>
<td>0</td>
<td>131 TO</td>
</tr>
<tr>
<td>0</td>
<td>1311 TO</td>
</tr>
<tr>
<td>0</td>
<td>141 TO</td>
</tr>
<tr>
<td>0</td>
<td>1411 TO</td>
</tr>
<tr>
<td>0</td>
<td>151 TO</td>
</tr>
<tr>
<td>0</td>
<td>1511 TO</td>
</tr>
<tr>
<td>0</td>
<td>161 TO</td>
</tr>
<tr>
<td>0</td>
<td>1611 TO</td>
</tr>
<tr>
<td>0</td>
<td>171 TO</td>
</tr>
<tr>
<td>0</td>
<td>1711 TO</td>
</tr>
<tr>
<td>0</td>
<td>181 TO</td>
</tr>
<tr>
<td>0</td>
<td>1811 TO</td>
</tr>
</tbody>
</table>
APPENDIX B

SIMULA LISTING OF SYSTEM X SIMULATOR PACKAGE AND CROSS REFERENCE
**GLOBAL PARAMETERS**

**COMMENT**

* ABDUL SALIH, Dec. 1979

**GEC MARK III HL MULTIPROCESSOR SUBSYSTEM AND**

**CIMER SYSTEM X SUBSYSTEMS SIMULATOR STRUCTURE**

**PROCESS**

```
I
CPU AP OS
I
I
I
RASH SIS DSS CPS ... SA
I

LOCAL EXCHANGE
```

**INTEGER** NUN, TOTALCCTS, INCONCCTS, MININCC, NMINOFPROCESSES,

MAXINCC, MINOUTCCT, MAXOUTCCT, BPCPS, SD1, SD2, SD3, SD4, SD5, SD6,

SD7, SD8, SD9, SD10, SD11, SD12, SD13, SD14, SD15, SD16, SD17, SD18,

SD19, SD20;

**REAL** SIMPERIOD;

**TOTALCCTS** := ININT;

**INCONCCTS** := ININT;

**MININCC** := ININT;

**MAXINCC** := ININT;

**MINOUTCCT** := ININT;

**MAXOUTCCT** := TOTALCCTS;

**NUM** := ININT;

**BPCPS** := 49;

**SIMPERIOD** := INREAL;

**BEGIN**

**PROCEDURE** OUTTV(T, V);

**COMMENT**

**BEGIN**

**PROCEDURE** OUTLINE(T);**END****OUTLINE*****;
PROCEDURE ENCR(NO);
COMMENT---------------------;
INTEGER NO;
BEGIN
OUTERR("###ERROR NO, ");
OUTINT(N,O,3);
END###ENCR PROCEDURE***;

PROCEDURE WRITE(T,V,I);
VALUE T,V; TEXT T,V; INTEGER I;
BEGIN
OUTTEXT(T);
OUTTEXT(V);
OUTINT(1,6);
OUTIMAGE;
END###WRITE***;

BOOLEAN PROCEDURE AND2(A,B); BOOLEAN A,B;
AND2 := IF A THEN B ELSE FALSE;

BOOLEAN PROCEDURE OR2(A,B); BOOLEAN A,B;
OR2 := IF A THEN TRUE ELSE B;

INTEGER PROCEDURE SISREP(CCTNUM); INTEGER CCTNUM;
COMMENT***DETERMINES CP'S CONNECTED TO CCTNUM***;
BEGIN
SISREP := IF AND2(CCTNUM >= 1, CCTNUM <= 58 ) THEN 0
ELSE 1;
END###SISREP***;

INTEGER PROCEDURE CPSREP(CCTNUM); INTEGER CCTNUM;
COMMENT***DETERMINES CPS BUSINESS ACCORDING TO CCTNUM***;
BEGIN
CPSREP := IF AND2(CCTNUM >= 1, CCTNUM <= 65 ) THEN 0
ELSE IF AND2(CCTNUM >= 66 , CCTNUM <= 101 ) THEN 1
ELSE IF AND2(CCTNUM >= 102 , CCTNUM <= 581 ) THEN 2
ELSE IF AND2(CCTNUM >= 582 , CCTNUM <= 1061 ) THEN 3
ELSE 999;
END###CPSREP***;

COMMENT************************AP PROCESS******************************
* DESCRIPTION:
* CONTAINS DATA STRUCTURES AND INTERFACING
* PROCEDURES COMMON TO ALL IBM PROCESSES
* FUNCTION:
* WHEN A PROCESS IS PREFIXED BY AP(P1) IT
* WILL HAVE ALL THE BELOW LISTED ATTRIBUTES
*
* VARIABLES:
*
* PI : PROCESS INDEX - INDICATIVE OF ITS PRIORITY
* CIX : PROCESS ALLOCATOR CALL INDEX
* PSTART : SIMULATED TIME VALUE AT THE START OF A PROLO WAIT
* PERIODIC : BOOLEAN, TRUE FOR A PERIODIC PROCESS
* REMAININGACTTIME: TIME TO COMPLETE AN ACTIVITY WHEN A PROCESS IS
* INTERRUPTED
* T : TIMES PROCESS INITIATED
* MIN : MIN " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " 

PROCEDURE

PROCEDURE

PROCEDURE

PROCEDURE

PROCEDURE

PROCEDURE

PROCEDURE

PROCEDURE

PROCEDURE
PROCEDURE HAND( );
BEGIN
  IF T NE "INT1" THEN FOR I:=0 STEP 1 UNTIL 7 DO
    RELEVANTCPU.G1):=G(1); 
    CIX: =2; 
    ACTIVATE PA AFTER CURRENT; 
    PASSIVATE; 
END*HAND**;

PROCEDURE BLOCK(N); INTEGER N;
BEGIN
  CIX: =5; 
  SI: =N; 
  RELEVANTCPU.CCS:=0; 
  ACTIVATE PA AFTER CURRENT; 
  PASSIVATE; 
END*BLOCK**;

PROCEDURE FETCH(N); INTEGER N;
BEGIN
  CIX: =4; 
  RELEVANTCPU.CCS:=0; 
  ACTIVATE PA AFTER CURRENT; 
  PASSIVATE; 
END*FETCH**;

PROCEDURE SEEK(N); INTEGER N;
BEGIN
  CIX: =3; 
  RELEVANTCPU.CCS:=0; 
  ACTIVATE PA AFTER CURRENT; 
  PASSIVATE; 
END*SEEK**;

PROCEDURE SELF(N); INTEGER N;
BEGIN
  CIX: =1; 
  FOR I:=0 STEP 1 UNTIL 7 DO
    RELEVANTCPU.G1):=G(1); 
    DI:=1; 
    ACTIVATE PA AFTER CURRENT; 
    PASSIVATE; 
END*SELF**;

PROCEDURE FHLOCK(P); INTEGER P;
BEGIN
  CIX: =6; 
  WI:=P; 
  RELEVANTCPU.CCS:=0; 
  ACTIVATE PA AFTER CURRENT; 
  PASSIVATE; 
END*FHLOCK**;

INTEGER PROCEDURE CC;
BEGIN
  CC:=RELEVANTCPU.CCS; 
END*CC**;
END*CLASS AP**PI**;

************** CPU PROCESS ***********************

*DESCRIPTION*: SIMULATES THE RELEVANT CHARACTERISTICS OF A CPU
*FUNCTION*: CREATES A PROCESS ALLOCATION INSTANCE FOR
* THIS CPU, RETAIN REFERENCE TO ITS CPU.

*VARIABLES*:
* CC: CPU CONDITION CODES
* CPP: ADDRESS TO THE CURRENT RUNNING PROCESS
**FUNCTION:** NCDF.LS

**DESCRIPTION:** RESPONSIBLE FOR STORE MANAGEMENT

**ACTIVATED BY:** NONE

**FILE:**

PROCESS CLASS CPU NUMBER); INTEGER NUMBER;
BEGIN
  INTEG RC.CCS:1;
  IF (I}AP(CURF);
  REAL FACT(STARTING);
  REF(PROCESSALLOCATION)MYPA;
  INTEG ER AWAY G(U:15);
  MYPA;NEW PROCESSALLOCATION;
  MYPA.CALLINGPROCESS:CURF;
  MYPA,NTCPU="THIS CPU;"
END**CPU**;

**DESCRIPTION:** STORAGE ALLOCATOR

**FUNCTION:** MODELS ONLY SERVICING OF THE FOLLOWING

**REQUESTS FROM OTHER PROCESSES:** OPEN FILE

**REQUISITE:** CLOSE FILE AND PART-FILE READ, WHEN REQUEST

**RETURNED SA RETURNS TASK TO REQUESTING PROCESS**.

**VARIABLES:**

* OPENFILE LABEL, OPEN FILE REQUEST TO SA
* CLOSEFILE LABEL, CLOSE FILE REQUEST TO SA
* PARTFILE LABEL, PART-FILE READ REQUEST TO SA

**INPUT:**

*REQUEST TO REQUESTING PROCESS

**OUTPUT FROM REQUESTING PROCESS

**ACTIVATED BY:** NONE
**ACTIVATED BY:** NONE

**CLASS STORAGEALLOCATOR;**

**AP CLASS STORAGEALLOCATOR;**

```pascal
A: IF FLAG THEN
  CALLING "RELEVANTCPU.G(2)"=RELEVANTCPU.G(2));
  IF RELEVANTCPU.G(2) = 8 THEN GO TO PARTFILE
  IF RELEVANTCPU.G(2) = 4 THEN GO TO OPENFILE
  ELSE
  IF RELEVANTCPU.G(2) = 5 THEN GO TO CLOSEFILE
  ELSE
  OUTLINE("ILLEGAL ENTRY TO STORAGE_ALLOCATOR*"");
  OPENFILE: HECR(4058.00);
  COMMENT:"TIME TO OPEN FILE"**;
  IF FLAG THEN
  OUTLINE("OPEN FILE REQUEST TO SA");
  G(0)=RELEVANTCPU.G(0);  
  G(1)=RELEVANTCPU.G(1);  
  G(2)=RELEVANTCPU.G(2);  
  RANG;
  COMMENT:"TASK RETURNED TO CALLING PROCESS WITHOUT CHANGE"
  ICTI (HECR) USED AS OCTI TO TRANSLATE TO CALLING
  PROCESS********;
  BLOCK(10);  
  GO TO A1;
  FILE: HOLD(4300.00);
  COMMENT:"TIME TO CLOSE FILE"**;
  IF FLAG THEN
  OUTLINE("CLOSE FILE REQUEST TO SA");
  G(0)=RELEVANTCPU.G(0);  
  G(1)=RELEVANTCPU.G(1);  
  G(2)=RELEVANTCPU.G(2);  
  RANG;
  COMMENT:"SEE COMMENT AFTER RANG ABOVE"**;
  BLOCK(10);  
  GO TO A1;
```
**AP Class RASH;**

```
BEGIN
  00522000 00523000
  00524000 00525000
  00526000 00527000
  00528000 00529000
  00530000 00531000
  00532000 00533000
  00534000 00535000
  00536000 00537000
  00538000 00539000
  00540000 00541000
  00542000 00543000
  00544000 00545000
  00546000 00547000
  00548000 00549000
  00550000 00551000
  00552000 00553000
  00554000 00555000
  00556000 00557000
  00558000 00559000
  00560000 00561000
  00562000 00563000
  00564000 00565000
  00566000 00567000
  00568000 00569000
  00570000 00571000
  00572000 00573000
  00574000 00575000
  00576000 00577000
  00578000 00579000
  00580000 00581000
  00582000
```

**AP Class INTIM;**

```
BEGIN
  00582000 00583000
  00584000 00585000
  00586000 00587000
  00588000 00589000
  00590000 00591000
  00592000 00593000
  00594000 00595000
  00596000 00597000
  00598000 00599000
  00600000 00601000
  00602000 00603000
  00604000 00605000
  00606000 00607000
  00608000 00609000
  00610000 00611000
  00612000 00613000
  00614000 00615000
  00616000 00617000
  00618000 00619000
  00620000 00621000
  00622000 00623000
  00624000 00625000
  00626000 00627000
  00628000 00629000
  00630000 00631000
  00632000 00633000
  00634000 00635000
  00636000 00637000
  00638000 00639000
  00640000 00641000
  00642000 00643000
  00644000 00645000
  00646000 00647000
  00648000 00649000
  00650000 00651000
  00652000 00653000
  00654000 00655000
  00656000 00657000
  00658000 00659000
  00660000 00661000
  00662000 00663000
  00664000 00665000
  00666000 00667000
  00668000 00669000
  00670000 00671000
  00672000 00673000
  00674000 00675000
  00676000 00677000
  00678000 00679000
  00680000 00681000
  00682000
```
PROCESS CLASS INTRIPPLICATES;
BEGIN
  BOOLEAN int10,suspend1,clock1;
  SHORT INTEGER ARRAY ptable[1;10;130];
  REF(CPU)lcpu;
  REF(ap,cusp,cup);
  REF(process allocator)lpa;
  START: lpa:=lcpu.xpa;
  cusp:=lpa.callingprocess;
  IF flag THEN
    OUTLINE("INTERIP ENTERED NOW!");
  IF (NOT(cuspof suspend1) THEN
    BEGIN
      erop(5);
      outmage;
      outvec("At Time=",time);
      outvec("Lcpu is no ",lcpu.number);
      WRITE "And Lcpu CURP is ",lcpu.curt,lcpu.curt.pi;
      OUT;
      GO TO ESCAPE;
    END;
  ELSE
    BEGIN
      IF suspend1 THEN
        BEGIN
          suspend1:=false;
          COMMENT**STEER INTERRUPT TO LCPU**;
          WHILE NOT lpa.ideal THEN WAIT(lpa);
          OUT;
          i:=0;
          WHILE (i NOT SUSPMA(1) OR i LE 126) DO
            i:=i+1;
          IF i NOT lpa.callingprocess.pi THEN
            BEGIN
              IF (cusp=none AND cusp.pi) THEN Activate cusp
              IF lpa.callingprocess.ideal THEN Activate lpa.callingprocess;
              OUTLINE("SUSPPOINT NOT SERVED"); GO TO ESCAPE;
            END;
            IF flag THEN
              END;
        END;
      END;
    END;
  END;
END;

COMMENT******** INTERRUPT TRIPPLICATES ********

* DESCRIPTION:
  THIS MODULE MODELS THE ACTIVITIES OF
  THE INTERRUPT TRIPPLICATES UNITS

* FUNCTION:
  NETAINS REF TO PA ON LCPU WHEN SUSPENDED
  QUEUE IS TRIGGERED OR A CLOCK INTERRUPT
  ARRIVES IT INTERRUPTS LCPU

* VARIABLES:
  INT: INTERRUPT TRIPPLICATES LOCAL OUT
  SUSPONT: TRUE WHEN SUSPENDED QUEUE IS TRIGGERED
  CLOCK: TRUE WHEN A CLOCK INTERRUPT OCCURS
  PTABLE: TABLE OF PERIODIC PROCESSES
  LCPU: CPU RUNNING LOWEST PRIORITY PROCESS
  CUSP: CURRENT SUSPENDED PROCESS
  CURP: CURRENT LPA CALLED PROCESS
  LPA: PROCESS ALLOCATOR CN-LCPU

* INPUT TO: PA
  OUTPUT FROM: PA,CLOCAINT
  ACTIVATES: LPA
  ACTIVATED BY: PA,CLOCAINT

************* COMMENT ENDS *************;

PROCESS CLASS INTRIPPLICATES:
BEGIN
  BOOLEAN int10,suspend1,clock1;
  SHORT INTEGER ARRAY ptable[1;10;130];
  REF(CPU)lcpu;
  REF(ap,cusp,cup);
  REF(process allocator)lpa;
  START: lpa:=lcpu.xpa;
  cusp:=lpa.callingprocess;
  IF flag THEN
    OUTLINE("INTERIP ENTERED NOW!");
  IF (NOT(cuspof suspend1) THEN
    BEGIN
      erop(5);
      outmage;
      outvec("At Time=",time);
      outvec("Lcpu is no ",lcpu.number);
      WRITE "And Lcpu CURP is ",lcpu.curt,lcpu.curt.pi;
      OUT;
      GO TO ESCAPE;
    END;
  ELSE
    BEGIN
      IF suspend1 THEN
        BEGIN
          suspend1:=false;
          COMMENT**STEER INTERRUPT TO LCPU**;
          WHILE NOT lpa.ideal THEN WAIT(lpa);
          OUT;
          i:=0;
          WHILE (i NOT SUSPMA(1) OR i LE 126) DO
            i:=i+1;
          IF i NOT lpa.callingprocess.pi THEN
            BEGIN
              IF (cusp=none AND cusp.pi) THEN Activate cusp
              IF lpa.callingprocess.ideal THEN Activate lpa.callingprocess;
              OUTLINE("SUSPPOINT NOT SERVED"); GO TO ESCAPE;
            END;
            IF flag THEN
              END;
        END;
      END;
    END;
  END;
END;

COMMENT******** INTERRUPT TRIPPLICATES ********

* DESCRIPTION:
  THIS MODULE MODELS THE ACTIVITIES OF
  THE INTERRUPT TRIPPLICATES UNITS

* FUNCTION:
  NETAINS REF TO PA ON LCPU WHEN SUSPENDED
  QUEUE IS TRIGGERED OR A CLOCK INTERRUPT
  ARRIVES IT INTERRUPTS LCPU

* VARIABLES:
  INT: INTERRUPT TRIPPLICATES LOCAL OUT
  SUSPONT: TRUE WHEN SUSPENDED QUEUE IS TRIGGERED
  CLOCK: TRUE WHEN A CLOCK INTERRUPT OCCURS
  PTABLE: TABLE OF PERIODIC PROCESSES
  LCPU: CPU RUNNING LOWEST PRIORITY PROCESS
  CUSP: CURRENT SUSPENDED PROCESS
  CURP: CURRENT LPA CALLED PROCESS
  LPA: PROCESS ALLOCATOR CN-LCPU

* INPUT TO: PA
  OUTPUT FROM: PA,CLOCAINT
  ACTIVATES: LPA
  ACTIVATED BY: PA,CLOCAINT

************* COMMENT ENDS *************;

PROCESS CLASS INTRIPPLICATES:
BEGIN
  BOOLEAN int10,suspend1,clock1;
  SHORT INTEGER ARRAY ptable[1;10;130];
  REF(CPU)lcpu;
  REF(ap,cusp,cup);
  REF(process allocator)lpa;
  START: lpa:=lcpu.xpa;
  cusp:=lpa.callingprocess;
  IF flag THEN
    OUTLINE("INTERIP ENTERED NOW!");
  IF (NOT(cuspof suspend1) THEN
    BEGIN
      erop(5);
      outmage;
      outvec("At Time=",time);
      outvec("Lcpu is no ",lcpu.number);
      WRITE "And Lcpu CURP is ",lcpu.curt,lcpu.curt.pi;
      OUT;
      GO TO ESCAPE;
    END;
  ELSE
    BEGIN
      IF suspend1 THEN
        BEGIN
          suspend1:=false;
          COMMENT**STEER INTERRUPT TO LCPU**;
          WHILE NOT lpa.ideal THEN WAIT(lpa);
          OUT;
          i:=0;
          WHILE (i NOT SUSPMA(1) OR i LE 126) DO
            i:=i+1;
          IF i NOT lpa.callingprocess.pi THEN
            BEGIN
              IF (cusp=none AND cusp.pi) THEN Activate cusp
              IF lpa.callingprocess.ideal THEN Activate lpa.callingprocess;
              OUTLINE("SUSPPOINT NOT SERVED"); GO TO ESCAPE;
            END;
            IF flag THEN
              END;
        END;
      END;
    END;
  END;
END;

COMMENT******** INTERRUPT TRIPPLICATES ********

* DESCRIPTION:
  THIS MODULE MODELS THE ACTIVITIES OF
  THE INTERRUPT TRIPPLICATES UNITS

* FUNCTION:
  NETAINS REF TO PA ON LCPU WHEN SUSPENDED
  QUEUE IS TRIGGERED OR A CLOCK INTERRUPT
  ARRIVES IT INTERRUPTS LCPU

* VARIABLES:
  INT: INTERRUPT TRIPPLICATES LOCAL OUT
  SUSPONT: TRUE WHEN SUSPENDED QUEUE IS TRIGGERED
  CLOCK: TRUE WHEN A CLOCK INTERRUPT OCCURS
  PTABLE: TABLE OF PERIODIC PROCESSES
  LCPU: CPU RUNNING LOWEST PRIORITY PROCESS
  CUSP: CURRENT SUSPENDED PROCESS
  CURP: CURRENT LPA CALLED PROCESS
  LPA: PROCESS ALLOCATOR CN-LCPU

* INPUT TO: PA
  OUTPUT FROM: PA,CLOCAINT
  ACTIVATES: LPA
  ACTIVATED BY: PA,CLOCAINT

************* COMMENT ENDS *************;

PROCESS CLASS INTRIPPLICATES:
BEGIN
  BOOLEAN int10,suspend1,clock1;
  SHORT INTEGER ARRAY ptable[1;10;130];
  REF(CPU)lcpu;
  REF(ap,cusp,cup);
  REF(process allocator)lpa;
  START: lpa:=lcpu.xpa;
  cusp:=lpa.callingprocess;
  IF flag THEN
    OUTLINE("INTERIP ENTERED NOW!");
  IF (NOT(cuspof suspend1) THEN
    BEGIN
      erop(5);
      outmage;
      outvec("At Time=",time);
      outvec("Lcpu is no ",lcpu.number);
      WRITE "And Lcpu CURP is ",lcpu.curt,lcpu.curt.pi;
      OUT;
      GO TO ESCAPE;
    END;
  ELSE
    BEGIN
      IF suspend1 THEN
        BEGIN
          suspend1:=false;
          COMMENT**STEER INTERRUPT TO LCPU**;
          WHILE NOT lpa.ideal THEN WAIT(lpa);
          OUT;
          i:=0;
          WHILE (i NOT SUSPMA(1) OR i LE 126) DO
            i:=i+1;
          IF i NOT lpa.callingprocess.pi THEN
            BEGIN
              IF (cusp=none AND cusp.pi) THEN Activate cusp
              IF lpa.callingprocess.ideal THEN Activate lpa.callingprocess;
              OUTLINE("SUSPPOINT NOT SERVED"); GO TO ESCAPE;
            END;
            IF flag THEN
              END;
        END;
      END;
    END;
  END;
END;
I ACTIVATE LPA AFTER CURRENT; END**SUSPOINT***;

