THE ADDITION OF MEMORY MANAGEMENT FACILITIES

TO A REAL-TIME OPERATING SYSTEM

PRESENTED IN PART FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MSc IN ENGINEERING IN THE FACULTY OF ELECTRICAL ENGINEERING, UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG.

MARCH 1984.

C CHARALAMBOUS

ABSTRACT

This dissertation outlines the addition of memory management facilities to the Standard Multitasking Operating System (SMT), a real-time kernel which provides a multitasking environement for application tasks written in the high-level language RTL/2.

The concepts and design philosophy used to achieve this objective as well as the underlying concepts of real-time programming are discussed.

An implementation for the standard memory management unit of the Digital Equipment Corporation PDP-11 series is described and an evaluation test in an on-line process control application is discussed.

DECLARATION

I hereby declare that this thesis is my own work and that no part or substance of it has previously been submitted for a degree at any University.

C Charalambous

March, 1984

PREFACE

During the period 1981-1982 the author was involved in the design and commissioning of the instrumentation for two large scale chemical process plants belonging to AECl Limited. This experience was followed by a year (1983) during which the author carried out extensive work on the SMT operating system.

The first two years gave the author experience in the area of process control, process characteristics, the limitation of standard instrumentation and the conceptual necessity for real-time process control computing. Installed as part of the second instrumentation project, was a large supervisory computer which performed the control of a fluidised bed reactor as well as providing the sophisticated presentation of process data.

The author was subsequently involved in the enhancement of an existing computer system used to present to an operator, the status and yield of mercury cells on a chlorine plant. At the start of the project, it was discovered that the operating system (SMT) used to provide the multitasking software environement, had reached the limits of its memory addressing capability. This implied that the necessary modifications could not be implemented in the computer system. This limitation had simultaneously been reached on other computer systems using SMT within AECl Ltd, as well as at Imperial Chemical Industries Ltd in the UK. The head of the process computing group of AECl Ltd decided that the project should be split into two parts, namely the enhancement of SMT to include memory management facilities, to be carried out by the author and secondly the implementation of the application modifications for chlorine cell control by a second engineer.

This dissertation describes the concepts and design philosophy underlying the addition of memory management facilities to SMT.

The work described in this dissertation was presented as a paper at the RTL/2 USERS GROUP of South Africa 1983 conference.

ACKNOWLEDGEMENTS

I wish to acknowledge the assistance and guidance given to me by Dr A D Heher during the project period. His wisdom, patience and guidance has enabled me to successfully complete this work.

I wish to thank Professor I MacLeod for his guidance while introducing me into the world of real-time computer programming.

I also wish to thank Mr G Brown and AECL Ltd for their permission to submit this work as a thesis to complete my Masters degree in Engineering.

I finally wish to thank my wife Daphne who has encouraged me during the duration of the project.

INDEX

APPENDICES

NOMENCLATURE

SMT - STANDARD MULTITASKING SYSTEM.

DSMT - DEVELOPED SMT.

MMU - MEMORY MANAGEMENT UNIT.

APR - ACTIVE PAGE REGISTER.

PAR - PAGE ADDRESS REGISTER.

PDR - PAGE DESCRIPTOR REGISTER.

PA - PHYSICAL ADDRESS.

VA - VIRTUAL ADDRESS.

DEC - DIGITAL ELECTRONIC COMPANY.

RTE - REAL-TIME EXECUTIVE.

CHAPTER 1

INTRODUCTION AND STATEMENT OF PROBLEM

1.1 INTRODUCTION

This dissertation outlines the addition of memory management facilities to the Standard Multitasking operating system (SMT). SMT was designed at Imperial Chemical Industries (ICI) Limited in the United Kingdom in 1977. The operating system was specifically designed for real-time multitasking process control systems.

SMT was designed so that small computer applications can be implemented in a simple manner. The operating system is compact in that the complete operating system occupies 4 k-words of memory. A kernel of this size is ideal for single board computers where the complete syster may reside in 32k-words of memory. SMT is also efficient in that it is written in RTL/2 specifically to support multitasking application tasks written in RTL/2. This feature is extremely useful, especially when considering the input/output method (streaming) of transmitting data in RTL/2 systems. RTL/2 is a computer language designed by ICI Ltd in the early 1970s, specifically to cater for the needs of real-time programming in process control computing. The language has a number of features which make it suitable for this purpose. The primary advantages of the language is that it is secure at system generation time as well as at run-time. Other important features of RTL/2 are its simplicity, the compact code produced by available compilers and also its portability.