WHILE NOT LPA.IDLE DO WAIT(LPAO);

IF FLAG THEN GOTO("INT. TRIPICATES ENTERED FOR CLOCKINT",TIME);

OUT();

IF NOT LPA.IDLE THEN

CLOCKINT:=TRUE;

BEGIN

CLOCKINT:=FALSE;

IF FLAG THEN GOTO("INT. TRIPICATES ENTERED FOR CLOCKINT",TIME);

OUT();

INPUT FROM: INTERRUPT TRIPICATES, PROCESSES/CALLS)

**NOT CLIC~INT**;

END**CLOCKINT**;

COMMENT**************** PROCESS ALLOCATOR ***********

INPUT TO: INTERRUPT TRIPICATES, PROCESSES/CALLS)

**INTERURPTED**;

BEGIN

GETFI: CUTGO TASK INDEX

SYSCHED(): ENGAGES PROCESS RUNNING ON CPU

CTRL(CPU): ENGAGES PROCESS RUNNING ON CPU

HANDLED(): ENGAGES PROCESS RUNNING ON CPU

SUSPOINT(): ENGAGES PROCESS RUNNING ON CPU

**SUSPENDED**;

PROCEDURE:

**-------**

* PBK/PROC 1: ENGAGES PRODUCE WAIT UNTIL RELEASED

* QLENTH(Job): CALCULATES PROCESS MAX AND MIN Q LENGTH

* FRELC2/ENGAGES FREELO AND FETCHES A FREE TASK BLOCK TO PREP

* FRENCH/ENCOMPENSES FREELO AND SETS TASK BLOCK TO FREE

* SPACE LIST

* SUSPLOT(SUSPLOT): INCLUDES A PROCESS IN SUSP. STATE MAP

* ENGAGING SUSPLOT

* SUSPLOT: REMOVES A PROCESS FROM SUSPENDED STATE MAP

* SUSPLOT: ENGAGING SUSPLOT

* LOADATI: LOADS TASK DETAILS INTO CPU G0-0? AND PROCESSES

* SYSC/ENGAGES INTO AND TRIGGERS SUSP,0 INTERRUPT IF

* ASUSPEND: ENGAGES INTO AND TRIGGERS SUSP,0 INTERRUPT IF

* ASUSPEND: ENGAGES INTO AND TRIGGERS SUSP,0 INTERRUPT IF

**SUSPENDED**;

PROCEDURE:

**-------**
PROCESS CLASS PROCESS ALLOCATION:

BEGIN
  INTEGER LASTTASK, L, J, OCT, Y, PAC;
  PAINT, PSTART, SUBSTART, INSTANT;
  REAL PAOVERHEAD, STARTTIME, PROVENHEAD;

  PROCESS INTERRUPT CLOCK:

  IF AP CALLING PROCESS CALLED PROCESS, UNGUARD

  READ IN PPEUS (SELECT, INP, SWINGING, BLOCKING, LOCKING);

  IF CPU IN WCPU;

  IF TASK BLOCK TASK HANDED, TASK FOUND, FREE TASK BLOCK;

  END PROCESS FREE LO( PROC);

  IF AP JOB;

  BEGIN

  PROCEDURE FREE LO:

  BEGIN

  FREE LO := FALSE;

  FREE LO := TRUE;

  END PROCESS FREE LO 2;

  BEGIN

  COMMENT***GETS AFFRE TASK BLOCK***;

  FREE LO := TRUE;

  COMMENT***GETS AFFRE TASK BLOCK***;

  FREE LO := FALSE;

  IF FREE LO := TRUE;

  FREE LO := TRUE;

  FREE LO := TRUE;

  COMMENT***GETS AFFRE TASK BLOCK***;

  FREE LO := TRUE;

  FREE LO := TRUE;

  BEGIN

  NEW TASK BLOCK, INTO FREE TASK LIST;

  GO TO TASK IN;

  END;

  FREE TASK BLOCK, OUT;

  FREE LO := FALSE;

  ACTIVATE FREE LO, FIRST;

  END***FREE LO, FIRST;
PROCEDURE SUSPLO;
BEGIN
  SUSPLO:=FALSE;
  SUSPEND:=SUSPEND, FIRST;
END*SUSPLO;**;**;

BEGIN
  INTEGER I;
  SUSPENDED:=SUSPENDED+1;
  SUSPENDED:=SUSPENDED,RUNNING;
  SUSPENDED:=SUSPENDED,FIRST;
END

PROCEDURE SUSPLO2;
BEGIN
  IF SUSPENDED THEN SUSPEND:=SUSPEND, FIRST;
END

BEGIN
  INTEGER I;
  IF SUSPENDED THEN SUSPENDED:=SUSPENDED,FIRST;
END

BEGIN
  BEGIN
    IF SUSPENDED THEN SUSPENDED:=SUSPENDED,FIRST;
  END
END

PROCEDURE ICATASK;
BEGIN
  IF NOT SUSPENDED THEN SUSPEND:=SUSPEND, FIRST;
END

BEGIN
  BEGIN
    IF SUSPENDED THEN SUSPENDED:=SUSPENDED,FIRST;
  END
END

BEGIN
  BEGIN
    IF SUSPENDED THEN SUSPENDED:=SUSPENDED,FIRST;
  END
END

BEGIN
  BEGIN
    IF SUSPENDED THEN SUSPENDED:=SUSPENDED,FIRST;
  END
END

BEGIN
  BEGIN
    IF SUSPENDED THEN SUSPENDED:=SUSPENDED,FIRST;
  END
END
BEGIN

COMMENT*NPROCEDURE CPUSCHED**;
BEGIN
INTEGER i,MAX,R;
SUSPZ;
MAX:=0;
FOR i:=1 STEP 1 UNTIL NUM DO IF CLMT.C(I),CURP.PI>MAX THEN MAX:=CLMT.C(I),CURP.PI;
COMMENT**COMPARE ALL CPUS TO FIND LCPU***;
INSTAT:=TIME;
WHILE INTAPI.INTLO DO #WAIT(INTLO); IF IMAX=INTLO.CARDINAL THEN IMAX=IMAX; IF IMAX lt IMIN THEN IMIN=IMAX;
INSTAT=INSTAT+TIME; IMAX:=IMAX+1;
PR:=1;
IF PR=1 THEN
COMMENT**INTERCEPT TIME at TIME***;
CURRY("BY PROCESS ALLOCATOR ON CPU NO.",*LCPU.NUMBER); 01009000
WRITE("BECAUSE HIGHEST PRI0. SUSP PROCESS IS *P(I),T,J); 01009000
WRITE("AND LCPU CURP IS *LCPU,CURP.PI); 01009000
CURRY("AND LCPU IS NO. *,LCPU.NUMBER); 01009000
CUTIME;
END;
HOLD(1,6);
COMMENT**INTERCEPT TIME***;
IF (PROCP.SS,PU(0)=4 AND TASKHANDED,TASKI2)(=CALLF.DPRCCFSS,PDC')( THEN Y:=1,0 ELSE Y:=0,0;
IF (CALLINGPRUCESS,CIX=2 THEN SECS:=28,6+10,R*I.+10,0*X+16,2Y ELSE SECS:=10,0*X+7,4*Y;
COMMENT**WHEN CIX=2 SECS IS PROLO ACTIVITY TIME IN SELF CALL WHEN AS OTHER SECS IS PROLO ACTIVITY TIME IN SELF CALL TO PA********; 01009000
-12~ULA

bll

•. 9

70

71
72
73
74
75
76

77
7!:1
79
oo

>1
~2
>3
>4

;s
;6

'7
<li

)9

lO

l1

l2
!3
!4
!5

lb
!7

!8
l9
10

11
12

)3

14

IS
)6

!7
!8
19

.o
.1
2

3
4

6·

.6

7

8

9

:o
:1

2
3

4

6

6
7
tl

9
0
1

2
3
4

5

6
7

8
9

0
1
2
3
4

5

6

7

8
9

0

1
2
3

'>
;

7
~
~

3

l
!
l
I

to7 (VEhS

O~,UO)

0111611000 105b
01069000
010701100
f'l\OCF.!l UR E. IIA NUTA.!oKANUSETSTA n:;
01071000
COMMENT**OhANUS lHE TASK To THE CALLED PROCESS AND
010721100
.!o~TS !TS ETATEOO;
01073000
01074000 1157
IIEGIN
RH( HI All )C;
01075000
l<bH TASKHLCCK )X;
01076000
IH.F( Ll N~AGE )Y;
01077000
0107&000
I'ROCEDUkl! SU.!oPHOC;
01079000
Ui:GIN
01080000 1158
010111000
CO/olMEII<T*** SUSI'P.NDS PkOo.ESS AND INst.RTS IN SUSP IIAP ***;
CALLEtHOC•SS,PD( ~) := J;
01 0112 0110
IIOLI)( SkCS l;
010113000
CALLE.~F~OCE.SS,PNOLO := FALSE;
01084000
SUSPLLl(CALLEDPWOCESSJ;
010115000
QLENGl'H( CALLI::DPROCESS J;
01086000
ACTIVATE CALLEDI'ROCES.!o,t'POLOQ,FlRST;
01087000
END •••susP~oc•••;
010RI!OOO E68
01011qooo
PROCbDURE NOSUSPROC;
01oqoooo
BEGIN
010'11000 116q
WIHTE( "PROCESS HANDED TASK BUT NOT" SUSPENDED IS",CALLBDPROCESS,·r,
01092000
CALLEDFhOCESS,t'lJ;
'
01 oq3ooo
UOLil( SECS l;
01094000
CALLE.Ilf90CUSS,PNOLO := FALSE;
01095000
ACTIVATE C~LLEDPFOCESS, t'POLOQ,F I liST;
01096000
01097000 1!69
END*** NCSUSPROC***i
t'ROL011CALLEDPROCESSJ;
01098000
o:-CALLEDPROCESS,INPUTQ;
01099000
Y:-o;
01100000
IF NOT Q,EMP'fY THEN
01101000
BEGIN
01102000 1160
IF OoLAST OUA TASK8LOCK,TASKC2 I Lll TASKRANDED,TASitC2)
01103000
01104000
THEN l :-C. LAST ELSE
FOR x:-Y,SUC OUA TASKIILOCK WHILE. X,TASKC21 LE TASKBANDEU,TASKC21 DO
01105000
v:-x;
01106000
01107000 E&O
FND.OI NPUl Q NUT EllPTY***i
TASKIIA NllED,FOLLOW( Y );
01108000
01109(100
CON!Ilili<T*OOENOOOO~ INSER"IlNG TASK•***i
11' P LAC lHFN
01110000
BLGIN
01111000 861
Wl!lTI>I <lfROCESS HANDEII TASK IS
"tCALLEDPJlOCESS,T,CALLEDPPOCESS.PI l i
01112000
OUTTV( "AT RUNTIIIE = 11 ,Tl11Eii
01113000
11
OUTTVC NO. OF TASo.S IN INPUTQ :n,o,CARDINALJ;
01114000
IF CALLINCJPI!O.:ESS,T NE 11 BACKGR.OUND 11 THEN
01116000
11 ,CALLI~GPROCESS,T,CALLINGPPOCESS,PII;
WRITE( 11 HANiliNG PROCESS IS
01116000
OUT IMAGE;
01117000
OUTLINEC 11 !NPUTO lASKS PIIIONITIES ARE:-nl;
01118000
y:-o;
0111qooo
FOR Y:-Y,SUC WHILE Y=/=NONE DO
01120000
OUTINTCY OUA TASKBLOCKoTASKC21,4J;
01121000
01122000
OUT I IIAGE;
'
01123000 E61
· BND;
01124000
IF CALLBDPIIOCESS, PENIODI: THEN
IILGIN
,
01126000 862
IF AND2(CALLEDPIIOCESS,PUCII I = 4,TASIUIANDI!DoTASitC21 = 6) rBEN.
01126000
SUSPRCC LLSE NOSUSPRoC
01127000
01128000 E62
ENo•••o~ A PERIODIC PROCES~***
01129000
ELSE
01130000 863
· BEGIN
11' AND2CCALLEOPROCESS,PD(II l = 4 , TASKIIAHDED.TASK.( 2 I LE
01131000
CALLEUFROCESS,PD(q) ) liiEN SUSPROC ELSE NOSUSPROC;
01132000
01133000 E63
END*** OF AN APEIHODJC PIIOCESSO*OO;
END• Oil AN DT ASJo.AN D SETSTA TE 0 **;
01134000 E67
01135000
01136000
01137000
COMMENT****••••****** 5TA"RT OF PROCESS ALLOCATOR ACTIONS ••*•*****;
01138000
01139000
PASTA !IT:
01140000
STARTTIME :• TIME;.
01141000
IF JNTEPRUF1 TliEN CO 10 INT ELSE
01142000 864
BEGIN
01143000
PAC: =PAC+ U
01144000
Go TO CLJNDEX(CALLUoGPROCESS,ClXJ;
01145000 E64
E.ND;
01146000
INT: INTEI!,UPT:=FALSF.;
01147000
PAINT:=PAII'T+1;
01148000
COMNE~T******'Oooooooooo•;
01149000
IF Plt\G THEN
01150000 1166
HEGIN
01151000
OUTTVC 11 CFU INTEIIIIUPTI>U lli N0, 11 ,MYCPU,NUIIBER);
01152000
OUTTV( 11 A~U ITS INTEIIIIUP'I N0, 10 , PAINT J;
01153000
ODTTV( 11 Al RUNTI.ME = 11 ,Tl11E);
.
01154000
OUT l.MASJE;
·
01155000 E65
END;
01151>000
·COMMENTOOOOOOOOOOOOO**********•oooeo;
01157000
• IIOLD( 122,0 1i .
011511000
COIIkENT**RE-ENT, TINE TO SCANNING OF INTo LEVELS**;
01159000
RUNNINGPROCESS:•CALLINGPROCE.SS;
01160000
FOR J:=O STl P 1 UNTIL 7 DO
01161000
·PUNNJ NGPI!OCF.SS, PD( 1+24 ): =IIIYCPU,G( I J;
01162000
RUNNINCJPIIOCE.SS,PD(23):=MYCPU,CCS"
01163000
COMMENTOONEST GRS IN PDC24-31)00.00;
01164000
RUNNINGPROCESS,PDCRJ:=2;


CALLING PROCESS;
BEGIN
  IF Suspended THEN
    SUSPEND=
    SUSPEND(0); // suspending
  MCYCU.HG:=TIME; // MCYCU.HG TIME=TIME
  MCPU, STARTTIME;
  WHILE Interrupt INTR. DO WRITE(Interrupt);
  Interrupt:=INTERRUPT;
  IF suspend THEN suspend:=Suspended;
  Interrupt:=Interrupt;
  Instant:=0;
  INTR:=INTR;
  BEGIN
    IF suspend THEN
      SUSPEND(0); // suspending
  MCYCU.HG:=TIME; // MCYCU.HG TIME=TIME
  MCPU, STARTTIME;
  WHILE Interrupt INTR. DO WRITE(Interrupt);
  Interrupt:=INTERRUPT;
  Instant:=0;
  INTR:=INTR;
  BEGIN
    IF suspend THEN
      SUSPEND(0); // suspending
  MCYCU.HG:=TIME; // MCYCU.HG TIME=TIME
  MCPU, STARTTIME;
  WHILE Interrupt INTR. DO WRITE(Interrupt);
  Interrupt:=INTERRUPT;
  Instant:=0;
  INTR:=INTR;
  BEGIN
    IF suspend THEN
      SUSPEND(0); // suspending
  MCYCU.HG:=TIME; // MCYCU.HG TIME=TIME
  MCPU, STARTTIME;
  WHILE Interrupt INTR. DO WRITE(Interrupt);
  Interrupt:=INTERRUPT;
  Instant:=0;
  INTR:=INTR;
  BEGIN
    IF suspend THEN
      SUSPEND(0); // suspending
  MCYCU.HG:=TIME; // MCYCU.HG TIME=TIME
  MCPU, STARTTIME;
  WHILE Interrupt INTR. DO WRITE(Interrupt);
  Interrupt:=INTERRUPT;
  Instant:=0;
BEGIN  01252000  072
  IF Unsuspended RemainingActivTime > 0.00 THEN Activate
  Unsuspended at (Time*Unsuspended, RemainingActivTime) ELSE Activate
  Unsuspended After Comment;
END;  01265000
BEGIN  01267000
  Comment*** To Account For Remaining Activity Time
  Comment** The Init.kwntf4 Process****
  Comment*******************************;
IF flag THEN  01269000
BEGIN  01270000
  Unsuspended 1000  073
  Write:"Process selected to run for", Unsuspended, T, Unsuspended, PI;
  Cutty**"Cpu no. =", MyCPU, Number);
  Cutty"at Fetime =", Time);
  Cutinage;
END;  01275000
BEGIN  01276000
  Comment*******************************;
  End** Of SUSF INT ***
ELSE  01280000  069
BEGIN  01281000
  If Laststate=3 THEN  01283000
BEGIN  01284000  075
  Unsuspended, INIT: = Unsuspended, INIT:;
  Comment*** Init STores Time When AP Initiated***;
  If CallingProcess.net"Background" THEN  01287000
  MyCPU, Starttime := Time;
  Hold;
  Comment*** Retract Time When Exiting to Sosp(UNBLK) Process
  Following: Block, Trap or Interrupt****;
  ProcLl(Unsuspended);
  Hold(3,5);
  Comment*** Prolo Activity Time****;
  If CallingProcess, Input, First IS TaskBlock THEN  01290000
  Taskpoint"CallingProcess, Input, First
  Que Tasklock Else Error(7);
  TaskfnnOut:
  Unsuspended, Prolo := FALSE;
  Activate Unsuspended, Prolo, First;
  Loadtask;
  Comment*******************************;
  If (Flag AND CallingProcess, PI ne 16) THEN  01300000
BEGIN  01301000
  Write:"Loaded Task in Go"=7 FOR Process","CallingProcess, T;
  CallingProcess, PI;
  For i := 0 Step 1 Until 7 Do  01306000
BEGIN  01308000  077
  Cutin(Mycpu, (1), 5);
  Cutinage;
END;  01310000
END;  01312000  077
BEGIN  01313000
  Comment*******************************;
  Press1;
  MyCPU, CCS := 1;
  If CallingProcess, PI > 100 THEN  01315000
  MyCPU, Starttime := Time;
  Pauoverhead := Pauoverhead + Time - Starttime;
  If (CallingProcess, PI) 30 AND CallingProcess, PI <50 ) THEN  01319000
  Froundhead := Froundhead + Time - Starttime;
  CallingProcess, RelevantCpu := MyCPU;
  If Lpa, First := None THEN Activate Unsuspended
  After Current ELSE
  If (Lpa, First /= None AND INTRIP, LPA = (THIS
  Process allocator)) THEN  01323000
BEGIN  01325000
  INTRIP, CUP("CallingProcess;
  Activate LPA, First After Current;
END  01328000
BEGIN  01329000  078
ELSE  01330000
BEGIN  01331000
IF (LPA, First /= None AND INTRIP, LPA = (THIS
Process allocator)) THEN Activate Unsuspended
After Current;
BEGIN  01335000
IF Flag THEN  01336000
BEGIN  01336000  079
  Write("Process selected to run for", Unsuspended, T, Unsuspended, PI;
  Cutty"To run on CPU no. =", MyCPU, Number);
  Cutty"No. of Tasks in Input Queue =", Unsuspended, Input, Cardinal);
  Cutty"At runtime =", Time);
  Cutinage;
END;  01341000
BEGIN  01343000
  Comment*******************************;
  Passival;
  Goto Fastart;
END** SUSP UNBL ***;
END;  01346000  074
BEGIN  01347000
FETching;
BEGIN  01348000
BEGIN  01349000
BEGIN  0135000
BEGIN  01351000
BEGIN  01352000
BEGIN  01353000
BEGIN  01354000
BEGIN  01355000
BEGIN  01356000
BEGIN  01357000
BEGIN  01358000
BEGIN  01359000
BEGIN  0136000
BEGIN  0136100
BEGIN  0136200
BEGIN  0136300
BEGIN  0136400
BEGIN  0136500
BEGIN  0136600
BEGIN  0136700
BEGIN  0136800
BEGIN  0136900
BEGIN  0137000
BEGIN  0137100
BEGIN  0137200
BEGIN  0137300
BEGIN  0137400
BEGIN  0137500
BEGIN  0137600
BEGIN  0137700
BEGIN  0137800
BEGIN  0137900
BEGIN  0138000
BEGIN  0138100
BEGIN  0138200
BEGIN  0138300
BEGIN  0138400
BEGIN  0138500
BEGIN  0138600
BEGIN  0138700
BEGIN  0138800
BEGIN  0138900
BEGIN  0139000
BEGIN  0139100
BEGIN  0139200
BEGIN  0139300
BEGIN  0139400
BEGIN  0139500
BEGIN  0139600
BEGIN  0139700
BEGIN  0139800
BEGIN  0139900
BEGIN  014000
BEGIN  014010
ULA 67 (VPFS 03.00)
OUTVACK;

END:

COMMENT************************************************************;
J:=CALLINGPROCESS,W;
IF CALLINGPROCESS,INPUT,FIRST IS TASKBLOCK THEN
TASKFOUND:=CALLINGPROCESS,INPUT,FIRST OR TASKBLOCK ELSE
TASKFOUND:=NONE;
K:=0;
IF TASKFOUND == NONE THEN GOTO NOTASK;
END:(TASKFOUND,TASK(2) NE AND TASKFOUND == NONE AND TASKFOUND,SUC IS TASKBLOCK) DO
BEGIN
TASKFOUND:=TASKFOUND,SUC OR TASKBLOCK;
K:=*1;
END:

IF TASKFOUND == NONE THEN
BEGIN
TASKFOUND:=OUT;
IF FLAG THEN
BEGIN
IF FLAG THEN WRITE"SEEK IS POS FOR PROCESS " ,CALLINGPROCESS,T,
CALLINGPROCESS,Pi;
OUTIMAGE:
END;
END:
ELSE
BEGIN
WRITE"SEEK IS NEG FOR PROCESS " ,CALLINGPROCESS,T,
CALLINGPROCESS,Pi;
OUTIMAGE;
END;
END:

BEGIN
WRITE"CALL TO BLOCK BY PROCESS " ,CALLINGPROCESS,T,
CALLINGPROCESS,Pi;
OUTV("ON CPU NO. ";,WCPU,NUMBER);
OUTV("AT RUNTIME ";,TIME);
OUTIMAGE;
END:

BEGIN
WRITE"CALL TO BLOCK IS POS BY PROCESS " ,CALLINGPROCESS,T,
CALLINGPROCESS,Pi;
OUTIMAGE;
END:
GO TO FOUND;
ELSE
BEGIN
WRITE"CALL TO BLOCK IS NEG FOR PROCESS " ,CALLINGPROCESS,T,
CALLINGPROCESS,Pi;
OUTIMAGE;
END:
FOR I:=1 STEP 1 UNTIL 7 DO
BEGIN
CALLINGPROCESS,Pi[*24]:=WCPU,0(I);
COMMENT"REST GRS IN PD 24=31"**;
CALLINGPROCESS,Pd[*]:=4;
CALLINGPROCESS,CPC=*9;=J;
WCPU,CCS:=0;
END;
BEGIN
END:

GO TO FUND;
BEGIN
WRITE"CALL TO BLOCK IS POS BY PROCESS " ,CALLINGPROCESS,T,
CALLINGPROCESS,Pi;
OUTIMAGE;
END:
FOR I:=1 STEP 1 UNTIL 7 DO
BEGIN
CALLINGPROCESS,Pi[*24]:=WCPU,0(I);
COMMENT"REST GRS IN PD 24=31"**;
CALLINGPROCESS,Pd[*]:=4;
CALLINGPROCESS,CPC=*9;=J;
WCPU,CCS:=0;
END;
BEGIN
END:

GO TO FUND;
BEGIN
WRITE"CALL TO BLOCK IS NEG FOR PROCESS " ,CALLINGPROCESS,T,
CALLINGPROCESS,Pi;
OUTIMAGE;
END:
FOR I:=1 STEP 1 UNTIL 7 DO
BEGIN
CALLINGPROCESS,Pi[*24]:=WCPU,0(I);
COMMENT"REST GRS IN PD 24=31"**;
CALLINGPROCESS,Pd[*]:=4;
CALLINGPROCESS,CPC=*9;=J;
WCPU,CCS:=0;
END;
BEGIN
END:

GO TO FUND;
BEGIN
WRITE"CALL TO BLOCK IS POS BY PROCESS " ,CALLINGPROCESS,T,
CALLINGPROCESS,Pi;
OUTIMAGE;
END:
FOR I:=1 STEP 1 UNTIL 7 DO
BEGIN
CALLINGPROCESS,Pi[*24]:=WCPU,0(I);
COMMENT"REST GRS IN PD 24=31"**;
CALLINGPROCESS,Pd[*]:=4;
CALLINGPROCESS,CPC=*9;=J;
WCPU,CCS:=0;
END;
BEGIN
END:

GO TO FUND;
BEGIN
WRITE"CALL TO BLOCK IS NEG FOR PROCESS " ,CALLINGPROCESS,T,
CALLINGPROCESS,Pi;
OUTIMAGE;
END:
FOR I:=1 STEP 1 UNTIL 7 DO
BEGIN
CALLINGPROCESS,Pi[*24]:=WCPU,0(I);
COMMENT"REST GRS IN PD 24=31"**;
CALLINGPROCESS,Pd[*]:=4;
CALLINGPROCESS,CPC=*9;=J;
WCPU,CCS:=0;
END;
BEGIN
END:

GO TO FUND;
BEGIN
WRITE"CALL TO BLOCK IS POS BY PROCESS " ,CALLINGPROCESS,T,
CALLINGPROCESS,Pi;
OUTIMAGE;
END:
FOR I:=1 STEP 1 UNTIL 7 DO
BEGIN
CALLINGPROCESS,Pi[*24]:=WCPU,0(I);
COMMENT"REST GRS IN PD 24=31"**;
CALLINGPROCESS,Pd[*]:=4;
CALLINGPROCESS,CPC=*9;=J;
WCPU,CCS:=0;
END;
BEGIN
END:

GO TO FUND;
BEGIN
WRITE"CALL TO BLOCK IS NEG FOR PROCESS " ,CALLINGPROCESS,T,
CALLINGPROCESS,Pi;
OUTIMAGE;
END:
FOR I:=1 STEP 1 UNTIL 7 DO
BEGIN
CALLINGPROCESS,Pi[*24]:=WCPU,0(I);
COMMENT"REST GRS IN PD 24=31"**;
CALLINGPROCESS,Pd[*]:=4;
CALLINGPROCESS,CPC=*9;=J;
WCPU,CCS:=0;
END;
BEGIN
END:
UPDATE 67 (VERS 04.00)

PLMCKING:

START TIME := TIME;

HOLD(12,4);

COMMENT**EXIT FROM TIME FOR NO TASK WHEN A TASK IS

FOUND (57,2-12,13=34.4) MUST BE ADDED****;

PROD(CALLINGPROCESS);

COMMENT********************;

IF FLAG THEN

BEGIN

01553000

01555000

01557000

01559000

01561000

01563000

01565000

01567000

END:

BEGIN

01568000

01569000

01570000

01571000

01572000

01573000

01574000

01575000

IF IF THEN

BEGIN

01576000

01577000

01578000

01579000

01580000

01581000

01582000

01583000

END:

BEGIN

01584000

01585000

01586000

01587000

END:

01588000

01589000

01590000

01591000

END:

01592000

01593000

01594000

01595000

01596000

01597000

01598000

01599000

IF IF THEN

BEGIN

01600000

01601000

01602000

01603000

01604000

01605000

01606000

01607000

END:

01608000

01609000

01610000

01611000

END:

01612000

01613000

01614000

01615000

01616000

01617000

01618000

01619000

END:

01620000

01621000

01622000

01623000

01624000

BEGIN

01625000

01626000

01627000

01628000

01629000

01630000

01631000

01632000

01633000

01634000

01635000

01636000

01637000

01638000

END:

01639000

01640000

01641000

01642000

01643000

01644000

01645000

01646000

01647000

01648000

01649000

END:

WRITE("CALL TO PLMCK BY PROCESS ",CALLINGPROCESS,T);

CALLINGPROCESS,P1);

OUTIMAGE;

END:

01500000

01502000

01504000

01506000

01508000

01510000

01512000

01514000

01516000

01518000

01520000

01522000

01524000

01526000

01528000

01530000

01532000

01534000

01536000

01538000

01540000

01542000

01544000

01546000

01548000

01550000

01552000

01554000

01556000

01558000

01560000

01562000

01564000

01566000

01568000

01570000

01572000

01574000

01576000

01578000

01580000

01582000

01584000

01586000

01588000

01590000

01592000

01594000

01596000

01598000

01600000

01602000

01604000

01606000

01608000

01610000

01612000

01614000

01616000

01618000

01620000

01622000

01624000

01626000

01628000

01630000

01632000

01634000

01636000

01638000

01640000

01642000

01644000

01646000

01648000

01650000
AFTER CURRENT
PROCCESS: G(1) := 0;
BEGIN
  IF LPAO FIRST /= NONE AND INTIP.LPA /= |
      (THIS PROCESSALLOCATOR) THEN ACTIVATE LPAO,FIRST
  END;
  PASSIVATE;
  GOTO PASTA1;
END;

SELF:;
STARTTIME := TIME;
MODEL({1,8});
COMMENTS:="ENTRANT ACTIVITY TIME**

END;

FOR I = 1, 0
SYNCHRONIZED

END;

DESCRIPTION: SIMULATES THE ARRIVAL OF A CLOCK

FUNCTION: CREATES CPU'S IN SYSTEM AND BACKGROUND

PROCESSES RUNNING ON THEM, GENERATE A CLOCK

INTERRUPT EVERY 10 USEC AND ALERTS INTERRUPT

TRIPPLES

BEGIN

PROCCESS CLASS CLOCK INTERRUPT;
BEGIN
  ref(CPU) := ARRAY C(1..NUM);
  FOR I := 1 STEP I UNTIL NUM DO
    ref(CPU) := ARRAY C(1..NUM);
  BEGIN
    newCPU(1); 
    newCPU(1); 
    newCPU(1); 
    newCPU(1); 
    newCPU(1); 
    newCPU(1); 
    newCPU(1); 
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1);
    newCPU(1)

END OF PROCESSALLOCATOR.
LA 67 (VENS QA/JO)

FUNCTION: TC INITIATOR TO INITIATE PROCESS

* VARIABLES: NCNE * ACTIVATED BY: NONE

AP CLASS INITIATOR;
INTEGER I, C := 0;
BEGIN
IF C = 1 THEN GO TO E1 ELSE
FOR I := 1 STEP 1 UNTIL NUMOFPURSES DO
BEGIN
G(I) := I;
END;
C:=C+1;
GO TO A;
END;

FUNCTION: OCCUPY A CPU WHEN IDLE TO SIMULATE BACKGROUND

* VARIABLES: NONE * ACTIVATED BY: NONE

AP CLASS BACKGROUND;
BEGIN
HLD: HOLD(SIMPERIOD-TIME);
IF TIME < SIMPERIOD THEN GOTO HLD ELSE
BEGIN
RELEVANTCPU.STARTTIME := RELEVANTCPU.STARTTIME + TIME;
END;
PASSIVATE;
END;

FUNCTION: DELAYSTAT

* DESCRIPTION: CALCULATES AND REPORTS BASIC STATISTICS ON DELAYS OF INTEREST FOR CALLS TO EACH OF THE SYSTEM X FAMILIES OF EXCHANGES

* VARIABLES:

PROCEDURES:

INPUT TO:

OUTPUT FWCM:

-19-
**CLASS DELAYSTAT(TITLE);**
**VALUE TITLE; TEXT TITLE;**

BEGIN
  INTEGER CHS;
  REAL X;
BEGIN
  OUTFILE;
  OUTTEXT("DELAY STATISTICS FOR RESPONSE TIME ");
  OUTLINE(TITLE);
  OUTVR("maximum delay in microseconds = ", MAX);
  OUTVR("average delay in microseconds = ", (SUM/OBS));
END**;

**PROCEDURE REPORT;**
BEGIN
  OUTLINE("***** ERROR ***** NOT AN OPEN OR CLOSE REQUEST TO STORAGE ALLOCATOR");
  OUTLINE("RECEIVED * * * ** *");
END**;

**PROCEDURE UPDATE(X): REAL X;**
BEGIN
  CHS := CHS + 1;
  SUM := SUM + X;
  IF OBS = 1 THEN MIN := MAX := X
  ELSE IF X < MIN THEN MIN := X
  ELSE IF X > MAX THEN MAX := X;
END**;

**PROCEDURE UPDATE(X): REAL X;**
BEGIN
  OUTFILE;
  OUTTEXT("DELAY STATISTICS FOR RESPONSE TIME ");
  OUTLINE(TITLE);
  OUTVR(11 NAX!OIUN UELAY IN 
  OUTVR(11 1UNIIIUII DELAY IN 
  OUTVR(11 AV"hAGE DELAY IN 
  OUTVR(11 MAX );
END**;

**DESCRIPTION:**
A SUITE OF PROCESS INSTANCES USED FOR THE EVALUATION OF A PROPOSED NEW CALL TO THE PROCESS ALLOCATOR - THE LOCK<P> 0 - FOR RELEASE 2 OF Wvu II ML SYSTEM. THEY ARE ADAPATION OF BASH PROCESS INSTANCES.

**FUNCTION:**
The PROCESS INSTANCES ARE USED TO LOAD THE SYSTEM BY ISSUING REQUESTS TO THE STORAGE ALLOCATOR TO OPEN AND CLOSE THEIR RESPECTIVE FILES.

**VARIABLES:**
- COUNTER TO REGISTER NO. OF TIMES AN INSTANCE TRAVERSES ITS LOOP
- STORAGE ALLOCATOR
- STORAGE ALLOCATOR, INTIM, INITIATOR
- ACTIVATES:
- NONE
- ACTIVATED BY:
- PROCESS ALLOCATOR

**AP CLASS WITHFBLCALL:**
BEGIN
  INTEGRER COUNTER;
  IF FELLOK(5) == THEN GO TO A
  IF RELEVANTCPU.<(2) == THEN GO TO A
  ELSE
    BEGIN
      IF RELEVANTCPU.<(2) == THEN GO TO CLOSEFILE
      ELSE
        BEGIN
          IF RELEVANTCPU.<(2) == THEN GO TO OPENFILE
          ELSE
            BEGIN
              OUTLINE("***** ERROR ***** NOT AN OPEN OR CLOSE FILE MESSAGE RECEIVED");
              END**;
              CF A TASK BEING RECEIVED *****;
              CLOSFILE:
              HOLD(579,0);
              G(0) := 1;
              G(1) := P1;
              G(2) := S1;
              HAND;
              COMMENT ***** CLOSE FILE REQUEST TO SA ";
              HOLD(459,0);
              COUNTER := COUNTER + 1;
              IF FLAG THEN OUTVR("COUNTER OF WITHFBLCALL = ", COUNTER);
              GO TO A;
              OPENFILE:
              HOLD(458,0);
              G(0) := 1;
              OUTLINE("***** ERROR ***** NOT AN OPEN OR CLOSE FILE MESSAGE RECEIVED");
              END**;
              CF A TASK BEING RECEIVED *****;
              CLOSFILE:
              HOLD(579,0);
              G(0) := 1;
              G(1) := P1;
              G(2) := S1;
              HAND;
              COMMENT ***** CLOSE FILE REQUEST TO SA ";
              HOLD(459,0);
              COUNTER := COUNTER + 1;
              IF FLAG THEN OUTVR("COUNTER OF WITHFBLCALL = ", COUNTER);
              GO TO A;
              OPENFILE:
              HOLD(458,0);
              G(0) := 1;

A 67 (VHS Un. 10)

FUNCTION
- A SUITE OF PROCESS INSTANCES USED FOR THE EVALUATION
- OF A NEW CALL TO THE PROCESS ALLOCATION SYSTEM
- THEY ARE ADAPTATION OF RASH PROCESS INSTANCES

VARIABLES:
- COUNTER TO REGISTER NO. OF TIMES AN INSTANCE TRAVERSES ITS LOOP
- STORAGE ALLOCATOR
- OUTPUT FROM:
  - STORAGE ALLOCATOR, INTIM, INITIATOR
- ACTIVATES:
  - NONE
  - PROCESS ALLOCATOR

END

BEGIN
  INTEGER COUNTER;
  A;
  FETCH(15);
  IF CC = 0 THEN
    BEGIN
      BLOCK(5); COMMENT*** THE INSTANCE BLOCKS ITSELF WAITING FOR A MESSAGE**;
      GO TO A;
    END
  ELSE
    BEGIN
      IF RELEVANTCPU.G(2) = 4 THEN GO TO CLOSEFILE
      ELSE
        IF RELEVANTCPU.G(2) = 5 THEN GO TO OPENFILE
        ELSE
          BEGIN
            OUTLINE("000 ERROR** NOT AN OPEN OR CLOSE FILE MESSAGE RECEIVED***");
            GO TO A;
          END
    END
BEGIN
  IF RELEVANTCPU.G(2) = 4 THEN GO TO CLOSEFILE
  ELSE
    IF RELEVANTCPU.G(2) = 5 THEN GO TO OPENFILE
    ELSE
      BEGIN
        OUTLINE("***ERROR*** NOT AN OPEN OR CLOSE FILE MESSAGE RECEIVED***");
        GO TO A;
      END
END

AP CLASS NFBLCALL:
BEGIN
  INTEGER COUNTER;
  A;
  FETCH(15);
  IF CC = 0 THEN
    BEGIN
      BLOCK(5); COMMENT*** THE INSTANCE BLOCKS ITSELF WAITING FOR A MESSAGE**;
      GO TO A;
    END
  ELSE
    BEGIN
      IF RELEVANTCPU.G(2) = 4 THEN GO TO CLOSEFILE
      ELSE
        IF RELEVANTCPU.G(2) = 5 THEN GO TO OPENFILE
        ELSE
          BEGIN
            OUTLINE("000 ERROR** NOT AN OPEN OR CLOSE FILE MESSAGE RECEIVED***");
            GO TO A;
          END
    END
BEGIN
  IF RELEVANTCPU.G(2) = 4 THEN GO TO CLOSEFILE
  ELSE
    IF RELEVANTCPU.G(2) = 5 THEN GO TO OPENFILE
    ELSE
      BEGIN
        OUTLINE("***ERROR*** NOT AN OPEN OR CLOSE FILE MESSAGE RECEIVED***");
        GO TO A;
      END
END

FUNCTION
- A SUITE OF PROCESS INSTANCES USED FOR THE EVALUATION
- OF A NEW CALL TO THE PROCESS ALLOCATION SYSTEM
- THEY ARE ADAPTATION OF RASH PROCESS INSTANCES

VARIABLES:
- COUNTER TO REGISTER NO. OF TIMES AN INSTANCE TRAVERSES ITS LOOP
- STORAGE ALLOCATOR
- OUTPUT FROM:
  - STORAGE ALLOCATOR, INTIM, INITIATOR
- ACTIVATES:
  - NONE
  - PROCESS ALLOCATOR

END

BEGIN
  INTEGER COUNTER;
  A;
  FETCH(15);
  IF CC = 0 THEN
    BEGIN
      BLOCK(5); COMMENT*** THE INSTANCE BLOCKS ITSELF WAITING FOR A MESSAGE**;
      GO TO A;
    END
  ELSE
    BEGIN
      IF RELEVANTCPU.G(2) = 4 THEN GO TO CLOSEFILE
      ELSE
        IF RELEVANTCPU.G(2) = 5 THEN GO TO OPENFILE
        ELSE
          BEGIN
            OUTLINE("000 ERROR** NOT AN OPEN OR CLOSE FILE MESSAGE RECEIVED***");
            GO TO A;
          END
    END
BEGIN
  IF RELEVANTCPU.G(2) = 4 THEN GO TO CLOSEFILE
  ELSE
    IF RELEVANTCPU.G(2) = 5 THEN GO TO OPENFILE
    ELSE
      BEGIN
        OUTLINE("***ERROR*** NOT AN OPEN OR CLOSE FILE MESSAGE RECEIVED***");
        GO TO A;
      END
END

AP CLASS NFBLCALL:
BEGIN
  INTEGER COUNTER;
  A;
  FETCH(15);
  IF CC = 0 THEN
    BEGIN
      BLOCK(5); COMMENT*** THE INSTANCE BLOCKS ITSELF WAITING FOR A MESSAGE**;
      GO TO A;
    END
  ELSE
    BEGIN
      IF RELEVANTCPU.G(2) = 4 THEN GO TO CLOSEFILE
      ELSE
        IF RELEVANTCPU.G(2) = 5 THEN GO TO OPENFILE
        ELSE
          BEGIN
            OUTLINE("000 ERROR** NOT AN OPEN OR CLOSE FILE MESSAGE RECEIVED***");
            GO TO A;
          END
    END
BEGIN
  IF RELEVANTCPU.G(2) = 4 THEN GO TO CLOSEFILE
  ELSE
    IF RELEVANTCPU.G(2) = 5 THEN GO TO OPENFILE
    ELSE
      BEGIN
        OUTLINE("***ERROR*** NOT AN OPEN OR CLOSE FILE MESSAGE RECEIVED***");
        GO TO A;
      END
END
CALL THE CALL ARRIVAL AND WHEN SUBSCRIBER CLARS 02034000
   U/G/W 02035000
   SLIGHT THE STATE OF THE GENERATED CALL. 02040000
   GENERATES AN O/G SIS H/W INSTANCE IF STATE OF CALL > 1 02042000
   INDICATES TO O/G SIS H/W ASSOCIATED WITH THE CALL 02043000
   WHEN THE SUBSCRIBER ANSWERS. 02044000
VARIABLES:
   SOC TO INDICATE 1 OUT OF 3 STATES FOR THIS CALL 02045000
   MYINCSISHW A REFER TO THE CORRESPONDING I/C 02047000
   SIS H/W 02048000
   MYOUTSISHW A REFER TO CORRESPONDING O/G SIS H/W. 02049000
   CIRCUITNUM REFERS TO THE CALL CIRCUIT NUMBER I.E. I/C CTT. 02050000
   OUTCCTNUM O/G CTT. UP/CORESS 02052000
   INPUT TC I/C SIS H/W.CALL GENERATOR 02053000
   OUTPUT TC I/C SIS H/W, O/G SIS H/W. 02054000
   ACTIVATES I/C SIS H/W, O/G SIS H/W 02055000
   ACTIVATION BY 1/C SIS H/W, O/G SIS H/W 02056000
   A 02057000

******************************************************************************

PROCESS CLASS CALLRECORD (CIRCUITNUM);
COMMENT================================== 
INTERN CIRCUITNUM;
BEGIN
  02060000 02061000

  INTEGER CIRCUITNUM K, SOC;
  02062000 0123
  BEGIN
    02063000 02064000

    BEGIN
      02065000 02066000

    BEGIN
      02067000 02068000

      BEGIN
        02069000 02070000

      BEGIN
        02071000 02072000

      BEGIN
        02073000 02074000

      BEGIN
        02075000 02076000

      BEGIN
        02077000 02078000

      BEGIN
        02079000 02080000

      BEGIN
        02081000 02082000

      BEGIN
        02083000 02084000

      BEGIN
        02085000 02086000

      BEGIN
        02087000 02088000

      BEGIN
        02089000 02090000

      BEGIN
        02091000 02092000

      BEGIN
        02093000 02094000

      BEGIN
        02095000 02096000

      BEGIN
        02097000 02098000

      BEGIN
        02099000 02090000

      BEGIN
        02091000 02092000

      BEGIN
        02093000 02094000

      BEGIN
        02095000 02096000

      BEGIN
        02097000 02098000

      BEGIN
        02099000 02090000

      BEGIN
        02100000 02101000

      BEGIN
        02102000 02103000

      BEGIN
        02104000 02105000

      BEGIN
        02106000 02107000

      BEGIN
        02108000 02109000

      BEGIN
        02110000 02111000

      BEGIN
        02112000 02113000

      BEGIN
        02114000 02115000

      BEGIN
        02116000 02117000

      BEGIN
        02118000 02119000

      BEGIN
        02120000 02121000

      BEGIN
        02122000 02123000

      BEGIN
        02124000 02125000

      BEGIN
        02126000 02127000

      BEGIN
        02128000 02129000

      BEGIN
        02130000 02131000

      BEGIN
        02132000 02133000

      BEGIN
        02134000 02135000

      COMMENT************ CALL GENERATOR *************

      DESCRIPTION:
      GENERATES TELEPHONE CALLS ACCORDING TO A SPECIFIC
      D TRAFFIC CALLS INTER-ARRIVAL TIMES ARE "WE EXACT
      ENTALLY DISTRIBUTED. 02124000