SMT will be described as a kernel in the remainder of this dissertation since SMT comprises the basic core of an operating system. SMT does not in its standard form, have file handling constructs and does not interface with a disc drive. The SMT kernel and all application tasks reside permanently in memory.

The SMT kernel is written predominanently in RTL/2 except for one module, written in macro-code, which contains all the procedures which interface with the processor. The advantage of having SMT written in RTL/2 is that it is easily transportable between different makes of processor. This feature also makes the production of an RTL/2 environement for user tasks easy to achieve.

A feature of SMT is its portability. SMT has been easily moved to a variety of processors since the percentage of the kernel written in machine dependent macro-code is small. The work in this thesis is confined to SMT running on the 16-bit PDP-11 range of processors manufactured by the DIGITAL EQUIPMENT CORPORATION. On these computers, SMT based systems were confined to fit within a 32k-word memory addressing range. With the increased functional requirements of individual process control computers based on SMT, it has been found that this addressing range is insufficient to contain all the tasks required.

This requirement for increased program size prompted the initiation of the project reported in this dissertation. The aim of the project was to enhance SMT to efficiently utilize the memory management unit on PDP processor boards. By doing this, the addressing range of the system could be increased to 128k words or 2M words - depending on whether an 18 or 22 bit addressing bus is used.

The SMT kernel is extensively used within AECI Limited and the success of the work has opened up new possibilities for the use of SMT.

The remainder of this chapter lays down the specifications for the memory managed SMT. The procedure used to reach a solution and an outline of the dissertation is given in section 1.3.

1.2 SPECIFICATIONS

The memory managed SMT was designed with the prime objective of satisfying all the specifications laid out below.

1. The present restriction of using 64k Bytes of memory should be eliminated.

2. The memory management unit on the PDP-11 minicomputer system should be utilized to provide either 18 bit or 22 bit physical address space addressing.

3. At present, individual tasks cannot be linked without rebuilding the complete system image. This limitation should be eliminated.

4. Individual tasks cannot be loaded into memory from a mass storage medium like a magnetic tape cartridge. This restriction should be eliminated.

5. System overhead due to the additional memory management facilities should be as small as possible, so that the overall system performance is not significantly degraded.

6. The operation of the SMT environement should not be jeopardised.

7. The method used to implement SMT based system should remain as simple as possible.

1.3 OUTLINE OF METHOD USED TO REACH A SOLUTION

The project was approached in a methodical manner so that a final solution could be attained in the minimum amount of time and a full understanding of the problem and the SMT kernel would be assured. A basic requirement for the proposed solution was that it should entail an integrated approach to aspects such as software documentation, user friendliness and that an RSX11M development environement should be used (since this is used within AECL Ltd), in addition to the real-time kernel enhancements specified in section 1.2.

The major stages of the work were as follows:

1. A study was carried out to determine the full implications of the requirements of this project.

2. A literature survey was carried out, that is a search was made into all available books and computer journals for articles and papers relating to the subject (See Section 1.4).

3. The existing kernel was then studied to understand the modus operandi of all the procedures and also all the data structures.

4. A solution was then proposed, implemented and tested on a host machine.

5. An existing system running under the original SMT kernel was modified to operate on the new memory managed version of SMT. This system was then installed on a plant computer.

6. A technical manual describing how to build systems to run under the new kernel was written.

The first 4 parts are explained in detail in this dissertation, whereas 5 and 6 cover implementation details and are not covered in this report. For further information Reference 2.8 should be consulted.

Chapter 2 gives a brief overview of real-time systems, their philosophies and implementations together with an examination of the literature on the subject.

In Chapter 3, the existing kernel (SMT) is examined to fully understand its mode of operation. An overview of RTL/2 based operating systems is given and the advantages of having SMT written in RTL/2 are discussed.

In Chapter 4, the philosophy used to implement memory management is presented. A detailed explanation of the implications of the proposed modifications is made.

Chapter 5 presents a summary of the system with particular emphasis on user experience in installing the new kernel into an existing process control system. This chapter also discusses possible further work that should be done to SMT.