      FUNCTION:
      ALLOCATRS RANDOLY THE NEWLY GENERATED CALL TO
      A FREE INCOMING CIRCUIT 02127000
      CREATES NEW CALL RECORD AND INCOMING SIS HARD
      WARE PROCESS INSTANCES FOR THE NEW CALL 02129000

      VARIABLES:
      X A REFER TO THE CREATED INCISHW INSTANCE 02131000
      I AN INTEGER VARIABLE 02132000

      OUTPUT TO:CALLRECORD 02134000
/**
 * ACTIVITY: CALLRECORD
 * *************** COMMENT ENDS ***************;
 */

PROCEDURE CALLGENERATOR;
BEGIN
  INTEGER i, COUNT;
  REAL INTERVAL;
  REAL INCISHW;
  LOOP:
    AGAIN: i := RANDOM(999999999, MAXOUTCCT, SD15);
    IF SHWINC(i) <= SOME THEN GO TO AGAIN;
    CALL(i) := NEW CALLRECORD(i);
    SISISHW := NEW INCISHW(i);
    COMMENT: WRITE CALLRECORD AND INCISHW FOR NEW CALL;
    CALL(i), WINCISHW := SHWINC(i);
    SHWINC(i) := CALL(i);
    ACTIVATE CALL(i) AFTER CURRENT;
    END := COUNT + 1;
    IF COUNT LT 10 THEN
      BEGIN
        INTERVAL := NEGL(KN 0.000000033, SD2);
        HOLD INTERVAL;
        COMMENT: CALL INTERVAL ARRIVAL TIME FOR 5000 TRAFFIC;
        END;
    END
    COMMENT: 10 CALLS ARE GENERATED AT START OF THE RUN ****;
    GO TO LOOP;
  END
  END
  END

COMMENT:*********************** I/C SIS H/W PROCESS ***********************;

DESCRIPTION:
* EVERY CALL IN PROGRESS HAS ITS OWN I/C SIS H/W PROCESS
* WHICH SENDS AND RECEIVES INFORMATION CONCERNING THE
* CALL TO THE I/C SIS S/W.
* FUNCTION:
* SENDS SEIZURE MESSAGE TO I/C SIS S/W
* SENSORS DIGITS DiaLED TO I/C SIS S/W
* SENDS "SUBS CLEAR DOWN" MESSAGE TO I/C SIS S/W
* VARIABLES:
* ANSWER TO INDICATE "ANSWEX" MESSAGE FROM I/C SIS S/W
* SEIZURE TO INDICATE CALL ARRIVAL
* SUBSCRIBED "SUBS, CLEAR DOWN" MESSAGE
* MYCALLREC REFERS TO CORRESPONDING CALL RECORD
* CCTFREE FROM I/C SIS S/W
* CCTNUM CIRCUIT NUMBER IDENTITY OF I/C CAll
* OUTCONNUM THE CORRESPONDING C/G Cct. NO.
* SOC TO INDICATE THE STATE OF CALL GENERATED
* RESPONSE: AN ARRAY TO STORE DELAYS OF INTEREST FOR THIS CALL
* TIMEOUT TIME DURING WHICH A RESPONSE IS EXPECTED BACK
* PROCEDURES:
* MYVIS RETURNs A REF. TO THE SIS INSTANCE IN CHARGE OF THE CALL
* INPUT TO:
* I/C SIS S/W
* OUTPUT FROM:
* I/C SIS S/W, CALLRECORD
* ACTIVATES
* CALLRECORD
* ACTIVATED BY:
* I/C SIS S/W, CALLRECORD
* *************** COMMENT ENDS ***************;

PROCEDURE CLASS INCISHW(CCTNUM): INTEGER CCTNUM;
BEGIN
  INTEGER SOC, 1, OUTCONCTNUM;
  BOOLEAN ANSWER, SUBSCRIBED, CCTFREE, SEIZURE;
  REF(CALLRECORD) MYCALLREC;
  REAL ARRAY RESPONSETIME(1: 8);
  REAL TIMEOUT;
  REF(SISSW) PROCEDURE MYVIS;
  MYVIS := POLY + SISRECI(CCTNUM)) OR SISSW;
  MYCALLREC := CALL(CCTNUM);
  OUTTRY := "WRSE THIS I/C SIS W/H ITS SD, QC, = ", SOC); 
  OUTTRY := "SEIS THIS I/C Cct. NO. IS ", CCTNUM);
  IF SOC = 1 THEN GO TO A
  ELSE
    IF SOC = 2 THEN GO TO B
  ELSE
    IF SOC = 3 THEN GO TO C
  ELSE
    BEGIN
      OUTLINE("??? AN UNALLOWED SOC GENERATED ???");
      OUTTRY("??? THE SOC VALUE IS ", SOC);
      END
      COMMENT: SOC IN ERROR;
      END
      IF NOT SEIZURE THEN
      BEGIN
        OUTLINE("??? I/C SIS H/W ACTIVATED WITHOUT 'SEIZURE' MESSAGE ???");
        OUTLINE("??? THIS IS A FATAL ERROR='RUN ABORTED ???");
        OUTLINE("??? NUMBER OF CALL GENERATOR PROCESS *****");
        END
        END
      END
      END
      END
      END
      END
BEGIN
GO TO 11;
END

LND** OR FALSE SEIZURE MESSAGE ***;
MYSL.WISSAGC1,1):=1;
MYSL.WISSAGC1,2):=CCTNUM;
COMMENT**SEIZURE MESSAGE AND I/C CCT NO***;
MYSL.WISSAGC1,2):=CCTNUM;
COMMENT**FAUSE BEFORE SENDING FIRST DIGIT**;
MYSL.WISSAGC1,1):=1;
MYSL.WISSAGC1,2):=CCTNUM;
COMMENT**1ST Digit AND I/C CCT NO. TO I/C SIS S/W REP**;
MYSL.WISSAGC1,2):=CCTNUM;
COMMENT**J80 Digit AND I/C CCT NO. TO I/C SIS S/W REP**;
MYSL.WISSAGC1,2):=CCTNUM;
COMMENT**INTER-DIGITAL PAUSE**;
MYSL.WISSAGC1,1):=1;
MYSL.WISSAGC1,2):=CCTNUM;
IF MYCALLREC.WOUTISH =/= NONE THEN
MYSL.WISSAGC1,3):=MYCALLREC.WOUTISHW.CCTNUM;
COMMENT**3RD Digit AND I/C CCT NO.***;
RESPONSETIME(1) := TIME;
COMMENT**FIRST RESPONSE 1 MEASUREMENT***;
HOLD(10000.00);
COMMENT**INTER-DIGITAL PAUSE**;
MYSL.WISSAGC1,1):=1;
MYSL.WISSAGC1,2):=CCTNUM;
OUTLINE "$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$";
GUITTY "$S THIS I/C SIS S/W NO IS ".CCTNUM);
MYSL.WISSAGC1,3):=MYCALLREC.WOUTISHW.CCTNUM;
COMMENT**4TH Digit AND I/C CCT NO.***;
RESPONSETIME(1) := TIME;
COMMENT**START RESPONSE3 MEASUREMENT***;
HOLD(10000.00);
COMMENT**INTER-DIGITAL PAUSE**;
MYSL.WISSAGC1,1):=1;
MYSL.WISSAGC1,2):=CCTNUM;
MYSL.WISSAGC1,3):=MYCALLREC.WOUTISHW.CCTNUM;
COMMENT**5TH Digit AND I/C CCT NO.***;
HOLD(10000.00);
COMMENT**INTER-DIGITAL PAUSE**;
MYSL.WISSAGC1,1):=1;
MYSL.WISSAGC1,2):=CCTNUM;
MYSL.WISSAGC1,3):=MYCALLREC.WOUTISHW.CCTNUM;
COMMENT**6TH Digit AND I/C CCT NO.***;
HOLD(10000.00);
COMMENT**INTER-DIGITAL PAUSE**;
MYSL.WISSAGC1,1):=1;
MYSL.WISSAGC1,2):=CCTNUM;
MYSL.WISSAGC1,3):=MYCALLREC.WOUTISHW.CCTNUM;
COMMENT**7TH Digit AND I/C CCT NO.***;
HOLD(10000.00);
COMMENT**INTER-DIGITAL PAUSE**;
MYSL.WISSAGC1,1):=1;
MYSL.WISSAGC1,2):=CCTNUM;
MYSL.WISSAGC1,3):=MYCALLREC.WOUTISHW.CCTNUM;
COMMENT**8TH Digit AND I/C CCT NO.***;
PASSIVATE;
GO TO LASTACTION;
BACK: PASSIVATE;
BC:
IF NOT ANSWER THEN
BEGIN
ERROR 22); GO TO BACK;
COMMENT**WAITING FOR PROPER ANSWER MESSAGE***;
AND**FALSE ANSWER**;
TIMEOUT := UNIFORM(0., 60000.0, SD10);
HOLD(TIMEOUT);
COMMENT**TIME TO CANCEL 1/C SIS S/W REP T/O***;
MYSL.WISSAGC10,1):=1;
MYSL.WISSAGC10,2):=CCTNUM;
MYSL.WISSAGC10,3):=MYCALLREC.WOUTISHW.CCTNUM;
COMMENT**CANCEL T/O***;
ACTIVE CALL(CCTNUM) AFTER CURRENT;
GO TO LASTACTION;
PASSIVE: PASSIVATE;
C:
IF NOT SUBSCRIBED THEN
BEGIN
ERROR 23); GO TO PASSIVE;
COMMENT**WAITING FOR TRUE MESSAGE***;
MYSL.WISSAGC11,1):=1;
MYSL.WISSAGC11,2):=CCTNUM;
MYSL.WISSAGC11,3):=MYCALLREC.WOUTISHW.CCTNUM;
COMMENT**FULL MESSAGE TO I/C SIS S/W REP***;
RESPONSETIME(7) := RESPONSETIME(8) := TIME;
PASSIVE;
IF NOT CANCELL THEN
BEGIN
ERROR 24); GO TO Q1
END ** FALSE "CCTFREL" MESSAGE***;
02232000 02232700
167 (YKB ObJ)

```c

ACTIVATE(A)(CXTNUM) AFTER CURRENT;
LASTACTION;
SISHWIN(CXTNUM):=NUNE;
MCAII=I IE: NUNE;
END ** OF I/C SIS H/W PROCESS **;

COMMENT************************ SIGNAL INTERWORKING SUBSYSTEM (SIS) *******
• THE SIS MODEL LISTED BELOW HANDLES BOTH THE INCOMING AND
• OUTGOING CIRCUITS, HOWEVER, FOR CERTAIN APPLICATIONS THERE
• IS A NEED TO SEPARATE THESE TWO FUNCTIONS, THE FOLLOWING
• NOTIONALLY DESCRIBES THE I/C AND O/G PARTS OF SIS IN
• DETAIL, THE SIS LISTING IS FOR THE COMPLETE SIS.

******************************* I/C SIS S/W ***********************

******************** O/G SIS S/W ******************************

AP CLASS SISW;

BEGIN
WHILE AND2(MESSAGE1).I=0, I<25) DO I=I+1;
IF MESSAGE1(I,1) = 1 THEN
BEGIN
OUTFILE:"**A H/W MESSAGE FROM AN.SIS H/W IS FOUND";
OUTTTY:"**THE MESSAGE INDEX IS = "I";
OUTTTY:"**RELATED TO I/C CTT. NO. **MESSAGE(I,2)");
IF MESSAGE1(I,2) = 0 THEN
OUTTTY:"**FOUR H/W MESSAGE RECEIVED ???";
GO TO JUMP(I);
END
```

ELSE
HELP1 = HOLD(1234,0);
COMMENT ** PROCESSING TIME **;
GO TO A;
END;
T1: HOLD(1234,0);
COMMENT ** PROCESSING TIME **;
=0; T2: HOLD(1365,0);
COMMENT ** PROCESSING TIME **;
1=RELEVANTCPU(1,1);
X=SISHWINCT(1);
X,ANSWER:=TRUE;
ACTIVATE X AFTER CURRENT;
GO TO A; COMMENT**"ANSWER" TO 1/C SIS H/W **;
T3: HOLD(1467,0);
COMMENT ** PROCESSING TIME **;
1=RELEVANTCPU(1,1);
X=SISHWINCT(1);
X.CURRENT:=TRUE;
ACTIVATE X AFTER CURRENT;
GO TO A;
COMMENT ** "CCTFREE" TO 1/C SIS H/W **;
M1: MESSAGE(1,1):=0;
COMMENT ** PROCESSING TIME **;
INCCT := MESSAGE(1,1);
G(0):=2*CPSREM(INCCT);
G(1):=INCCT;
G(3):=INCCT;
BAND;
COMMENT ** "IA(M SZ)" TO 1/C CPS REP **;
GO TO D;
M2: MESSAGE(2,1):=0;
COMMENT ** PROCESSING TIME **;
HOLD(1531,0);
COMMENT ** PROCESSING TIME **;
INCCT:=MESSAGE(2,2);
G(0):=2*CPSREM(INCCT);
G(1):=2;
G(3):=INCCT;
BAND;
COMMENT ** SAM1 T/O 1/C CPS REP **;
GO TO D;
M3: MESSAGE(3,1):=0;
COMMENT ** PROCESSING TIME **;
HOLD(1517,0);
COMMENT ** PROCESSING TIME **;
INCCT:=MESSAGE(3,2);
G(0):=2*CPSREM(INCCT);
G(3):=INCCT;
BAND;
COMMENT ** SAM2 T/O 1/C CPS REP **;
GO TO D;
M4: MESSAGE(4,1):=0;
COMMENT ** PROCESSING TIME **;
HOLD(1517,0);
COMMENT ** PROCESSING TIME **;
INCCT:=MESSAGE(4,2);
G(0):=2*CPSREM(INCCT);
G(1):=2;
G(3):=INCCT;
BAND;
COMMENT ** SAM3 TO 1/C CPS REP **;
GO TO D;
M5: MESSAGE(5,1):=0;
COMMENT ** PROCESSING TIME **;
HOLD(1517,0);
COMMENT ** PROCESSING TIME **;
INCCT:=MESSAGE(5,2);
G(0):=2*CPSREM(INCCT);
G(1):=2;
G(3):=INCCT;
BAND;
COMMENT ** SAM4 TO 1/C CPS REP **;
GO TO D;
M6: MESSAGE(6,1):=0;
COMMENT ** PROCESSING TIME **;
HOLD(1517,0);
COMMENT ** PROCESSING TIME **;
INCCT:=MESSAGE(6,2);
G(0):=2*CPSREM(INCCT);
G(1):=2;
G(3):=INCCT;
BAND;
COMMENT ** SAM5 TO 1/C CPS REP **;
GO TO D;
M7: MESSAGE(7,1):=0;
COMMENT ** PROCESSING TIME **;
HOLD(1517,0);
COMMENT ** PROCESSING TIME **;
INCCT:=MESSAGE(7,2);
G(0):=2*CPSREM(INCCT);
COMMENT ** PROCESSING TIME **
Y:=SISHWINC(INCCT);
G(I):=2*CPSREP(INCCT);
G(1):=10;
G(J):=INCCT;
G(4):=X.OUTGCTCTNUM;
HAND;
COMMENT ** SAW6 T/O I/C CPS REP **;
GO TO D;
M1: MESSAGE(21,1):=0;
INCCT:=MESSAGE(21,1);
HOLD(1517,0);
COMMENT ** PROCESSING TIME **;
Y:=SISHWINC(INCCT);
G(I):=2*CPSREP(INCCT);
G(1):=10;
G(J):=INCCT;
G(4):=X.OUTGCTCTNUM;
HAND;
COMMENT ** SAW7 T/O I/C CPS REP **;
GO TO D;
M1: MESSAGE(4,1):=0;
INCCT:=MESSAGE(4,1);
HOLD(1517,0);
COMMENT ** PROCESSING TIME **;
Y:=SISHWINC(INCCT);
G(I):=2*CPSREP(INCCT);
G(1):=10;
G(J):=INCCT;
G(4):=X.OUTGCTCTNUM;
HAND;
COMMENT ** SAW8 T/O I/C CPS REP **;
GO TO D;
M1: MESSAGE(10,1):=0;
HOLD(1265,0);
COMMENT ** PROCESSING TIME **;
GO TO D;
M1: MESSAGE(12,1):=0;
INCCT:=MESSAGE(12,1);
HOLD(1517,0);
COMMENT ** PROCESSING TIME **;
COMMENT ** "METER OVER JUNCTION" SIGNAL IS NOT SIMULATED **;
G(I):=2*CPSREP(INCCT);
G(I):=14;
G(J):=MESSAGE(11,2);
G(4):=SISHWINC(INCCT).OUTGCTCTNUM;
HAND;
COMMENT ** "RELEASE" MESSAGE TO I/C CPS **;
GO TO D;
T4: HOLD(3410,0);
COMMENT ** PROCESSING TIME **;
I:=RELEVANTCPU.G(4);
Y:=SISHWINCUTI(1);
RESPONSE(3,2,1):=UPDATE TIME - SISHWINC RELEVANTCPU.G(3), RESPONSE TIME(1);
COMMENT ** END RESPONSE **;
Y:=MESSAGE(11,2);
Y:=MESSAGE(11,2):=RELEVANTCPU.G(3);
COMMENT ** SEIZE Y MESSAGE TO O/G SIS H/W **;
ACTIVATE Y AFTER CURRENT;
GO TO A;
T5: HOLD(2297,0);
COMMENT ** PROCESSING TIME **;
I:=RELEVANTCPU.G(4);
Y:=SISHWINCUTI(1);
Y:=MESSAGE(3,3):=1;
COMMENT ** DIGIT2 TO O/G SIS H/W **;
Y:=MESSAGE(3,3):=RELEVANTCPU.G(3);
RESPONSE(3,2,1):=UPDATE TIME - SISHWINC RELEVANTCPU.G(3), RESPONSE TIME(3);
ACTIVATE Y AFTER CURRENT;
GO TO A;
T6: HOLD(2297,0);
COMMENT ** PROCESSING TIME **;
I:=RELEVANTCPU.G(4);
Y:=SISHWINCUTI(1);
Y:=MESSAGE(4,1):=1;
COMMENT ** DIGIT3 TO O/G SIS H/W **;
Y:=MESSAGE(4,2):=RELEVANTCPU.G(3);
ACTIVATE Y AFTER CURRENT;
GO TO A;
T7: HOLD(2297,0);
COMMENT ** PROCESSING TIME **;
I:=RELEVANTCPU.G(4);
Y:=SISHWINCUTI(1);
Y:=MESSAGE(5,1):=1;
COMMENT ** DIGIT 4 TO O/G SIS H/W **;
Y:=MESSAGE(5,2):=RELEVANTCPU.G(3);
ACTIVATE Y AFTER CURRENT;
GO TO A;
T8: HOLD(2297,0);
COMMENT ** PROCESSING TIME **;
I:=RELEVANTCPU.G(4);
Y:=SISHWINCUTI(1);
Y:=MESSAGE(9,1):=1;
Y:=MESSAGE(9,2):=RELEVANTCPU.G(3);
ACTIVATE Y AFTER CURRENT;
GO TO A;
T9: HOLD(2297,0);
I:=RELEVANTCPU.G(4);
Y:=SISHWINCUTI(1);
Y:=MESSAGE(7,1):=1;
COMMENT** DIGIT6 TO O/G SIS H/****;
Y.MESSA(;7,2):=RELEVANTCPU,G(3);
ACTIVATE Y AFTER CURRENT;
GO TO A;
TIU; HOLD(21,0,0);
COMMENT** PROCESSING TIME **;
Y:=RELEVANTCPU,G(4);
Y:=SHINQUT(I);
Y.MESSA(;9,1):=1;
Y.MESSA(;9,2):=RELEVANTCPU,G(3);
PSPNL(1,UPDATE TIME - SHINQUT RELEVANTCPU,G(3),RESPONSETIME(8));
ACTIVATE Y AFTER CURRENT;
GO TO A;
COMMENT** IDLE OR CLEAR FORWARD TO O/G SIS H/****;
GO TO A;
M12: MESSAGE(12,1):=0;
HOLD(1615);
COMMENT** PROCESSING TIME **;
Y:=SHINQUT(MESSAGE(12,3));
Y.MESSA(;2,1):=1;
Y.MESSA(;2,2):=MESSAGE(12,2);
COMMENT** DIGIT1 TO O/G SIS H/****;
ACTIVATE Y AFTER CURRENT;
GO TO D;
M13: MESSAGE(13,1):=0;
HOLD(1813,3);
COMMENT** PROCESSING TIME **;
Y:=SHINQUT(MESSAGE(13,3));
Y.MESSA(;4,1):=1;
Y.MESSA(;4,2):=MESSAGE(13,2);
COMMENT** DIGIT1 TO O/G SIS H/****;
ACTIVATE Y AFTER CURRENT;
GO TO D;
M14: MESSAGE(14,1):=0;
HOLD(1813,3);
COMMENT** PROCESSING TIME **;
Y:=SHINQUT(MESSAGE(14,3));
Y.MESSA(;5,1):=1;
Y.MESSA(;5,2):=MESSAGE(14,2);
COMMENT** DIGIT1 TO O/G SIS H/****;
ACTIVATE Y AFTER CURRENT;
GO TO D;
M15: MESSAGE(15,1):=0;
HOLD(1813,3);
COMMENT** PROCESSING TIME **;
Y:=SHINQUT(MESSAGE(15,3));
Y.MESSA(;6,1):=1;
Y.MESSA(;6,2):=MESSAGE(15,2);
COMMENT** DIGIT1 TO O/G SIS H/****;
ACTIVATE Y AFTER CURRENT;
GO TO D;
M16: MESSAGE(16,1):=0;
HOLD(1813,3);
COMMENT** PROCESSING TIME **;
Y:=SHINQUT(MESSAGE(16,3));
Y.MESSA(;7,1):=1;
Y.MESSA(;7,2):=MESSAGE(16,2);
COMMENT** DIGIT1 TO O/G SIS H/****;
ACTIVATE Y AFTER CURRENT;
GO TO D;
M17: MESSAGE(17,1):=0;
HOLD(1813,3);
COMMENT** PROCESSING TIME **;
Y:=SHINQUT(MESSAGE(17,3));
Y.MESSA(;8,1):=1;
Y.MESSA(;8,2):=MESSAGE(17,2);
COMMENT** DIGIT1 TO O/G SIS H/****;
ACTIVATE Y AFTER CURRENT;
GO TO D;
M18: MESSAGE(18,1):=0;
HOLD(2419,0);
COMMENT** PROCESSING TIME **;
Y:=SHINQUT(MESSAGE(18,3));
Y.MESSA(;9,1):=1;
Y.MESSA(;9,2):=MESSAGE(18,2);
COMMENT** DIGIT1 TO O/G SIS H/****;
ACTIVATE Y AFTER CURRENT;
GO TO D;
M19: MESSAGE(19,1):=0;
HOLD(1236,0);
COMMENT** PROCESSING TIME **;
Y:=SHINQUT(MESSAGE(19,3));
Y.MESSA(;10,1):=1;
Y.MESSA(;10,2):=MESSAGE(19,2);
COMMENT** DIGIT1 TO O/G SIS H/****;
ACTIVATE Y AFTER CURRENT;
GO TO D;
M20: MESSAGE(20,1):=0;
HOLD(1975,0);
COMMENT** PROCESSING TIME **;
Y:=SHINQUT(MESSAGE(20,3));
Y(1):=2;
Y(3):=MESSAGE(20,3);
Y(4):=MESSAGE(20,3);
Y(5):MESSAGE(20,3);
Y(6):MESSAGE(20,3);
COMMENT** CIRCUIT FREE TO O/G CPS REP. **;
GO TO D;
END *** CF SIS *****;
A 67 (VERS UN.00)

* DESCRIPTION:
  RESPONSIBLE FOR THE 1/C CALL PROCESSING

* FUNCTION:
  1/C FILE READ TO STORAGE ALLOCATOR

* VARIABLES:
  WIDTH IS THE NUMBER OF 1/C SIS

* CALL A SWITCH TO SELECT ACTION ACCORDING TO 1/C CK1

* TEST 1/C CK1 VALUE

* OUTPUT REP NO. OF 1/C CPS

* INCT 1/C CIRCUIT NO.

* OUTCT 1/C CIRCUIT NO.

* LALLOC/2/4 TO CALL,"COMMAND"

* X REF TC 1/C SIS H/W

* Y REF TO 0/G SIS H/W

* INPUT TO:
  1/C SIS S/(+,O/G CPS, DSS S/W, 0/G SIS S/W, SA

* OUTPUT FROM:
  1/C SIS S/(+,O/G CPS, DSS S/W, SA

* ACTIVATES:
  NONE

* ACTIVATED BY:
  NONE

*********************************************************************** COMMENT ENDS***********************************************************************

AP CLASS CPSW:

COMMENT ===============;

BEGIN

SWITCH CASE:=T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13;

T4,T5,T6,T7,T8,T9,T10,T11,T12,T13;

INTEGER TSK,OUTREP,OUTCT,INCT,1,U;

REFS SISW(0),INCSREP;

REFOUTS(SISW);

H1 TSK=RELEVANTCPU,G(1);

GO TO CASH(TSK);

A1: FETCH(15);

IF CPU THEN GO TO B

ELSE

BEGIN

H1=0;

GO TO B;

END

END TASK IN I/F O**;

BEGIN

COMMENT** PROCESSING TIME AND RESPONSE TO IAMCSZ**;

GO TO A1;

COMMENT** PROCESSING TIME AND SAM1 RESPONSE**;

GO TO A1;

COMMENT** PROCESSING TIME AND SAM2 RESPONSE**;

GO TO A1;

COMMENT** PROCESSING TIME**;

GO TO A1;

COMMENT** PROCESSING TIME**;

OUTREP := RANDBIN(0, 3, 80);

COMMENT RANDBIN O/G CPS REP. TOTAL NO, OF CPS REPLICATIONS:

G/C CCTS IS TOTALCCTS, CPSREPS IS TOTAL NO. OF CPS REPLICATIONS:

02717000

02718000

02719000

02720000

02721000

02722000

02723000

02724000

02725000

02726000

02727000

02728000

02729000

02730000

02731000

02732000

02733000

02734000

02735000

02736000

02737000

02738000

02739000

02740000

02741000

02742000

02743000

02744000

02745000

02746000

02747000

02748000

02749000

02750000

02751000

02752000

02753000

02754000

02755000

02756000

02757000

02758000

02759000

02760000

02761000

02762000

02763000

02764000

02765000

02766000

02767000

02768000

02769000

02770000

02771000

02772000

02773000

02774000

02775000

02776000

02777000

02778000

02779000

02780000

02781000

02782000

02783000

02784000

02785000

02786000

02787000

02788000

02789000

02790000

02791000

02792000

02793000

02794000

02795000

02796000

02797000

02798000

02799000

02800000

02801000

02802000

02803000

02804000

02805000

02806000

02807000

02808000

02809000

02810000

02811000

02812000

02813000
COMMENT ** PROCESSING TIME **;
INCT1 := RELEVANTCPU.G(3);
OUTCT1 := RELEVANTCPU.G(4);
G(0) := 3 + SLEEP INCT1;
G(1) := 1;
G(3) := INCT1;
G(4) := OUTCT1;
HAND;
COMMENT** K. RECEIVED TO I/C SIS REP **;
HOLD(1980,0);
COMMENT ** PROCESSING TIME **;
G(0) := 2;
G(1) := 1;
G(3) := INCT1;
G(4) := OUTCT1;
HAND;
COMMENT ** SET UP SWITCH PATH MESSAGE TO DSS S/ **;
HOLD(1690,0);
COMMENT ** PROCESSING TIME **;
GO TO A1;
T13: HOLD(2165,0);
COMMENT ** PROCESSING TIME **;
INCT1 := RELEVANTCPU.G(3);
OUTCT1 := RELEVANTCPU.G(4);
G(0) := 8 + CPSKEP OUTCT1;
G(1) := 10;
G(3) := INCT1;
G(4) := OUTCT1;
HAND;
COMMENT** SET UP COMPLETE TO O/G CPS REP **;
HOLD(520,0);
COMMENT ** PROCESSING TIME **;
SEEK(14);
HOLD(10,0);
COMMENT ** PROCESSING TIME **;
SEEK(5);
HOLD(30,0);
COMMENT** PROCESSING TIME**;
ACTIVATE SISWINC(INCT1) AFTER CURRENT;
GO TO A1;
T14: RCLD(80,0);
COMMENT ** PROCESSING TIME **;
INCT1 := RELEVANTCPU.G(3);
OUTCT1 := CALL(INCT1), OUTGCTNUM;
G(0) := 8 + CPSKEP OUTCT1;
G(1) := 21;
G(3) := INCT1;
G(4) := OUTCT1;
HAND;
COMMENT ** RELEASE MESSAGE TO O/G CPS REP**;
HOLD(220,0);
COMMENT ** PROCESSING TIME **;
GO TO A1;
T15: RCLD(1265,0);
COMMENT ** PROCESSING TIME **;
GO TO A1;
T16: RCLD(100,0);
COMMENT ** PROCESSING TIME **;
INCT1 := RELEVANTCPU.G(3);
OUTCT1 := RELEVANTCPU.G(4);
G(0) := 3 + SLEEP INCT1;
G(1) := 3;
G(3) := INCT1;
G(4) := OUTCT1;
HAND;
COMMENT** CIRCUIT FREE TO I/C SIS REP**;
HOLD(658,0);
COMMENT ** PROCESSING TIME **;
SEEK(14);
HOLD(10,0);
SEEK(5);
HOLD(30,0);
GO TO A1;
T17: RCLD(8950,0);
COMMENT ** PROCESSING TIME**;
OUTCT1 := RANDINT(INCT1), MAXOUTCT1, SD18;
WHILE SISWINC(OUTCT1) /= NULL DO;
OUTCT1 := RANDINT(INCT1), MAXOUTCT1, SD19;
COMMENT **OUTCT1 STORES THE ABSOLUTE O/G CCT, NO. **;
SISWINC(OUTCT1) := NEW OUTGCT(WINOUTCT1);
COMMENT** GENERATE AN INSTANCE OF O/G SIS H/ **;
CALL(RELEVANTCPU.G(3), MYOUTCTWIN = SISWINC(OUTCT1));
CALL(RELEVANTCPU.G(3), OUTGCTNUM = OUTCT1);
SISWINC(RELEVANTCPU.G(3), OUTGCTNUM = OUTCT1);
SISWINC(OUTCT1), MYCALLREC := CALL(RELEVANTCPU.G(3));
COMMENT** LINK IT TO ITS CALLREC **;
G(0) := 8 + CPSKEP(RELEVANTCPU.G(3));
G(1) := 61;
G(3) := RELEVANTCPU.G(3);
G(4) := OUTCT1;
HAND;
COMMENT ** START SENDING MESSAGE TO I/C CPS REP **;
HOLD(220,0);
COMMENT ** PROCESSING TIME **;
GO TO A1;
T18: RCLD(2235,0);
AP CLASS DSSSW;

* DESCRIPTION;
* FUNCTION!

* RESPONSE TO I/C CPS REP, **;
* RESPONSE TO I/C CPS, **;
* RESPONSE TO DSS H/W;
* RESPONSE TO MSS NOT SIMULATED **;
* GO TO A;

T21: HOLD(1325,0);

COMMENT ** PROCESSING TIME **;

HOLD(1410,0);

COMMENT ** PROCESSING TIME **;

GO TO A;

T22: HOLD(3365,0);

COMMENT ** PROCESSING TIME **;

HOLD(2050,0);

COMMENT ** CALL RELEASED" TO I/C CPS REP **;

HOLD(1640,0);

COMMENT ** PROCESSING TIME **;

GO TO A;

END ** CPS PROCESS **;

COMMENT***************DSS S/W***************

* DESCRIPTION:
* DIGITAL SWITCH SUB-SYSTEM SOFTWARE PROCESS
* FUNCTION:
* SETS UP AND CLEAR PATHS OF INDIVIDUAL TELEPHONE CALLS
* VARIABLES:
* X REFLEX TO I/C SIS H/W
* RESPONSES TO I/C SIS H/W
* RESPONSES "SET UP PATH" RESPONSE FROM DSS H/W
* RESPONSES "CLEAR PATH" RESPONSE FROM DSS H/W
* INCCTS I/C CPT FOR SWITCH SET-UP
* OUTFCTS C/G ** CLEAR DOWN **
* INCCTC I/C ** CLEAR DOWN **
* OUTCTC ** CLEAR DOWN **
* CASE A SWITCH TO SELECT APPROPRIATE ACTION
* INPUT FROM:
* I/C CPS, O/G CPS, DSS H/W
* OUTPUT TO:
* I/C CPS, O/G CPS, DSS H/W
* ACTIVATES:
* DSS H/W
* ACTIVATED BY:
* NONE

***************COMMENT ENDS ***************

AP CLASS DSSSW;
COMMENT ===============;

BEGIN
  SWITCH CASE := T1, T2;
  BOOLEAN RESPONSE1, RESPONSE2;
  INTEGER INCTS, OUTCTS, INCCTC, OUTCTTC;
  BEGIN
    A: FETCH(15);
    IF CC>0 THEN GO TO CASHT RELATIVECPU.(O(1))
    ELSE IF RESPONSE1 THEN
      BEGIN
        RESPONSE1 := FALSE;
        COMMENT ** PROCESSING TIME ##:
        RESPONSE(6), UPDATE TIME := SISHING(INCCTS), RESPONSETIME(6));
        COMMENT ** END RESPONSES STORED IN CALLREG ##:
        G(0) := $2$; UPDATE(INCCTS);
        G(1) := 13; G(2) := INCCTS;
        HAND;
        COMMENT ** SWITCH RESPONSE TO I/C CPS REP ##:
        GO TO A;
      END
    ELSE IF RESPONSE2 THEN
      BEGIN
        RESPONSE2 := FALSE;
        COMMENT ** PROCESSING TIME ##:
        COMMENT ** SETUPTIME AND STORE PI OF O/G CPS REP ##:
        RESPONSE(7), UPDATE TIME := SISHING(INCCTS), RESPONSETIME(7));
        COMMENT ** END RESPONSES STORED IN CALLREG ##:
        G(0) := $2$; UPDATE(INCCTS);
        G(1) := INCCTS;
        G(4) := OUTCTTC;
        HAND;
        COMMENT ** SWITCH RESPONSE TO O/G CPS REP ##:
        GO TO A;
      END
    ELSE
      BEGIN
        BEGIN
          BLOCK(100);";
          GO TO A;
        END
        IF CC>0 PART OF IF STATEMENT##:
        TI: HOLD(1905,0);";
        COMMENT ** PROCESSING TIME ##:
        DIGITALSWITCH.SETUPPATH := TRUE;
        DIGITALSWITCH.INCCTS := RELEVANTCPU.G(3);
        DIGITALSWITCH.OUTCTTC := RELEVANTCPU.G(4);
        COMMENT ** SO THAT DSS H/W SENDS RESPONSES LATER ##:
        ACTIVATE DIGITALSWITCH AFTER CURRENT;
        GO TO A;
        T2: HOLD(1905,0);";
        COMMENT ** PROCESSING TIME ##:
        DIGITALSWITCH.CLEARPATH := TRUE;
        DIGITALSWITCH.INCCTS := RELEVANTCPU.G(3);
        DIGITALSWITCH.OUTCTTC := RELEVANTCPU.G(4);
        ACTIVATE DIGITALSWITCH AFTER CURRENT;
        GO TO A;
        END
  END
  DSS S/W ##;

COMMENT************************** DSS H/W *******************************

* DESCRIPTION:
* DIGITAL SWITCH HARDWARE
* FUNCTION:
* CONNECTS I/C AND O/G CCTS
* VARIABLES:
* SETUPPATH TO INDICATE ARRIVAL OF THAT MESSAGE
* CLEARPATH TO INDICATE ARRIVAL OF THAT MESSAGE
* INCTS I/C C/T FOR SWITCH SET-UP
* OUTCTS C/G " " " CLEAR DOWN
* INCCTS I/C " " " CLEAR DOWN
* SETUPTIME TO SET-UP SWITCH
* CLEARTIME TIME TO CLEAR DOWN THE CONNECTION
* INPUT TO:
* DSS S/W
* OUTPUT FROM:
* DSS S/W
* ACTIVATES:
* COMMENT ENDS *******************************;

END

BEGIN
  BOOLEAN SETUPPATH, CLEARPATH;
  INTEGER INCTS, OUTCTS, INCCTS, OUTCTTC;
  BEGIN
    IF SETUPPATH THEN
      SETUPPATH := FALSE;
      SETUPTIME := UNIFORM(1,0, 10000,0, SD11);
      HOLD(SETUPTIME);
      COMMENT ** TIME TAKEN TO SET UP THE PATH **;
  END

PROCESS CLASS DSSHW;

BEGIN
  BOOLEAN SETUPPATH, CLEARPATH;
  INTEGER INCTS, OUTCTS, INCCTS, OUTCTTC;
  BEGIN
    IF SETUPPATH THEN
      SETUPPATH := FALSE;
      SETUPTIME := UNIFORM(1,0, 10000,0, SD11);
      HOLD(SETUPTIME);
      COMMENT ** TIME TAKEN TO SET UP THE PATH **;

031U5000
03160000 031A4J
03107000
03198000
03110000
03111000
03112000
03113000
03114000 03144
03115000
03133000
03135000
03136000
03137000
03138000
03139000
03140000 03145
03141000
03142000 03146
03143000
03144000
03145000 03146
03146000
03147000
03148000
03149000
03150000
03151000
03152000
03153000
03154000
03155000
03156000
03157000
03158000
03159000
03160000
03161000 031643
03162000
03163000
03164000
03165000
03166000
03167000
03168000
03169000
03170000
03171000
03172000
03173000
03174000
03175000
03176000
03177000
03178000
03179000
03180000
03181000
03182000
03183000
03184000
03185000
03186000
03187000
03188000
03189000
03190000
03191000
03192000 03147
03193000
03194000
03195000
03196000
03197000 03148
03198000
03199000
03200000
03201000
COMMENT*************** O/G SIS H/W ********************

* DESCRIPTION
* O/G SIGNAL INTER-WORKING SUBSYSTEM
* FUNCTION:
* SENDS T/O OR ON O/G SIS S/W, TELLS O/G SIS REP. WHEN
* SWIS, ANSWERS AND CLEARS DOWN
* VARIABLE:
* MESSAGE AN ARRAY TO STORE I/C MESSAGES AND I/C CCT NO.
* MYSIS REF T/O CURRENCCY OF THIS CALL
* CASE A SWITCH TO JUMP TO APPROPRIATE CODE
* MYSIS REF TO O/G SIS S/W REPLICATION
* INTER-WORKING SDIISYSTEII

BEGIN

PROCESS CLASS OUTG SISN(W CCTNUM); # OUTG SISN(W CCTNUM)

COMMENT************************** COMMENT ENDS ***************;

COMMENT*************** O/G SIS H/W ********************

* DESCRIPTION
* O/G SIGNAL INTER-WORKING SUBSYSTEM
* FUNCTION:
* SENDS T/O OR ON O/G SIS S/W, TELLS O/G SIS REP. WHEN
* SWIS, ANSWERS AND CLEARS DOWN
* VARIABLE:
* MESSAGE AN ARRAY TO STORE I/C MESSAGES AND I/C CCT NO.
* MYSIS REF T/O CURRENCCY OF THIS CALL
* CASE A SWITCH TO JUMP TO APPROPRIATE CODE
* MYSIS REF TO O/G SIS S/W REPLICATION
* INTER-WORKING SDIISYSTEII

BEGIN

INTEGER NCRYBUS, NCRYES, NCRNO, NCRFREE;
INTEGER NCRFREE, NCRNO, NCRYES, NCRYBUS;
INTEGER NCRYBUS, NCRYES, NCRNO, NCRFREE,

BEGIN

PROCESS CLASS OUTG SISN(W CCTNUM);

COMMENT************************** COMMENT ENDS ***************;

BEGIN

INTEGER NCRYBUS, NCRYES, NCRNO, NCRFREE;
INTEGER NCRYBUS, NCRYES, NCRNO, NCRFREE,

BEGIN

PROCESS CLASS OUTG SISN(W CCTNUM);

COMMENT************************** COMMENT ENDS ***************;

BEGIN

INTEGER NCRYBUS, NCRYES, NCRNO, NCRFREE;
INTEGER NCRYBUS, NCRYES, NCRNO, NCRFREE,

BEGIN

PROCESS CLASS OUTG SISN(W CCTNUM);

COMMENT************************** COMMENT ENDS ***************;

BEGIN

INTEGER NCRYBUS, NCRYES, NCRNO, NCRFREE;
INTEGER NCRYBUS, NCRYES, NCRNO, NCRFREE,
A b7 IVEkS Ob.UOI
~YS~L~SAC~I
12,3l:=CCTNUM PAS5fVATt;

GO TO A:
M2: MESSAGE(2,11)=0;
TIMEOUT := UNIFORM(1.0, 50000.0, SD4); 

HOLD(TIMEOUT);
COMMENT ++ DIGITAL TIME **;
MYS.IS,MESAGE(L3,1)=MESSAGE(2,2);
COMMENT ++ T/02 TO O/G SIS REP **;
PASSIVATE;
GO TO A:
M3: MESSAGE(3,11)=0;
TIMEOUT := UNIFORM(1.0, 170000.0, SD5);

HOLD(TIMEOUT);
COMMENT ++ DIGIT2 TIME **;
MYS.I:5,MESAGE(14,1)=1;
MYS.IS.MESAGE(L4,3)=MESSAGE(3,2);
COMMENT ++ T/03 TO O/G SIS REP **;
PASSIVATE;
GO TO A:
M4: MESSAGE(4,11)=0;
TIMEOUT := UNIFORM(1.0, 90000.0, SD6);

HOLD(TIMEOUT);
COMMENT ++ DIGIT3 TIME **;
MYS.IS,MESAGE(16,1)=1;
MYS.IS.MESAGE(16,2)=MESSAGE(4,2);
COMMENT ++ T/04 TO O/G SIS REP **;
PASSIVATE;
GO TO A:
M5: MESSAGE(5,11)=0;
TIMEOUT := UNIFORM(1.0, 150000.0, SD7);

HOLD(TIMEOUT);
COMMENT ++ DIGIT4 TIME **;
MYS.IS,MESAGE(18,1)=1;
MYS.IS.MESAGE(18,2)=MESSAGE(5,2);
COMMENT ++ T/05 TO O/G SIS REP **;
PASSIVATE;
GO TO A:
M6: MESSAGE(6,11)=0;
TIMEOUT := UNIFORM(1.0, 120000.0, SD8);

HOLD(TIMEOUT);
COMMENT ++ DIGITS TIME **;
MYS.IS,MESAGE(20,1)=1;
MYS.IS.MESAGE(20,2)=MESSAGE(6,2);
COMMENT ++ T/06 TO O/G SIS REP **;
PASSIVATE;
GO TO A:
M7: MESSAGE(7,11)=0;
TIMEOUT := UNIFORM(1.0, 70000.0, SD9);