1.4 LITERATURE REVIEW

In trying to find information on real-time operating systems and real-time programming it was soon found that very few books on this subject exist. There are however many papers on the subject, each detailing a certain aspect of real-time computing.

The paper by Kylstra (1.1) presents a clear model for real-time sytems. His model outlines the transition of a task through the various states that it may occupy at any point in time. Kylstra then outlines how a real-time operating system should be able to support multitasking system. This paper gave the author the insight into real-time programming.

The paper by Anderson (2.2) , presents in a simple manner, the purpose of having an operating system, the concepts of tasks (or processes), the concepts of re-entrancy and the various elements which together make up an operating system. This paper is recommended to any person who wishes to understand an operating system conceptially, without wishing to know implementation details.

J.G.P. Barnes, a leading authority on real-time computing, has published a number of papers on the subject of real-time computing and on RTL/2. His paper "The use of RTL/2 for Multitasking" isolates the key concepts underlying real-time software for process control. This paper also outlines the idea of input/output data streaming and the problems associated with task scheduling. As the designer of RTL/2, any user of RTL/2 should have access to all of Barnes's papers which detail the aims and objectives of RTL/2 and the subleties of real-time multitasking.

It is felt that the reference section of this report, summarizes the key references concerning the design and operation of practical real-time kernels.

On the general principles of operating systems papers 1.1, 2.1 and 4.1 were of particular interest. These papers led the author to a deeper understanding of operating systems, their functions, objectives and structure.

On the subject of real-time programming techniques, references 3.1 and 6.2 were of particular interest and usefullness.

CHAPTER 2

REAL-TIME CONCEPTS

2.1 REAL-TIME COMPUTER SYSTEMS

There are many classes of computer system. This report is specifically concerned with real-time computer systems as used in the process control industry. These computer systems are used for the gathering of plant information and the control of plant variables.

Real-time computer control systems operate in a time-dependent environment in which the controlling system should be sufficiently responsive to changes that are occurring in the process that is being controlled so that the future behaviour of the process can be successfully modified.

The computer system should appear to react instantaneously to any stimulus from the environment. This has to occur in order to avoid the loss of information concerning the changing environement. If the computer does not accept the process information(data) fast enough, the data in the computer will be out of date and therefore useless.

The design of software to run on real-time systems requires an intimate knowledge of the process being controlled, the user requirements of the computer and the mechanisms of real-time processing.

This chapter will outline the various accepted concepts relevant to real-time computing.

2.2 KERNELS AND OPERATING SYSTEMS.

Modern general purpose operating systems have many facilities which ease the work of software engineers in attaining their goals. These operating systems, contain structures which enable tasks created to be incrementally added to the system as soon as the task image has been created. Such operating systems have structures to handle file manipulation on secondary storage media such as discs. These operating systems usually communicate with the hardware via a layer of software different to the high level language being used.

In contrast, kernels such as SMT operate with a minimal set of features. These kernels normally occupy a small amount of memory. Kernels are ideal for single board computers or microprocessor based systems. Thus a kernel is a subset of a larger operating system. Kernels are often faster than larger operating systems since they operate very close to the processor hardware and extra software layers are absent.

For this reason SMT will be referred to as a kernel rather than an operating system in the remainder of this dissertation.

2.3 REAL-TIME PRIMITIVES

Creating tasks to run in a real-time environement requires certain special features, of prime importance in real-time programming are the following:

1. The ability for tasks to be able to communicate with one another and to synchronize with one another.
2. The ability of tasks to communicate with I/O interface hardware with

the highest efficiency.

3. The ability of tasks to gain exclusive access to shared resources.
4. The ability to be time dependent/controlled (timing).
5. Interrupt handling mechanism to control the interaction with the I/O hardware.
6. Error recovery and reporting mechanisms used to protect the system from errors generated by user tasks.

A real-time primitive is an operating system call that conceptually performs one indivisible operation when seen from the point of view of the issuing program. The real-time primitives form an interface between the user tasks and the executive.

In the SMT kernel discussed in this dissertation, this interface is formed by a combination of the executive calls such as WAIT, DELAY, SET and others, together with the hardware (H) - tasks. H-tasks are tasks written to service interrupts from the hardware and which operate in a non RTL/2 environement.