HOLD(TIMEOUT);
COMMENT ++ DIGIT6 TIME **;
MYS.IS,MESAGE(22,1)=1;
MYS.IS,MESAGE(22,2)=MESSAGE(7,2);
MYS.IS.MESAGE(22,3)=CCTNUM;
COMMENT ++ T/07 TO O/G SIS REP **;
RESPONSETIME_ELEM:1,1; RESPONSE(7,2), RESPONSETIME(5)=TIME;
ACTIVATE CALL(MYCALLSEC.CIRCUITNDN) AFTER CURRENT;
GO TO LASTACTION;
M8: MESSAGE(8,11)=0;
MYS.IS,MESAGE(19,1)=1;
MYS.IS,MESAGE(19,2)=MESSAGE(8,2);
MYS.IS,MESAGE(19,3)=CCTNUM;
COMMENT ++ "ANSWER" TO O/G SIS S/W REP **;
GO TO LASTACTION;
M9: MESSAGE(9,11)=0;
SENDACK := UNIFORM(1.0, 3000.0, SD13);

HOLD(SENDACK);
COMMENT ++ TIME TO SEND BACK"BUSY" SIGNAL **;
SENDFREE := UNIFORM(1.0, 100000.0, SD14);

HOLD(SENDFREE);
COMMENT ++ TIME TO SEND FREE"SIGNAL **;
MYS.IS,MESAGE(20,1)=1;
MYS.IS,MESAGE(20,3)=MESSAGE(9,2);
COMMA ++"FREE" MESSAGE TO O/G SIS REP **;
LASTACTI:
SISHWOT(CCTNUM) := NONE;
MYCALLSEC := NONE;
END **CALL OF THIS SIS N/W **;

COMMENT************* HYPOTHETICAL DNNSC *************

THE FOLLOWING 11 PROCESSES SIMULATE THE FUNCTIONING OF THE
-HYPOTHETICAL DNNSC -SEE CHAPTER 6 OF THESIS THE PROCESSES TASK
*PROCESS SIMULATION IS DESIGNED TO INSURE THAT THE RESULTS CORRECTLY
PROCESS COMMUNICATION EXERCISES ALL THE SERVICES OFFERED BY THE
OPERATING SYSTEM, SO THAT THE DETAILED OUTPUT TRACE COULD BE USED
*TO VERIFY THE LOGIC OF THE SIMULATOR
*THE PROCESSES MAY BE DIVIDED INTO 4 FUNCTIONAL GROUPS AS FOLLOWS:
*DIRECTLY INTERFACES WITH THE INCOMING AND OUTGOING LINES AND

END **CALL OF THIS SIS N/W **

COMMENT************* HYPOTHETICAL DNNSC *************

THE FOLLOWING 11 PROCESSES SIMULATE THE FUNCTIONING OF THE
-HYPOTHETICAL DNNSC -SEE CHAPTER 6 OF THESIS THE PROCESSES TASK
*PROCESS SIMULATION IS DESIGNED TO INSURE THAT THE RESULTS CORRECTLY
PROCESS COMMUNICATION EXERCISES ALL THE SERVICES OFFERED BY THE
OPERATING SYSTEM, SO THAT THE DETAILED OUTPUT TRACE COULD BE USED
*TO VERIFY THE LOGIC OF THE SIMULATOR
*THE PROCESSES MAY BE DIVIDED INTO 4 FUNCTIONAL GROUPS AS FOLLOWS:
*DIRECTLY INTERFACES WITH THE INCOMING AND OUTGOING LINES AND
AP CLASS TV:
COMMENT**TRAFFIC RECORDING,PI=21**;
BEGIN
  START: SEEN(13);
  IF CC>0 THEN HOLD(200.01);
  BLOCK(15);
  WHILE CC>0 DO
    BEGIN
      HOLD(200.01);
      BLOCK(15);
    END;
  PASSIVATE;
  GOTO START;
END***TV***;

AP CLASS LCH1(PI);INTEGER PI;
COMMENT**LINE CIRCUIT HANDLER PI=11**;
BEGIN
  PROCEDURE LASTJOB;
  BEGIN
    BLOCK(15);
    WHILE CC>0 DO
      BEGIN
        HOLD(400.01);
        BLOCK(15);
      END;
    PASSIVATE;
    GOTO START;
  END;
  LASTJOB;
  BEGIN
    IF CC>0 THEN
      BEGIN
        PASSIVATE;
        GOTO START;
      END
    ELSE
      BEGIN
        HOLD(200.01);
        GOTO LASTJOB;
      END;
  END;
END***LCH1***;

AP CLASS ICS;
COMMENT**INCOMING CALL SUPERVISION,PI=14**;
BEGIN
  SWITCH T1 := LEGA, LEGB, LEGC;
  PROCEDURE LASTJOB;
  BEGIN
    BLOCK(15);
    WHILE CC>0 DO
      BEGIN
        IF PA.TASKFOUNDO1 (1) =1 THEN
          BEGIN
            COMMENT**TASK(1) STORES I/C T1**;
            IF PA,TASKFOUNDO1=1 THEN LASTJOB
          END;
        END;
        END;
      END;
END***LCH2***;

AP CLASS LCH2:
COMMENT**EXCHANGE TRUNKING AND HANDLES ALL SET UP AND DISCONNECT CALLS, PI=12**;
BEGIN
  IF CC>0 THEN
    BEGIN
      PASSIVATE;
    END;
END***LCH2***;
BEGIN
HOLD(200.1); 
ULCK(J); 
END;
PASSIVATE;
GOTU START;
LND**LASTJCB**;
START: HLCK(J);
IF CC=0 THEN
BEGIN 
PASSIVATE;
GOTU START;
END
ELSE
BEGIN
G(J):=2;
HAND:
HOLD(400.0);
FETCH(6);
IF CC=0 THEN
BEGIN 
PASSIVATE;
END
ELSE
BEGIN
SEEK(b)
IF CC>O THEN
BEGIN
0101:=6;
HAND;
LAST JOB;
END
ELSE
BEGIN
FETCH(10)
IF CC>0 THEN
BEGIN
IF PA.TASKFOUND.TASK(1)=3 THEN
BEGIN
G(J):=3;
HAND;
LASTJCB;
END
BEGIN
IF PA.TASKFOUND.TASK(1)=1 THEN
BEGIN
G(J):=1;
HAND:
END
PROCEDURE LASTJOB;
BEGIN
HOLD(1200.0);
END;
PASSIVATE;
GOTU START;
END**LASTJCB**;
END**arrival OF UNBLOCKING TASK**;
END**ICG**;
AP CLASS IICG;
COMMENT**INCOMING LINE CIRCUIT ROUTiner,PI=19**;
BEGIN
PROCEDURE LASTJOB;
BEGIN
HOLD(1200.0);
WHILE CC>0 DO
BEGIN
HOLD(100.0);
END;
PASSIVATE;
GOTU START;
END**LASTJCB**;
START: HLCK(J);
IF CC=0 THEN
BEGIN 
PASSIVATE;
GOTU START;
END
ENDIF CC<O DO
BEGIN
0101:=4;
HAND;
HOLD(100.0);
G(J):=4;
HAND;
START: HLCK(J);
IF CC>0 THEN
BEGIN
0101:=1;
HAND;
GOTO LASTJOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
HAND:
GOTO LASTJOB;
END
ENDIF CC>O DO
BEGIN
0101:=2;
HAND;
HOLD(21.0);
LAST JOB;
ELSE
BEGIN
FETCH(10)
IF CC>O THEN
BEGIN
0101:=4;
HAND;
LAST JOB;
END
ENDIF CC=0 DO
BEGIN
0101:=1;
AP CLASS INSF;
COMMENT## INCORRECT REGISTER SELECTION PROCESS PI=16***;
BEGIN
PROCEDURE LASTJOB;
BEGIN
hold(1300,0);
block(15);
while cc<0 do block(15);
passivate;
goto start;
end
lastjob;
end
else
begin
if cc<0 then
block(15);
while cc<0 do block(15);
passivate;
goto start;
end
else
begin
if cc<0 then
begin
if pa.taskfound, task(1)=2 then
begin
g0(1)=2;
hand;
lastjob;
end
else
begin
if pa.taskfound, task(1)=3 and
pa.taskfound, task(5)=6 then
begin
g0(1)=1;
g0(2)=9;
hand;
lastjob;
end
end
end
end
else
begin
begin
if pa.taskfound, task(1)=3
begin
begin
begin
begin
end
end
end
end
end
end
end
end
end
AP CLASS NHR;
COMMENT## NETWORK ROUTING, PI=18***;
BEGIN
STEPI: NEXT(7);
if cc<0 then
begin
hold(300,0);
block(15);
while cc<0 do block(15);
passivate;
goto start;
end
else
begin
begin
g0(1)=1;
hand;
hold(300,0);
block(15);
while cc<0 do block(15);
passivate;
end
end

AP CLASS SW:
COMMENT***SWITCH HANDLER P1=13***;
BEGIN
SWITCH SW:=TII,T12,T13,T14,T15,T16;
START: HLCMK(3);
IF CC=0 THEN
BEGIN
PASSIVATE;
GOTO START;
END;
ELSIF
BEGIN
AGAIN:=FETCH(14);
IF CC=0 THEN
BEGIN
BLOCK(15);
WHILE CC<0 DO BLOCK(15);
PASSIVATE;
GOTO START;
END;
ELSIF
BEGIN
END**UNBLOCKING TASK***;
END**SH***;
AP CLASS OCS:
COMMENT***OUTGOING CALL SUPERVISION,P1:=15***;
BEGIN
SWITCH P1:=ICT11,ICT12,ICT13,ICT14;
START: HLCMK(3);
IF CC=0 THEN
BEGIN
PASSIVATE;
GOTO START;
END;
ELSIF
BEGIN
FINI FETCH(13);
IF CC=0 THEN
BEGIN
HLCMK(15);
WHILE CC<0 DO BLOCK(15);
PASSIVATE;
GOTO START;
END;
ELSIF
BEGIN
IF CC>0 THEN
BEGIN
GO TC P1=PA,TASKFOUND,TASK(1));
ICT11: G(0):=2;
HAND;
HOLD(600,0);
GOTC FIN;
ICT12: G(0):=1;
HAND;
HOLD(600,0);
GOTC FIN;
ICT13: G(0):=4;
HAND;
HOLD(1000,0);
GOTC FIN;
ICT14: G(0):=5;
HAND;
HOLD(1000,0);
GOTC FIN;
END;
ELSIF
BEGIN
END**UNBLOCKING TASK***;
END**SH***;
END;
END;
END;
END;
END;
**AP CLASS ULC#1**:  
COMMENT**OUTGOING LINE CIRCUIT ROUTINER,**PI=2000**;  
BEGIN  
SWITCH PT:=L1,L2,L3;  
MLK: HLCM(15);  
IF CC=0 THEN  
BEGIN  
PASIVATE;  
GOTO MLK;  
END  
ELSE  
BEGIN  
IF CC>0 THEN  
BEGIN  
GO TO F( PA.TASKFOUND.TASK(1));  
L1: G(0):=3;  
HAND:  
HOLD(100,0);  
GOTO MLK;  
L2: G(0):=1;  
HAND:  
HOLD(100,0);  
GOTO MLK;  
L3: G(0):=2;  
G(3):=1;  
COMMENT**G(3)VALUE SET HERE***;  
HAND:  
HOLD(1400,1);  
GOTO MLK;  
END;  
END*CC>0**;  
END**OLCR***;  
AP CLASS ULC#2**:  
COMMENT**OUTGOING REGISTER SELECTION PROCESS,**PI=1700**;  
BEGIN  
SWITCH ST:=LE01,LE02;  
BLOK: HLCM(15);  
IF CC=0 THEN  
BEGIN  
PASIVATE;  
GOTO BLOK;  
END  
ELSE  
BEGIN  
IF CC>0 THEN  
BEGIN  
G(0):=1;  
SEL(7);  
HOLD(1400,0);  
GO TO F( PA.TASKFOUND.TASK(1));  
LEG1:G(0):=1;  
HAND:  
HOLD(400,0);  
GOTO BLOK;  
LEG2:G(0):=2;  
HAND:  
HOLD(600,0);  
GOTO BLOK;  
END;  
END*CC>0**;  
END**OLCR***;  
**AP CLASS LCN#2**:  
COMMENT**LINE CIRCUIT HANDLER,**PI=12000**;  
BEGIN  
SWITCH Sw:=L1,L2,L3;  
START: BLOK(3);  
IF CC=0 THEN  
BEGIN  
PASIVATE;  
GOTO START;  
END  
ELSE  
BEGIN  
BLCK: BLOK(15);  
IF CC=0 THEN  
BEGIN  
PASIVATE;  
GOTO START;  
END  
ELSE  
BEGIN  
PASIVATE;  
GOTO START;  
END  
END**OLCR***;
COMMENT

INITIALISATION

BEGIN
IF PA.TASKFOUND.TASK(1) KO 0 THEN GOTO NLCA
FLSL
HUGIA
GU TC SW( PA.TASKFOUND.TASK(1));
ELSE (GU):=2;
SELF((J));
G(J):=1;
HAND;
MULU(110H,01);
GOTO NLCA;
LC(5 IF G(J)=6 THEN
HFCIN
COM:MT**G(J) VALUE TESTED HERE***;
G(J):=2;
G(J):=4;
HAND;
HCLUD(900,01);
GOTU.NLCA;
END**G(J) = 6**
ELSE
BEGIN
G(J):=2;
G(J):=6;
HAND;
HCLUD(900.01);
END**G(J) = 6**;
LC(0) := 5;
HAND;
HCLUD(900,01);
GOTO NLCA;
END**COO**;
END
END**UNLOCKING TASK**;
END**ICH2**;

COMMENT

INITIALISATION

BEGIN
COM:MT** THE FOLLOWING IS AN EXAMPLE OF INITIALISING A MODEL
* OF A MEMBER OF SYSTEM X FAMILY OF EXCHANGES - THE DMSC ******;
CALLSGEN := NEW CALLGENERATOR;
COMMENT**NOW CREATE PROCESSES INSTANCES***;
P(25):=DSSHANDLER(NEW DSSWM 25);
P(25):=CCPY("DSSHANDLER");
FOR A:= 0 STEP 1 UNTIL 1 DO
BEGIN
P(26 + K) := NEW SISSWM(26 + K);
P(26 + K,T) := CCPY("SISSWM");
END;
D I G I TA L S W I T C H := NEW DSSWM;
FOR K := 0 STEP 1 UNTIL 3 DO
BEGIN
P(49 + K) := NEW CFPSWM(49 + K);
P(49 + K,T) := CCPY("CFPSWM");
END;
BEGIN
COMMENT**NOW CREATE INPUTQ PHOLQ FOR ALL PROCESSES
SET THEIR STATES TO HLOCKED AND REQUESTED TASK OF
PRIORITY 15******;
FOR 1:=25 STEP 1 UNTIL 27,49 STEP 1 DO
BEGIN
P(1,INPUTC := NEW HEAD;
P(1,PROLOC := NEW HEAD;
P(1,PDM(1):=4;
P(1,PDM(1,1):=15;
END;
FOR J:=1 STEP 1 UNTIL 10 DO
BEGIN
IF J=0 STEP 1 UNTIL 2 DO
BEGIN
INTRIP,PMTABLE((J+1)):=(J+1);
END;
END;
FOR J:=1 STEP 1 UNTIL 4 DO
BEGIN
FOR K:=1 STEP 1 UNTIL 2 DO
BEGIN
P(49 + J,T) := CCPY("CFPSWM");
COMMENT**PERIODIC PROCESSES TABLE IS INITIALIZED
WITH DSWM AND SISSWM HAVING PERIODICITY OF 10 MSEC
NEXT INITIALIZATION PHOT'S*****;
FOR J:=2 STEP 1 UNTIL 6 DO
BEGIN
FOR J:=1 STEP 1 UNTIL 4 DO
BEGIN
P(49 + J, T) := CCPY("CFPSWM");
COMMENT**BY SO INITIALIZING TIT CONTENTS OF ONE
REPLICATION NEED ONLY BE READ IN, OTHERS TIT'S ARE
COPIED FROM THIS ONE*****;
END;
INITIALISATION 2

===============================================

PROCesses := INTt; 
P(24) := STtAcH := NEW INITIATOR(24); 
P(23) := COPY("INITIATOR") ; 
FOR I := 1 STEP 1 UNTIL PROCesses DO 
BEGIN 
STARTER.TIT(I,1) := I; 
STARTER.TIT(I,2) := 40 * I; 
END; 
BEGIN 
FOR J := 1 STEP 1 UNTIL PROCesses DO 
BEGIN 
INITIP .PTTAH[1, J*2] := 40 * J; 
END; 
BEGIN 
COMMENT*** CREATE INSTANCES OF INITIATOR PROCESS***; 
P(14) := NEW STORAGE ALLOCATOR(14); 
P(14) := NEW STORAGE ALLOCATOR(14); 
FOR I := 1 STEP 1 UNTIL PROCesses DO 
BEGIN 
P(14).TIT(I,1) := I; 
P(14).TIT(I,3) := 40 * I; 
END; 
BEGIN 
END*** FOR STATEMENTS***; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END; 
BEGIN 
END;
ULA A7 (VER: 9b.9)

1. \( \text{PRINT I:1 STEP 1 UNTIL NUM DO} \)
2. \( \text{PRINT } \)
3. \( \text{OUTTXT("CPU CPU NO.",1);} \)
4. \( \text{OUTTXT("--------------------");} \)
5. \( \text{OUTTXT("(IN BACKGROUND PROCESS =") CLINT.C(I), HEIGHTIME);} \)
6. \( \text{OUTTXT("(PERCENTAGE CPU OCCUPANCY OF PROCESSES OTHER THAN BACKGROUND="),} \)
7. \( \text{(SIMPLRCH) = (CLINT.C(I), HEIGHTIME + CLINT.C(I), NYPA, PAOVERHEAD));} \)
8. \( \text{SIMPLRCH(I100);} \)
9. \( \text{OUTIMAGL; OUTIMAGL;} \)
10. \( \text{END;} \)
11. \( \text{EJECT(2);} \)
12. \( \text{OUTLINE("RESULTS WITHOUT FALLOC(P) CALL TO PA");} \)
13. \( \text{OUTLINE("--------------------");} \)
14. \( \text{OUTTXT("ON A MULTI-PROCESSOR SYSTEM WITH");} \)
15. \( \text{OUTTXT(NUF,3);} \)
16. \( \text{OUTTXT("CPU AND"), OUTINT(NUF+PROCESSES,2);} \)
17. \( \text{OUTLINE("FCBLCALL PROCESSES =")}; \)
18. \( \text{OUTLINE("TIMES LOOP TRAVERSED");} \)
19. \( \text{FOR I:1 STEP 1 UNTIL NUMPROCESSES DO} \)
20. \( \text{OUTTXT("NCBLCALL"); OUTINT(1,2); OUTINT(P.401) QUA NOFBLCALL,} \)
21. \( \text{COUNTER,12); OUTIMAG; OUTIMAG;} \)
22. \( \text{END**OF OUTPUTTING COUNTERS CONTENTS***;} \)
23. \( \text{END;} \)
24. \( \text{END;} \)
<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TASKBLOCK</td>
<td>1696 2402 2409 2410 3444 3515 3578 3586 3636 3644 3645 3716 3771 3809 3894 3875 3885</td>
</tr>
<tr>
<td>TASKROUND</td>
<td>415 794 863 864 1076 1103 1105 1121 1295 1297 1368 1359 1460 1463 1466 1468 1515 1516 1570</td>
</tr>
<tr>
<td>TASKHANDED</td>
<td>794 1056 1060 1103 1105 1108 1126 1131 1204 1645 1658 1693</td>
</tr>
<tr>
<td>TIME</td>
<td>3488 3695</td>
</tr>
<tr>
<td>TASK</td>
<td>654 864 893 901 908 820 835 851 856 880 885 896 914 919 932 947 992 997 1005</td>
</tr>
<tr>
<td>TITLE</td>
<td>1030 1035 1113 1140 1153 1167 1168 1173 1222 1241 1243 1245 1249 1264 1267 1317 1318 1320 1340</td>
</tr>
<tr>
<td>UPDATE</td>
<td>1351 1361 1401 1403 1426 1428 1444 1454 1491 1510 1554 1565 1607 1617 1617 1659 1676 1687 1701 1746 1754</td>
</tr>
<tr>
<td>TIMEOUT</td>
<td>2208 2301 2302 2309 2313 2332 2333 2334 2340 2341 2350</td>
</tr>
<tr>
<td>TITLE</td>
<td>301 386 390 391 1623 1629 1630 1631 3964 3965 4005 4006 4007 4008 4035 4036 4037</td>
</tr>
<tr>
<td>TITLE</td>
<td>1852 1853 1873</td>
</tr>
<tr>
<td>TITLE</td>
<td>3696 3717</td>
</tr>
<tr>
<td>TITLE</td>
<td>3696 3721</td>
</tr>
<tr>
<td>TITLE</td>
<td>3696 3725</td>
</tr>
<tr>
<td>TITLE</td>
<td>3696 3729</td>
</tr>
<tr>
<td>TITLE</td>
<td>3696 3733</td>
</tr>
<tr>
<td>TITLE</td>
<td>3696 3737</td>
</tr>
<tr>
<td>TOTALCCCTS</td>
<td>133 138 143</td>
</tr>
<tr>
<td>TR</td>
<td>3411</td>
</tr>
<tr>
<td>TSX</td>
<td>2780 2783 2784</td>
</tr>
<tr>
<td>TSK</td>
<td>1863 366</td>
</tr>
<tr>
<td>TIT</td>
<td>2399 2432 2778 2792 3107 3146</td>
</tr>
<tr>
<td>TITLE</td>
<td>2399 2624 2778 2883</td>
</tr>
<tr>
<td>TITLE</td>
<td>2778 2896</td>
</tr>
<tr>
<td>TITLE</td>
<td>2778 2909</td>
</tr>
<tr>
<td>TITLE</td>
<td>2778 2932</td>
</tr>
<tr>
<td>TITLE</td>
<td>2778 2952</td>
</tr>
<tr>
<td>TITLE</td>
<td>2778 2965</td>
</tr>
<tr>
<td>TITLE</td>
<td>2778 2968</td>
</tr>
<tr>
<td>TITLE</td>
<td>2778 2985</td>
</tr>
<tr>
<td>TITLE</td>
<td>2778 3007</td>
</tr>
<tr>
<td>TITLE</td>
<td>2778 3017</td>
</tr>
<tr>
<td>TITLE</td>
<td>2778 3035</td>
</tr>
<tr>
<td>TITLE</td>
<td>2778 3056</td>
</tr>
<tr>
<td>TITLE</td>
<td>2778 3071</td>
</tr>
<tr>
<td>TITLE</td>
<td>2799 2442 2778 2798</td>
</tr>
<tr>
<td>TITLE</td>
<td>2399 2569 2778 2801</td>
</tr>
<tr>
<td>TITLE</td>
<td>2399 2580 2778 2809</td>
</tr>
<tr>
<td>TITLE</td>
<td>2399 2590 2778 2822</td>
</tr>
<tr>
<td>TITLE</td>
<td>2399 2599 2778 2842</td>
</tr>
<tr>
<td>TITLE</td>
<td>2399 2608 2778 2857</td>
</tr>
<tr>
<td>TITLE</td>
<td>2399 2616 2778 2870</td>
</tr>
<tr>
<td>TITLE</td>
<td>2780</td>
</tr>
<tr>
<td>TITLE</td>
<td>2301 3199 3214 3294 3304 3313 3322 3331 3340 3349 3366 3369</td>
</tr>
<tr>
<td>TITLE</td>
<td>1421 1494</td>
</tr>
<tr>
<td>TITLE</td>
<td>790 939 940 941 943 1215 1220 1223 1226 2272 1321 1322 1323 1337 1339</td>
</tr>
<tr>
<td>TITLE</td>
<td>1858 2573 2587 2630 2826 2846 3118 3133 3206</td>
</tr>
<tr>
<td>TITLE</td>
<td>171 173 176 181 185 206 207 210</td>
</tr>
<tr>
<td>TITLE</td>
<td>299 364 1589</td>
</tr>
<tr>
<td>TITLE</td>
<td>1666 692 696 802 831 852 881 915 993 1031 1169</td>
</tr>
<tr>
<td>TITLE</td>
<td>1811 206 365 946 1007 1008 1092 1112 1116 1272 1305 1337 1358 1375 1385 1451 1476 1488 1507 1523</td>
</tr>
<tr>
<td>TITLE</td>
<td>1533 1562 1578 1688 1614 1684</td>
</tr>
<tr>
<td>TITLE</td>
<td>298 379 580 582 584 585 600 601 1053 1056 1059 1053 1055 1058 1061 1076 1105 1106 1155 1162 1862 1864 1866</td>
</tr>
<tr>
<td>TITLE</td>
<td>2143 2403 2438 2439 2440 2445 2446 2447 2484 2485 2496 2497 2501 2509 2513 2521 2525 2537 2545</td>
</tr>
<tr>
<td>TITLE</td>
<td>2545</td>
</tr>
<tr>
<td>TITLE</td>
<td>2566 1061 1063 1077 1100 1104 1105 1106 1107 1108 1119 1120 1121 1209 1209 1209 1660 1654 2405 2572 2575</td>
</tr>
<tr>
<td>TITLE</td>
<td>2576 2578 2583 2584 2585 2593 2594 2596 2597 2602 2603 2608 2606 2611 2612 2613 2614 2618 2619</td>
</tr>
<tr>
<td>TITLE</td>
<td>2621 2622 2627 2628 2629 2631 2637 2638 2639 2641 2782</td>
</tr>
<tr>
<td>TITLE</td>
<td>786 1652 1653 1655</td>
</tr>
</tbody>
</table>

**TOTAL NUMBER OF DIFFERENTLY SPELLED IDENTIFIERS AVAILABLE IN THIS PROGRAM:** 463

**NO DIAGNOSTICS FOR THIS COMPILATION.**
APPENDIX C

A SAMPLE OF THE TRACE PROGRAM OUTPUT

PA SIMULATION RUN TRACE
===================================

SIMULATED TIME = 1000000000
CLOCKINT AT TIME = 0
INTRIP ENTERED NOW
INT. TRIPPLEATES ENTERED FOR CLOCKINT 0
INTRIP HAS SERVCEL CLOCKINT
CPL INTERRUPTED IS AC. 2
AND ITS INTERRUPT NC. 1
AT RUNTIME = 0

PROCESS ENTERED IN SUSPMAP IS BACKGROUND 117
AT RUNTIME = 132

SUSPENDED PROCESSES IN SUSPMAP ARE:-
BACKGROUND 117

PROCESS HANDEL TASK IS INTIM 16
AT RUNTIME = 267
NC. OF TASKS IN INPUT = 1

INPUT TASKS PRIORITIES ARE:-

PROCESS ENTERED IN SUSPMAP IS INTIM 16
AT RUNTIME = 305

SUSPENDED PROCESSES IN SUSPMAP ARE:-
INTIM 16
BACKGROUND 117

CHosen PROCESS IS INTIM 16
AT TIME = 327
TC RUN ON CPU AC. 2

SUSPENDED PROCESSES NOW ARE:-
BACKGROUND 117

LCFL UPDATED ACW IS NC. 1
SELECTED PROCESS STATE IS SUSP (UNBLOCKED) AT RUNTIME = 359
AND ITS PI = 16

PROCESS SELECTED TO RUN IS INTIM 16
TO RUN ON CPU AC. 2
NC. OF TASKS IN INPUT QUCE = 0
AT RUNTIME = 575

INTIM CORRECTELY ENTERED
VALUE OF VARIABLE X NEW IS 1
PI STORED IN PPTAULE = 0

CALL TO HANC BY PROCESS INTIM 16
ON CPU NU. = 2
AT RUNTIME = 648
OGTI 1
AND CALL INDEX = 2
PROCESS HANDED TASK IS RASH 41
AT RUNTIME = 663
NO. OF TASKS IN INPUT = 1
HANCING PROCESS IS INTIM 16

INPUT TASKS PRIORITIES ARE:-
5
PROCESS INTERED IN SUSMAP IS RASH 41
AT RUNTIME = 713

SUSPENDED PROCESSES IN SUSMAP ARE:-
RASH 41
BACKGROUND 117

SUSPPOINT TRIGGERED AT TIME = 713
BY PROCESS ALLOCATOR ON CPU NO.
BECAUSE HIGHEST PRIC. SUSP PROCESS IS RASH 41
AND LCPU CURP IS BACKGROUND 116
AND LCPU IS NC.
1

INTRIP ENTERED NEW
INTRIP ENTERED BECAUSE SUSPPOINT TRIGGERED
CPU INTERRUPTED IS NO.
1
AND ITS INTERRUPTED IS.
1
AT RUNTIME = 724

VALUE OF VARIABLE X NEW IS
2
PI STORED IN PTABLE = 41
CALL TO HANG BY PROCESS INTIM 16
ON CPU NO.
AT RUNTIME = 615
UGTI
AND CALL INDEX =
2
PROCESS HANDED TASK IS RASH 42
AT RUNTIME = 835
NO. OF TASKS IN INPTG = 1
HANCING PROCESS IS INTIM 16

INPTG TASKS PRIORITIES ARE:-
5
PROCESS INTERED IN SUSMAP IS BACKGROUND 116
AT RUNTIME = 856

SUSPENDED PROCESSES IN SUSMAP ARE:-
RASH 41
BACKGROUND 116
BACKGROUND 117

PROCESS INTERED IN SUSMAP IS RASH 42
AT RUNTIME = 884

SUSPENDED PROCESSES IN SUSMAP ARE:-
RASH 41
RASH 42
BACKGROUND 116
BACKGROUND 117

SUSPPOINT TRIGGERED AT TIME = 903
BY PROCESS ALLOCATOR ON CPU NO.
BECAUSE HIGHEST PRIC. SUSP PROCESS IS RASH 41
AND LCPU CURP IS BACKGROUND 116
AND LCPU IS NC.

CHosen PROCESS IS RASH 41
AT TIME = 903
TO RUN ON CPU NO.
1

SUSPENDED PROCESSES NOW ARE:-
RASH 42
BACKGROUND 116
BACKGROUND 117
APPENDIX D
A SAMPLE OF OUTPUT REPORTS OF FBLOCK(P)

EXPERIMENT
* * * PROCESS ALLOCATOR SIMULATOR OF HKE DL SYSTEM * * *
* * * A. S. SALTZ PLYMOUTH POLYTECHNIC JUNE, 1979 * * *
** PARAMETERS OF THIS SIMULATOR ARE: **
** SIMULATION LANGUAGE: SIMULA **
** NUMBER OF CPUS = 2 **
** EXPERIMENTAL EVALUATION OF FBLOCK CALL TO PA **

** SIMULATED TIME = 720000.00 **

SOFTWARE PROCESS AND THEIR PROCESS INDICES.

INDEX OF INITI! PROCESS IS 16
INDEX OF STORAGE ALLOCATOR PROCESS IS 14
INDEX OF NOFPROC!CALL 115 41
INDEX OF NOFPROC!CALL 215 46
INDEX OF NOFPROC!CALL 315 46

**STATISTICS FOR PROCESS NO 16 **

MAX NO. OF TASKS IN I/P Q = 1
MIN NO. OF TASKS IN I/P Q = 0
MAX PROC ALLOCS WAITING PROLOM = 0
MIN PROC ALLOCS WAITING FOR PROLOM = 0
TIMES PROC INITIATED = 3760
TIME PROC ALLOCS WAITING PROLOM = 0.00
TIMES PROCESS INITIATED = 1440

**STATISTICS FOR PROCESS NO 41 **

MAX NO. OF TASKS IN I/P Q = 2
MIN NO. OF TASKS IN I/P Q = 0
MAX PROC ALLOCS WAITING PROLOM = 0
MIN PROC ALLOCS WAITING FOR PROLOM = 0
TIMES PROC INITIATED = 4321
TIME PROC ALLOCS WAITING PROLOM = 0.00
TIMES PROCESS INITIATED = 720

**STATISTICS FOR PROCESS NO 46 **

MAX NO. OF TASKS IN I/P Q = 2
MIN NO. OF TASKS IN I/P Q = 0
MAX PROC ALLOCS WAITING PROLOM = 0
MIN PROC ALLOCS WAITING FOR PROLOM = 0
TIMES PROC INITIATED = 4321
TIME PROC ALLOCS WAITING PROLOM = 0.00
TIMES PROCESS INITIATED = 720

**STATISTICS FOR PROCESS NO 45 **

MAX NO. OF TASKS IN I/P Q = 2
MIN NO. OF TASKS IN I/P Q = 0
MIN NO. OF TASKS IN I/P Q = 0
MAX PROC ALLOCS WAITING PROLOM = 0
MIN PROC ALLOC Waiting FOR PROLOM = 0
TIMES PROLO INITIATED = 4321
TIMES PROC ALLOCS WAITING PROLOM = 2,00
TIMES PROCESS INITIATED = 720

**STATISTICS FOR THE PROCESS ALLOCATOR OF CPU NO. 1**

TIMES PROCESS ALLOCATOR CALLED = 10153
TIMES PROCESS ALLOCATOR INTERRUPTED = 721
THIS PA OVERHEAD = 236061.00
THIS PA FETCH AND BLOCK OVERHEAD = 509584.69

**STATISTICS FOR THE PROCESS ALLOCATOR OF CPU NO. 2**

TIMES PROCESS ALLOCATOR CALLED = 5834
TIMES PROCESS ALLOCATOR INTERRUPTED = 2159
THIS PA OVERHEAD = 1619032.00
THIS PA FETCH AND BLOCK OVERHEAD = 370890.37

**STATISTICS OF FREELO**

MAX NO. OF PROCESS ALLOCATORS WAITING FOR FREELOM = 1
MIN NO. OF PROCESS ALLOCATORS WAITING FOR FREELOM = 0
TIME FREELO ENGAGED = 14527
TIME SPENT BY PROCESS ALLOCS WAITING FOR FREELOM = 369

**STATISTICS OF SUSPLO**

MAX PROC ALLOCS WAITING SUSPLOM = 0
MIN PROC ALLOCS WAITING SUSPLOM = 0
TIME SUSPLO ENGAGED = 14544
TIME SPENT BY PROC ALLOCS WAITING SUSPLOM = 0

**STATISTICS OF INTLO**

MAX PROC ALLOCS WAITING INTLOM = 1
MIN PROC ALLOCS WAITING INTLOM = 0
TIME INTLO ENGAGED = 12212
TIME SPENT BY PROC ALLOCS WAITING INTLOM = 12260

**CPU STATISTICS**

FOR CPU NO. 1

TIME ON BACKGROUND PROCESSES = 2517245.00
PERCENTAGE CPU OCCUPANCY OF PROCESSES OTHER THAN BACKGROUND = 34.60

FOR CPU NO. 2

TIME ON BACKGROUND PROCESSES = 4122133.00
PERCENTAGE CPU OCCUPANCY OF PROCESSES OTHER THAN BACKGROUND = 47.30

RESULTS WITHOUT FELOCK(P) CALL TO PA

FOR A MULTI-PROCESSOR SYSTEM WITH 3 CPUS AND 3 NOFBLCALL PROCESSES

NOFBLCALL 1 720
NOFBLCALL 2 720
NOFBLCALL 3 720
SOFTWARE PROCESS AND THEIR PROCESS INDICES

INDEX OF INITIAL PROCESS IS 15
INDEX OF STORAGE ALLOCATION PROCESS IS 16
INDEX OF WITHFBLCALL 115
INDEX OF WITHFBLCALL 215
INDEX OF WITHFBLCALL 315

*STATISTICS FOR PROCESS NO 1*
...
...

*STATISTICS FOR PROCESS NO 16*
...
...

*STATISTICS FOR PROCESS NO 41*
...
...

*STATISTICS FOR PROCESS NO 42*
...
...

*STATISTICS FOR PROCESS NO 43*
...
...

*STATISTICS FOR THE PROCESS ALLOCATOR OF CPU NO 1*
...
...

*STATISTICS FOR THE PROCESS ALLOCATOR OF CPU NO 2*
...
...

*STATISTICS OF FREELO **
...
...

*STATISTICS OF SUSPLO**
...
...

*STATISTICS OF INTULO**
...
...

*CPUS STATISTICS**
...
...

RUN RESULTS WITH PBLOCK(P) CALL

ON A MULTI-PROCESSOR SYSTEM WITH 2 CPU'S AND WITHFBLCALL PROCESSES 1-

TIME LOOP TRAVERSED
...
\[
= \pm \sqrt{\frac{\sum (xy)^2}{\sum x^2} \frac{\sum x^2}{\sum y^2}} = \pm \sqrt{\frac{(\sum xy)^2}{(\sum x^2)(\sum y^2)}}
\]

or \[ r = \frac{\sum xy}{\sqrt{(\sum x^2)(\sum y^2)}} \] \[ ..........(7) \]

when \( Y \) increases as \( x \) increases, then \( r \) is positive. This is known as the product-moment formula for correlation coefficient.
REFERENCES

ANDE 72 ANDERSON H.A. and SARGANT R.G.
"A Statistical Evaluation of the Scheduler of an Experimental Interactive Computing System"

ANDE 72 ANDERSON R.R., HAYES J. and SHERMAN D.N.
"Simulated Performance of a Ring-Switched Data Network"

AMOS 76 AMOS E. and JOEL J.
"Electronic Switching : Central Office Systems of the World"
IEEE Press, 1976

ARGO 79 ARGOE D.T. and WILBER J.A.
"System Recovery in the 2B Processor"

BABC 78 BABCICKY K.
"SIMULA Programming Efficiency Considerations"
Association of SIMULA Users Workshop 'Second Steps in SIMULA' University of Bradford 18th December, 1978

BAER 68 BAER J.L. and RUSSELL E.C.
"Modelling and Scheduling of Computer Programs for Parallel Processing Systems"
Proc. 2nd Conf. on Applications of Simulation Dec. 1968 N.Y. pp 278-281

BAKE 75 BAKER F.T.
"Notes on Structured Programming"
In 'Structured Programming' Academic Press, 1972

BARK 59 BARKER P.E. and WATSON H.K.
"Calibrating the Simulation Model of the IBM System /360 Time Sharing System"
Proc. Conf. on Applications of Simulation, Los Angeles Dec 1969 pp 130-137

BARR 71 BARRON D.W.
"Computer Operating Systems"
Chamption and Hall Ltd, 1971

BAUE 74 BAUER M.J.
Proc. ACM-SIGSIM Symp. 4-6 June, 1974 pp 165-173

BEAR 79 BEAR D.
"Outline of a Theoretical Traffic Model of the Mark II BL System"
The General Electric Co. Ltd., Hirst Research Centre Memorandum No. TRL/767, February, 1979
BEAS 73  BEASTALL H. and POVEY J.A.
"Teletraffic Studies of TXE4"

BECK 73  BECK D. S.
"The Plessey System 250-Facilities for Simulation of
Telecommunication Equipment"
Proc. IEE Conf. on Software Engineering for Telecommunications
Switching Systems, 2-5, April, 1973 IEE Pub. No. 97 pp 31-38

BEIL 77  BEILNER H. and GELENBE E. (Editors)
"An Approach to the Straightforward Production of Computer
System Simulators"
In 'Modelling and Performance Evaluation of Computer Systems'
North-Holland Pub. Co. 1977

BELL 72  BELL T.E.
"Objectives and Problems in Simulating Computers"
Proc. AFIPS-FJCC Conf. 1972, pp 287-297

BELS 78  BEILSNESS D. and BRINGSRUD K.A.
"X.25 DTE Implemented in SIMULA"
EUROCOMP 78, London, May, 1978

BERN 68  BERNARD R. and GRANDJEAN C.
"Application of GPSS to the Simulation of Telephone Systems"
Proc. 2nd Conf. on Applications of Simulation 2-4 Dec.,
1968 N.Y. pp 194-197

BIRT 73  BIRTWHISTLE G.M., DAHL O.J., MYHRHANG B. & NYGAARD K.
"SIMULA Begin"
Studentlitteratur, Sweden, 1973

BIRT 79  BIRTWHISTLE G.M.
"DEMONS - Discrete Event Modelling on SIMULA"
Macmillan, 1979

BLUN 67  BLUNDEN G.P.
"Implicit Interaction in Process Models"
Proc. IFIP CONF. on Simulation Programming Languages, May, 1967
Oslo, Norway pp 283-291

BLUN 74  BLUNK R.G.
"Simulation of an OS/360 MVT Environment with GPSS"
Proc. ACM-SIGSIM Symp. 4-6, June, 1974 pp 53-61

BOUT 78  BOUTE R.T.
"Logical Models for Computer Control of Telephone Exchanges"
3rd Inter. Conf. on Software Eng. for Telecomm. Switching

BRAE 79  BRAEK R.
"Functional Specification and Description Languages as a
Practical Tool for Improved System Quality"
ITU Telecom. 79 Forum, Geneva, Sept. 1979 pp 1.3.1.1-1.3.1.9

BRIL 77  BRILEY B.E. and TOY W.N.
"Telecommunications Processors"

R2
BROW 78
BROWN J.G.
"Developing an Electronic Local Exchange Register Translator for the British Post Office"
Electronics and Power June 1978 pp 448-450

BRUC 79
BRUCE R.A., GILOTH P.K. & SPIEGEL E.H.
"No. 4 ESS: A Continuing Evolution"
Bell Labs. Records, Vol57, No.6 1979 pp 155-162

BUCH 69
BUCHOLZ W.
"A Synthetic Job for Measuring System Performance"

BUSS 68
BUSSARD D.L.
"Communication Network Design Using Message Flow Simulation"
Proc. 2nd Conf. on Applications of Simulation N.Y. Dec 1968 pp 198-201

BUXT 62
BUXTON J.N. and LASKI J.G.
"Control and Simulation Language"

BUXT 66
BUXTON J.N.
"Writing Simulations in CSL"
Computer Journal, 1966 Vol 9, pp 137-143

CALL 67
CALINGERT P.
"System Performance Evaluation: Survey & Appraisal"

CASY 77
CASY L.M.
"Computer Structures for Distributed Systems"

CCIT 77
CCITT (Consultative Committee for International Telephony & Telegraphy)
"Functional Specification and Description Language (SDL)"
CCITT Recommendations Z-101-Z103, Orange Book Vol. VI.4
ITU Geneva, 1977

CCIT 77A
Consultative Committee for International Telephony & Telegraphy
"Programming Languages for Stored-Programme Control Exchanges"
CCITT Orange Book Vol VI.4 Geneva, 1977

CHAN 78
CHANDY K.M. & YEH R.T. (Editors)
"The Regenerative Method for Simulation Analysis"
Prentice-Hall Inc. 1978

R3
<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Title</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHENG P.S.</td>
<td>Trace Driven System Modelling</td>
<td>IBM System Journal, Vol.8, No.4, 1969 pp 280-289</td>
</tr>
<tr>
<td>CULLEN A.</td>
<td>The TRL Multimicroprocessor Operating System: First Design</td>
<td>GEC Hirst Research Centre Memo No. TRL/814 Nov. 1979</td>
</tr>
</tbody>
</table>
DAHL 66  DAHL O.J. and NYGAARD K.
"SIMULA - An ALGOL-based Simulation Language"
Comm. of ACM Vol 9, No.9 Sept 1966
pp 671-678

DAHL 67  DAHL O.J. and NYGAARD K.
"Class and Sub-Class Declarations"
Proc. IFIP Conf. on Simulation Programming Languages
May, 1967 Oslo Norway, pp 158-174

DAHL 68  DAHL O.J.
"Discrete Event Simulation Languages"
In 'Programming Languages' NATO Advanced Study
pp 349-395

DAHL 70  DAHL O.J., MYHRHAUG B. and NYGAARD K.
"Common Base Language"

DAVI 74  DAVIS M. and MILLER G.E.
"What Price Virtual Memory"
Proc. ACM-SIGSIM 4-6 June, 1974 pp 85-93

DENN 68  DENNING P.J.
"The Working Set Model for Program Behaviour"
Communications of ACM Vol 11, No.5, May 1968
pp 323-333

DENN 70  DENNING P.J.
"Virtual Memory"
pp 153-189

DENT 78A  DENT G.R.
"Call Processing Subsystem - General Description"
Document 1 ADA 00003 AAD-BA. The General Electric
Co. Ltd., of England. 1978

DENT 78B  DENT G.R.
"CALL Processing Subsystem - Design Text"
Document 1 ADA 00003 AAD-CA. The General Electric
Co. Ltd., of England 1978

DENT 78C  DENT G.R.
"Calling Processing Subsystem - Interface Specification"
Ltd. of England 1978

DIET 75  DIETRICH G.
"Traffic Model of Common Control Switching Systems"
Electrical Communications (GB) Vol. 50 No.1, 1975
pp 28-34

DIJK 68  DIJKSTRA E.W.
"Co-operating Sequential Processes"
in F. Gennys (Ed.): Programming Languages
Academic Press, London and N.Y. 1968

R5
DIJK 72  DIJKSTRA E. W.  
"Notes on Structured Programming" in 'Structured  
Programming', Academic Press. 1972

DGWE 73  DGWELL D.  
"Processor Environment Simulation"  
Proc. IEE Conf. on Software Eng. for Telecomm. Switching  
Systems 2-5 April, 1973 IEE Pub. No. 97

EBNE 79  EBNER G.C. & TOMKO L.A.  
"CCIS : Signalling System for the Stored Program Controlled  
Network"  
Bell Lab. Records Vol.57, No.2 Feb. 1979

EDGE 72  EDGE G.  
"System 250 and Diagnostics"  

ELLI 78  ELLISON D.  
"Need a Language - Where Shall I Turn?"  
In SIMULATION Newsletter, Summ. 78 Centre in Simulation  
University of Lancaster, Lancaster U.K.