This software layer enhances the raw hardware, by adding additional functions needed for real-time computing.

2.4 TASKS, PROCEDURES AND MULTITASKING

This section identifies the difference between tasking, procedures and multitasking. In many of the papers consulted by the author, the term task as used in RTL/2 based systems is used interchangeably with the term process. In this dissertation the term task is used.

A task is the conceptual thread of control defining an active program in a computer system. It is the execution of a sequential program to perform one self contained function. The application software in a complex real-time computer system is organized as a set of tasks. These tasks execute "concurrently" and co-operate to realize the overall system function. Apparent concurrency is provided by a multi-tasking executive as described below.

The task is thus the highest form of control in the hierachy of executable statements. A task may call procedures and a procedure may execute a statement, but the reverse is not true.

As an example, in a process control application, the computer system may contain one task controlling input/output, another task doing control and optimisation calculations and a third operator interface routines.

Real-time tasks may be programmed in a variety of ways. The task may be dormant until some significant event restarts it, or the task may execute in some cyclic fashion, with a time delay between executions. At any instant of time a task may occupy one of the states shown in Figure 2.1. The state it occupies is dependent on the action of the task, the priority of the task and the state of the computer system.

A procedure is a collection of executable statements, with the objective of performing one function. A procedure may for example add two numbers or calculate the elapsed time since some event or it may perform some algorithm for example, a PID calculation for a process control loop.

A procedure is passive and merely contains a number of related executable statements. The procedure is an important tool for the structured design

of tasks and the management of complexity.

Multitasking refers to the implementation of the concept of tasking, by multiplexing a single processor amongst multiple tasks. All the tasks are apparently running in parallel to one another, but only one is in the "running" state as shown in Figure 2.1 while the others occupy one of the other states.

Multitasking enables the partitioning of the system function into a number of individual tasks, each responsible for a single function, for example, one task may do I/O operations only, another may be doing computations only while a third may be communicating with an operator.

Multitasking also permits the resources to be used to their fullest. For example, the processor may execute a second task while the first is busy communicating with the interface hardware.

For multitasking to be performed, an executive kernel must exist to manage these tasks. Real-time operating systems are special and these are discussed in Section 2.9.

2.5 INTERRUPTS

An interrupt may be defined as the suspension of the running task due to a significant event occuring in the hardware or in a task elsewhere in the computer system. When an interrupt occurs, the context of the interrupted task has to be saved and an interrupt handling procedure is to be called to service the interrupt. There are three types of interrupts, namely

1. Clock Interrupts
2. Hardware Interrupts
3. Software Interrupts

In time-critical situations it is imperative that the operating system does not inhibit interrupts for long intervals. If interrupts are inhibitted, an interrupt from the system hardware timer used to update the time parameters may be missed and thus the tables may fall out of step with the true time. Similarly interrupts from devices requiring to be serviced by the processor may be missed, resulting in the information (data or statii) coming from the plant, or environement, being lost.

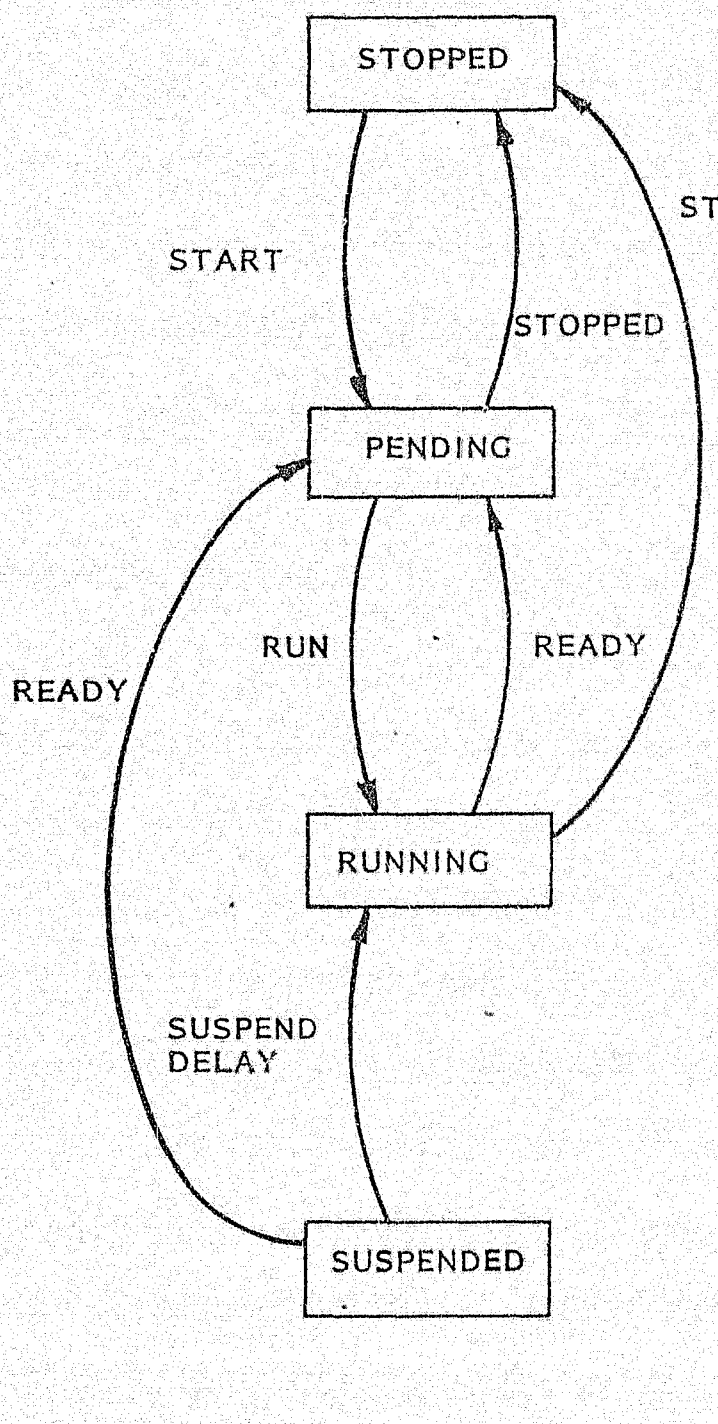
In a real-time computer system the updating of the present time clock in the kernel is critical, since in a real-time system it is the time dimension which controls the state of the processed tasks. In most real-time systems the clock interrupts the processor at a fixed time interval. The time spent servicing this interrupt should be as short as possible since the clock interrupts the processor frequently and could consume a great deal of processor time. The operating system uses this time interrupt to update all the time-dependent system tables. An example of this could be a table of all the tasks being delayed.

A hardware interrupt could originate from any I/O interface device connected to the processor. Any device that wishes to give data or status information to the computer system, first flags the processor for attention, and then the kernel executes the "servicing routine" which controls the protocol of the communication with the device.

A "software interrupt" , or "significant event" is a term used when a task reaches a stage when it wishes to stop execution, be delayed or it is awaiting to access an input/output facility. In such a case the task

Figure 2.1

STATE TRANSITION DIAGRAM FOR TASKS



TASK - STATUS NOT OK
STOPPED BY OPERATOR
OR ANOTHER TASK.

TASK - STATUS OK
NOT THE HIGHEST PRIORITY

TASK - HIGHEST PRIORITY
STATUS OK

TASK - STATUS NOT OK, DUE TO:-
WAITING FOR AN EVENT
FACILITY NOT AVAILABLE
DELAY NOT COMPLETED

cannot execute further and thus another task should run, i.e. a task change should occur. This task change is a form of an interrupt but is controlled by task communications and resource allocation.

2.6 MEMORY MANAGEMENT

2.6.1 Introduction

A problem encountered in many computers is that a fixed word-length, limits the addressing range to 2^n, where n is the word length. That is for an 8 bit computer system, a memory of 0 - 256 bytes can be addressed directly or 65536 bytes for a 16 bit machine.

To overcome this limitation, the concept of a virtual address space was introduced. A virtual address space is that part of the total physical addressing range which is in scope (ie directly addressable) at any instant of a task's execution. The total physical addressing space can now be considerably larger but at any one instant, a task can only access a subset of the total physical memory.

In the future, processors with larger word lengths will not have this problem, since the virtual address space will be so large that memory management facilities will not be neccesary.