EMSH 70  EMSHOFF J.R. & SISSON R.L.  
"Design and Use of Computer Simulation Models"  
The Macmillan Co. 1970

FAGA  FAGAN B.M. & KLEINE H.  
"A Simulation Model of a Message Switching System"  
Proc. ACM-SIGSIM, 3-6, June, 1974 pp 62-73

FISH 67  FISHMAN G.S. & KIVIAT P.J.  
"Digital Computer Simulation : Statistical Considerations"  
The Rand Corp. RM-5287-PR, 1967 Santa Monica, Calif.

FISH 68  FISHMAN G.S. & KIVIAT P.J.  
"The Statistics of Discrete Event Simulation"  
Simulation, April, 1968 pp 185-195

FISH 73  FISHMAN G.S.  
"Concepts & Methods in Discrete Event Digital Simulation"  
John Wiley & Sons Ltd., 1973

FLOO 75  FLOOD J.E. (Ed)  
"Telecommunications Networks"  
Peter Peregrinus Ltd., 1975

FLOO 76  FLOOD J.E.  
"Alexander Graham Bell & the Invention of the Telephone"  

FONT 71  FONTAINE B.  
"Real-Time Environment Simulation"  
Electrical Communications (GB) Vol.46 No.3, pp 188-190, 1971

FRAN 73  FRANKLIN M. & LOONEY M.  
"Simulation of a Computer System with Single and Dual  
Density Disks"  
Proc. ACM Symp. on Simulation of Computer Systems, 19-20 June  
1973 pp 259-267
FRAN 74  FRANTA W.R. and HOULE P.A.  
"Comments on Models of Multiprocessor Multimemory  
Bank Computer Systems"  
pp87-97  
FRAN 77  FRANTA W.R.  
"The Process View of Simulation"  
Elseveir North-Holland Inc., 1977  
FREI 72  FRIELBERGER E. W. - Editor  
"Statistical Computer Performance Evaluation"  
GALE 75  GALE N.  
"Application of State Transition Diagrams to  
System Independent Specification of Telephone  
Facilities and Systems"  
ATR (Australia) Vol 9, No.2, 1975 pp 3-12  
GEC 75A  "GEC Mark II BL Communications Control Processor  
System"  
GEC Document 950-0001-001 Sept, 1975 Coventry, U.K.  
GEC 75B  "GEC Mark II BL Interrupt and Timing Process Definition  
Text"  
GEC 76A  "POPUS - Process Allocator General Description"  
GEC Document lSAA-00-254-AAS/BA, 1976 Coventry, U.K.  
GEC 76B  "Process Allocator for Emulator"  
GEC Document 803-9085-001, 1976, Coventry, U.K.  
GEC 77  "Storage Allocator Definition Text"  
GEC Document lSAA 00012 AAP/BA, 1977 Coventry, U.K.  
GEC 78  "GEC Mark II BL Thrash and Crash Processes Definition  
Text"  
GERR 74  GERRAND P.H.  
"Use of Call-State Transition Diagrams in the Analysis  
of Telephone Call Failure Probabilities"  
A.T.R. (Australia), Vol 8, No.1, 1974 pp 7-12  
GERR 79  GERRAND P.H. and PARK J.L.  
"Development of a Processor Monitoring Instrument  
for SPC Exchanges: Facilities and Application Areas"  
pp 113-120  
GIBS 70  GIBSON J.C.  
"The GIBSON Mix"  
IBM TR 00-2043, June, 1970  
GORD 78  GORDON G.  
"System Simulation"  
GRANTGES R.F. and SIMOWITZ N.R.  
"NEASIM : A General Purpose Computer Simulation Program for Load-loss Analysis of Multistage Central Office Switching Networks"  
B.S.T.J. May, 1964 Vol.43, No.3 pp 965-1004

GREEN H.  
"Design Suggestions for Overload Control"  
GEC Telecommunications Ltd., Memo No. GEC/DMNSC/DEV/DP (80) 15, October, 1980

GRUSZECKI M. and CORNELIS F.  

GUITONNEAU G.T., LEMMONIER J.L. and SOULET J.J.  
"A Simulator for Debugging and Evaluating the E.11 System"  

HALTON D.  
"System Reliability - S250 Multiprocessor"  
IEE Vac. School on 'Stored Program Control of Telephone Switching Systems' University of Essex, 11-16, Sept, 1977 pp 13/1 -13/13

HANSEN Per Brinch  
"Operating Systems Principle"  

HANSEN Per Brinch  
"The Architecture of Concurrent Programs"  

HARRIS L.R.F. and DAVIS E.  
"System X and the Evolving U.K. Telecommunications Network"  
P.O.E.E.J. Vol, 72, April, 1979 pp 2-8

HARTLEY M.G. (ED)  
"Digital Simulation Methods"  
IEE Monograph No. 15, Peter Peregrinus Ltd, 1975

HILLS P.R.  
"An Introduction to Simulation Using SIMULA"  
Norwegian Computing Centre Pb. No. S55, Oslo, 1973

HILLS P.R.  
"Simulation Model Structures"  
SIMULA Newsletter, Vol. 1 No.2, Aug, 1973 pp 3-4

HILLS P.R. and BIRTWHISTLE G.  
"SIMON 75 : Reference Manual"  
Robin Hills (Consultants) Ltd., 48, High Street, Exeter U.K. 1976

HILLS M.T. and KANO S.  
"Programming of Electronic Switching Systems"  
Peter Peregrinus Ltd, 1976
HORN 72 HORN R.V.
"Validation"
In 'The Design of Computer Simulation Experiments'
Edited by T.H. Naylor, Duke University Press, Durham
1972 pp 232-251

HUES 67 HUESMAN L.R. and GOLDBERG R.P.
"Evaluating-Computer Systems Through Simulation"
pp 150-155

HUTC 67 HUTCHINSON G.K.
"Some Problems in the Simulation of Multiprocessor
Computer Systems"
Proc. IFIP Conf. on Simulation Programming Languages,
Oslo, May, 1967 pp 305-318

HUTC 73 HUTCHINSON G.K.
"The Use of Micro-level Simulation in the Design of
a Computer Supervisory System"
Proc. ACM-SIGSIM Symp. on Simulation of Computer Systems

IGLE 78 IGLEHART D.L. and SHELDER G.S.
"Regenerative Simulation of Response Times
in Networks of Queues"

JAME 78 JAMES Y.
"Simulation of a Business Communication System Switching
Network"

JONE 79 JONES W.G.T., KIRTLAND J.P. and MOORE W.B.
"Principles of System X"
P.O.E.E.J. Vol 72, July, 1979 pp 75-80

JOUB 78 JOUBERT T. and MILLET C.
"Modelling of Stored Program Controlled Switching
Systems: Simulation versus Analytical Methods"
International Conf. on Software Engineering for Telecommunication

KALV 72 KALVENES S.
"An Introductory Course in Statistics for Simulation"

KASP 74 KASP H. and MILLER D.S.
"Job and Job Stream Modelling in the TRW Timeshare
System Simulation"
Proc. ACM-SIGSIM 4-6, June, 1974 pp 94-121

KATZ 66 KATZ J.H.
"Simulation of a Multiprocessor Computer System"
Proc. AFIPS-SJCC Vol28, 1966 pp 127-139

KAWA 71 KAWASHIMA H., FUTAMI K. and KANO S.
"Functional Specification of Call Processing by State
Transition Diagrams"
pp 581-587
KAWASHIMA H., ARIMA T. and SHIMIZU Y.
"Software Structure for Today and Tomorrow"
pp 1.3.2.1-1.3.2.8

KAY I.M.
"An Over-the-Shoulder Look at Discrete Simulation Languages"

KELLEY D.H. and BUXTON J.N.
"Montecode - an Interpretive Program for Monte Carlo Simulations"

KITZ B.
"Problems Encountered in the Programming of Games and Simulations"

KIVIAT P.J.
"Digital Computer Simulation : Modelling Concepts"
The Rand Corp. RM-5378-PR Santa Monica Cal. 1967

KIVIAT P.J.
"Digital Computer Simulation : Computer Programming Languages"
The Rand Corp. Santa Monica, Cal., 1969

KIVIAT P.J., VILLANEUVA R., and MARKOWITZ H.
"The SIMSCRIPT II Programming Language"

KLEINE H.
"A Survey of Users' Views of Discrete Simulation Languages"

KLEINE H.
"A Second Survey of Users' Views of Discrete Simulation Languages"
Simulation August, 1971 pp 89-94

KLEIJNEN J.P.C.
Marcel Dekker Inc. N.Y. 1974

KNUTH D.E. and MCNELEY J.L.
"SOL - A Symbolic Language for General-Purpose System Simulation"
IEEE Trans. on Electronic Computers, Aug.1964 pp 401-408

KOJAYASHI H.
"Modelling and Analysis: An Introduction to System Performance Evaluation Methodology" - Addison-Wesley, 1978

KOBUS S., TRUITHOF A. & VIELLEVOYE L.
"Central Control Philosophy for the METAONTAL Switching System"
Electrical Communications Vol.47 No.3, 1972 pp 159-162


KRAS 67  KRASNOW H.S.  "The Process View and Management Systems"  Proc. IFIP Conf. on Simulation Programming Languages  May 1967 Oslo, Norway pp 1-12


LAMP 68  LAMPSON B.W.  "A Scheduling Philosophy of Multiprocessing Systems"  Comm. of ACM Vol.11 No.5 May, 1968 pp 347-360

LAMP 79  LAMPREABE M.A. DEL COSO  "Environment Simulation for Real-Time Software Processes"  Electrical Communication (GB) Vol.54 No.2 1979 pp 143-148


LAWR 77  LAWRENCE G.N.  "Program Organisation : Application Programs"  2nd IEE Vac. Sch. on Stored Program Control of Telephone Switching Sys. Univ. of Essex, 11-16 Sept. 77


LEHM 68  LEHMANN M. and ROSENFIELD J.L.  
"Performance of a Simulated Multiprogramming System"  
Proc. AFIPS-FJCC Vol 33, 1968 pp 1431-1442

LEO 68   LEO J.C.  
"S3, The System and Software Simulation"  

LOKE 74   LOKEN B.  
"TETRASIM Model Description"  
Norwegian Computing Centre. Oslo, Norway Pub. 5-26 Aug, 1974

LOVD 77  LOVDAL E. and BEILNESS D.  
"Use of Simulation Techniques in Actual Network Implementation"  
Proc. Symp. on Simulation & Modelling of Data Networks,  
Oslo, Sept. 1977 pp 1-16

LUCA 71  LUCAS H.C.  
"Performance Evaluation and Monitoring"  

MACD 73A  MACDOUGAL M. and McALPINE J.S.  
"Computer System Simulation with ASPOL"  

MACD 73B  MACDOUGAL M.  
"Simulating the NASA Mass Data Storage Facility"  
Symp. on the Simulation of Computer Sys. 4-6 June, 1974  
pp 32-43 SIGSIM-ACM

MACH 74  MACHESON S.T.  
"Simulation Program Generators"  
Simulation, 23(6), Dec. 1974 pp 181-189

MAGU 72  MAGUIRE J.H.  
"Discrete Computer Simulation-Technology and Applications - the Next Ten Years"  

MAND 76  MANDIGO P.D.  
"No. 2B ESS : New Features for a More Efficient Processor"  
Bell Labs. Records. Vol.54 No.1 1976 pp 304-309

MARK 77  MARK A., EGGLEBERGER O. and NEHERER J.  
"Experiences in the Design and Implementation of a Structured Real-time Operating System"  
Proc. IFAC/IFIP Workshop on Real Time Program. Eindhoven,  
Netherlands, 20-22, June, 1977 pp 133-137

MART 79  MARTIN J.  
"System X"  
P.O.E.E.J. Vol.71, Jan 1979 pp 221-224

MAY 80  MAY C.A.  
"Communication"  

MCQU 81  McQUADE E., GRAY H. and SALIH A.M.  
"A Process Interaction Approach to the Simulation of a Multiprocessor Computer System"  
To be pub. in SIMULATION, the Technical Journal of the Society of Computer Simulation, U.S.A.
MIHR 71  MIHRAM G.A.  
"Simulation: Statistical Foundation and Methodology"  
Academic Press, N.Y. 1971

MIHR 72  MIHRAM G.A.  
"Some Practical Aspects of the Verification and Validation of Simulation Models"  

MORA 79  MORALEE D.  
"The System X Project"  
Electronics and Power, Vol.25 No.8 August, 1979 pp 544-551

NAUR 63  NAUR P. (ED.)  
"Revised Report on the Algorithmic Language ALGOL 60"  
CACM Vol16. No.1, 1963, pp 1-17

NAYL 66  NAYLOR T.H., BLINTFY J.L. BURDICK D.S. and CHU K.  
"Computer Simulation Techniques" - Chapter 8  
John Wiley & Sons Inc. 1966

NAYL 67  NAYLOR T.H. and FINGER J.M.  
"Verification of Computer Simulation Models"  

NIEL 67  NIELSEN N.R.  
"The Simulation of Time-Sharing Systems"  
Communications of the ACM Vol.10, No.7 July 1967 pp 397-412

NISS 79  NISSEN J.C.D. and GIEGER G.V.  
"A Fault-Tolerant Multimicroprocessor: for Telecommunications and General Applications"  
GEC Telecommunications Ltd., Coventry England 1979

NOE 74  NOE J.D. and NUTT G.J.  
"Validation of a Trace-Driven CDC 6400 Simulation"  

NUTT 74  NUTT G.J.  
"The Simulation of Computer Systems in a University Environment"  
Proc. ACM-SIGSIM Symp. on Simulation of Computer Systems  
Gaithersburg, U.S.A. 4-6, June, 1974 pp 25-29

OPPE 68  OPPENHEIMER G. and WEIZER N.  
"Resource Management for a Medium Scale Time-Sharing Operating System"  
Communications of the ACM Vol.11, No.5 May, 1968 pp 313-322

OWEN 73  OWEN G.J.  
"ROLLBACK - A Method for System Recovery"  

OWEN 75  OWEN G.J.  
"Hardware vs. Software: Techniques for Obtaining the Optimum Balance"  
Conf. on Software Eng for Telecomm. Switching Systems  
PALME J.
"A Reader Comments on Henry Kleine's Initial Survey"
Simulation, August, 1971 pp 94

PASAHOW E.J.
"Testing Real-time Systems with Simulation"
Proc. ACM-SIGSIM Symp. on Simulation of Computer Systems
Gaithersburg, USA 19-20 June, 1973 pp 40-45

PETRONE L.
"On a Simulation Language Completely Defined onto the
Programming Language PL/1"
Proc. IFIP Conf. on Simulation Programming Languages Oslo
Norway, May, 1967 pp 61-85

PIENE J.O.
"Simulation of Computer Systems - A Case Study on the
CDC 6000/SCOPE System"

POVEY J.A. and COLE A.C.
"The Use of Electronic Digital Computers for Telephone
Traffic Engineering"
P.O.E.E.J. NO. 58 1965 pp 203-209

PRITSKER A.A. and KIVIAT P.J.
"Simulation with GASP II"

REINO A.M. and FRED C.D.
"Simulation Design of a Multiprocessing System"

REITMAN J.
"Computer Simulation Applications"
John Wiley & Sons Inc. 1971

SALIH A.M.
"Digital Computer Simulation as a Tool for Evaluating the
Performance of SPC Telephone Exchanges : Experience with a Model
for the GEC Mark II.BL and Suggestions for Future Work."

SALIH A.M.
"A Simulation Model for the DMNSC Exchange"

SALIH A.M., McQUADE E., CUNNINGHAM R.J., GRAY H. and OWEN G.J.
"Multi-Level Process Simulation as a Design Aid for SPC Switching

SALTZER J.H.
"A Simple Linear Model of Demand Paging Performance"
Comm. of the ACM Vol.17, No. 4, April, 1974 pp 181-186
SCHM 79 SCHMIDT C.E., RABINER L.R. and BERKLEY D.A.  
"A Digital Simulation of the Telephone System"  
BSTJ Vol. 58, No.4, 1979 pp 839-855

SEDG 70 SEDGWICK R., STONE R. & MACDONALD J.N.  
"SPY - A program to Monitor OS/360"  

SHAN 73 SHANNON R.E. & WAYATT M.W.  
"Discrete-Simulation Language Users' Survey Revisited"  
Simulation Vol.19 May, 1973 pp 168-169

SHER 72 SHERMAN S., BASKET F. & BROWNE J.C.  
"Trace-Driven Modelling and Analysis of CPU Scheduling in a Multiprogramming System"  

SIMU 79 SIMULATION NEWSLETTER  
"Language Comparison - State-of-the-Art"  
No.2, Winter 1979, Publ by Centre in Simulation, University of Lancaster, Lancaster, U.K.

SIMU 75 "SIMULA USERS GUIDE"  
IBM/System/360 Revised Ed. 1975, NCC Pub. No.S-24-1

SINC 80 SINCLAIR T.M.  
"PPU Model : Functional Description"  

SPIE 72 SPIEGEL M.R.  
"Statistics"  
Schaum's Outline Series, McGraw-Hill, 1972

STAN 68 STANLEY W.I & HERTEL H.F.  

SVOB 76 SVOBODOVA L.  

SWAM 78 SWAMINATHAN K.  
"Computers in Telephone Districts"  
Telecommunications (India) Vol.28, No.1, June, 1978 pp 5-7

TEIC 67 TEICKROW D., LUBIN J.F. & TRUIT T.D.  
"Computer Simulation - Discussion of the Technique & Comparison of Languages " - Simulation Vol.19, 1967 pp 181-190

TEOR 73 TEOREY J.T. & MERTEN A.G.  
"Consideration of the Level of Detail in Simulation"  
Proc. ACM-SIGSIM Symp. - Simulation of Computer Systems, June, 73

THOM 79 THOMAS J.C. & PAUL J.H.  
"Analysis and Simulation of No.4 ESS Network Performance"  

TIPP 77 TIPPLER J.  
"Review of Switching Requirements - Role of SPC"  
IEE Vac. School on Stored Program Control of Telecomm. Switching

R15
Systems. 11-16 Sept. 1977 University of Essex.

TIPP 79 TIPPLER J.
"Architecture of System X : Part 1 - An Introduction to the System X family"
POEEJ, Vol.72, Oct. 1979 pp 138-141

TOCK 60 TOCKER K.D. and OWEN D.G.
"The Automatic Programming of Simulations"
Proc. of 2nd International Conf. on Oper. Res. pp 50-68

TOCK 65 TOCKER K.D.
"Review of Simulation Languages"

TOGN 72 TOGNETTI K.P. and BRETT C.
"SIMSCRIPT II and SIMULA 67 - A Comparison"

TSAO 72 TSAO R.F. and MARGOLIN B.H.
"A Multi-Factor Paging Experiment: II Statistical Methodology"

UNGE 72 UNGER B.W.
"A Simulation Model for Evaluating Computing Resource Architecture and Management"

UNGE 75 UNGER B.W.
"Validation of a Computer Resource Allocation Model"
Summer Computer Simulation Conference. San Francisco July 1975

UNGE 77 UNGER B.W.
"A Computer Resource Allocation Model with some Measured and Simulation Results"
IEEE Trans. on Computers, Vol.C-26, No.3 March 77 pp 243-259

VAUC 71 VAUCHER J.G.
"Simulation Data Structures Using SIMULA 67"
Winter Simulation Conf. 1971 pp 255-260

VAUC 73 VAUCHER J.G.
"WAIT-UNTIL' Algorithms for General Purpose Simulation Languages"
Proc. Winter Simulation Conf. 1973 pp 177-183

VIRJ 72 VIRJO A.
"A Comparison of Some Discrete Event Simulation Languages"
NORDATA 72 Conf., Helsinki, 1972

WARD 69 WARD M. & ACKROYD E.
"A Multiprocessor Control Unit for a Stored Programme Switching System"

WARD 72A WARD M.
"The Mark II Communications Processor"
GEC Telecommunications Journal No.39 pp 13-15 1972
WARD 72B  WARD M. and NISSEN J.C.D.
"Software Security in a Stored Program Controlled Switching System"

WEAT 73  WEATHERBY J.E.
"Simulation in the Evaluation of an Executive System Design"

WILL 74  WILLIS R.R.
"Structured Model Development Techniques"
Proc. ACM-SIGSIM Symp. on Simulation of Computer Systems 4-6, June, 1974, pp 132-135

WILS 78  WILSON J.C.T.
"Research in the Centre"
Simulation Newsletter, Centre In Simulation, University of Lancaster, Lancaster, U.K.

YAN 78  YAN J.
"Simulation of a Business Communication System Switching Network"
Proc. Winter Simulation Conf. 4-6 Dec. 1978 pp 751-757

ZURC 68  ZURCHER F.W. and RANDEL B.
"Iterative Multi-level Modelling - A Methodology for Computer System Design"