2.6.2 Paging System

A technique commonly used is memory paging. The memory addressing area is segmented into a number of memory pages, with each page having a certain maximum length.

2.6.3 Memory Management Hardware Support

The hardware device used to implement paging is commonly called a "memory management unit". The primary function of this device is to transform the fixed virtual word-length of the processor to a larger word-length used to address the physical memory.

2.7 STACK-BASED MICROCOMPUTER SYSTEMS

2.7.1 Introduction

A real-time language or system based on the stack method of operation offers many advantages over other methods.

A system which maintains a separate stack for each task enables re-entrancy of procedures to be achieved. Re-entrancy is possible when the local variables of a procedure are not stored in the procedure code area but rather in an area associated with the calling task. A method for achieving this is by the use of a stack.

2.7.2 Stacks

A stack is an area in memory which is dynamically allocated to store local variables and intermediate results of a procedure. In RTL/2 the link cell which stores the program counter of the calling procedure is also stored on the stack. To be fully specified each task, in RTL/2 has to have its own stack and its own 'base

procedure'.

In fact Barnes in Ref 3.1 stated that "In fact the stack can be seen as the concrete personification of the abstract task". Having a stack enables procedures to be re-entrant since all the local variables are stored on the stack.

2.7.3 Re-entrancy

In a multitasking environment re-entrancy is a desirable feature since many tasks, each in a different state of execution may use the same procedure simultaneously. In fact, a procedure may call itself.

A re-entrant procedure contains code that does not write or alter data globally. Such a procedure only manipulates data that has been passed to it by the calling procedure and/or data declared by the procedure itself, both of which are stored on the task stack.

Re-entrancy in a stack based language may be implemented fairly easily, but with certain precautions. All the data used in a procedure should be stored on the stack. In the RTL/2 language, for reasons of execution time predictability only scalar local data can be stored on the stack. This implies that procedures using arrays and accessing them directly are not re-entrant. If the procedure is passed a pointer to the array, the procedure becomes re-entrant. (The concepts of re-entrancy in RTL/2 is explained well in Reference 3.1).

2.8 SCHEDULING OF TASKS

The heart of a real-time kernel is the scheduler. The scheduler decides which task should execute next. In other words the scheduler obeys a scheduling algorithm which is a set of rules that determine what task is to be executed at a particular moment. The scheduler also controls the saving and swopping of task environments at task change time (ie despatching).

The scheduling of tasks by the operating system may be done in a variety of ways. Of importance are the following methods:

2.8.1 Run to Completion Techniques

Each task runs from start to finish where upon the following task will run. In a real-time environment this is not acceptible since tasks may be interdependent and a task may never run to completion, due to intercommunication between tasks.

2.8.2 Time Slicing on a Round Robin Basis

This method utilizes the technique of allocating a fixed time slice to each task. Each task created is put into a circular list. The execution of the tasks occur consecutively as laid out in the list. The parameter used to determine when a task swop is to occur, is the elapsed time of execution. (See Figure 2.2).

This method does not take into account the relative priority of all the tasks. Each task has the same priority as all the others and its turn to execute is purely dependent on where the task is in the

Figure 2.2

TIME - SLICE EXECUTION

LIST

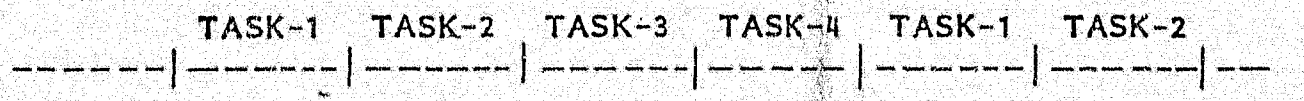
TASK-1

TASK-2

TASK-3

TASK-4

EXECUTION



circular list.

If a task is not in an executable state, that is it is waiting for a significant event to occur, it is still called, but returns control to the scheduler immediately. This useless call is waste full of CPU time and thus the CPU is not used optimally.

A time slice task will execute until either:

1. it is waiting for I/O or some significant event to occur or

2. it's time slice gets exhausted.

Another disadvantage of the time slicing technique is the waiting for resources. If a given task wishes to access some resource or event, it has to wait its turn to execute.

2.8.3 Event Driven Scheduling

This form of scheduling is also commonly referred to as priority scheduling or pre-emptive scheduling.

Using this technique, each task is assigned a relative priority. The order of priority is obviously in urgency order, background tasks are normally assigned low priority whereas foreground tasks are assigned higher priorities. That is, tasks that are time critical should have higher priorities, so that they execute more often than background tasks.

The operating system employing this method will always try and execute the highest priority task.

2.9 REAL-TIME COMPUTER OPERATING SYSTEMS

A real-time kernel should provide all the facilities to control a multitasking environemnt. The main purpose of any real-time kernel are the following:

1. The allocation of resources e.g. CPU, device drivers, memory etc.

2. The scheduling of tasks using one of the methods outlined in Section 2.8 (Task Management).

3. Providing the inter-task communication channels.

4. The support of multitasking, thus maximizing the usage of all system resources.

5. An interrupt handling mechanism.

6. Time keeping and updating mechanism to update time-dependent system tables.

7. Error recovery mechanisms to avoid the total collapse of the system.

The manner in which a kernel implements these facilities gives an indication on its power, flexibility and efficiency. Furthermore, a ker- should implement these facilities without consuming too much of the

processor time (processor overhead). If this is not the case, a great deal of time is spent servicing all these facilities rather than processing the user tasks.

2.10 THE RTL/2 LANGUAGE

The RTL/2 language was designed specifically to cater for the needs of real-time process control computing.

The language was defined in 1972 within Imperial Chemical Industries Ltd in the UK to satisfy the needs of real-time computing, at a time when computer languages such as FORTRAN were available only. RTL/2 provides features for the design of real-time systems which even at present cannot be easily matched by many languages. The language provides features such as efficiency, security, portability flexibility and also productivity.

The language is secure due to the following reasons,

1. Secure compile time checks and compiler virtually bug free.

2. Strong typing checks are forced on the programmer.

3. Run-time checks and error handling mechanism is provided.

4. Checks on the scope limitation of the GOTO statement at compile time.

The language forces the user to obey strong typing rules. Included in this, is the specification and initialisation of all data at compile time, thus avoiding tasks starting up with unknown quantities in data structures. The scope limitation when using LABELS in source code is also rigorously checked. The compiler itself is written in RTL/2. The high quality compiler executes rigorous compile time checks on the source program which results in predictable task execution at run-time. The compiler has been approved as a British Standard ensuring uniform code production where ever RTL/2 is used.

At run-time the language continuously executes stack and array bound checks, trapping any unpredictable data structure access. If an error is detected, the error handling mechanisms ensure that the error is recorded for later analysis.

The language is portable, since one of the language definition objectives was to make it processor independent. The only assumption made, is that RTL/2 will run on a processor with a minimum word length of 16 bits. To date RTL/2 has been implemented on at least 30 different types of processors.

The language is flexible since it supports all the common primitive data structures namely bytes, fractions, integers and reals. Also, the MODE declaration allows the user, to declare his own data structure in the form of a record.

User productivity is increased, since RTL/2 being a high level language enables the use of all structured programming techniques. Another point to note is that the full language can be learnt in 3-4 days.

Computer systems written in the high-level language RTL/2 are easy to maintain. The use of machine specific code, which may obscure the logic of a user task, is eliminated. This fact also protects system managers against staff mobility etc.

A full description of the language cannot be given in a few pages. If any one wishes to delve deaper into the language, References 8.3, 8.5 and 8.6 are particularly recommended.

A few features which should be mentioned briefly, are the following:

The language does not cater for dynamic data storage allocation. The main reason for doing this is for execution time predictability of the executing programs.

The "GOTO" function is provided but may only be used internal to a block. This avoids unconditional jumps among code which may corrupt the stack environment. The language is amenable to the concept of structured programming in that the language is block orientated.

The RTL/2 language itself does not explicitly define input/output mechanisms, operating system features and so on. In certain areas , guide lines have been given, on how to stream data, to and from tasks written in RTL/2. (see REF 8.7) This has been done specifically to keep the language portable.

2.11 INTERTASK COMMUNICATION

Signalling between a hardware device and the processor takes place via interrupts. In a similar way in many operating systems, communication amongst tasks is supported by a resource called a software event.

A software event is a flag, but the mechanism used to set and reset this flag, gives the event its special feature. Each event is identified, by a number and can occupy only one of two states, namely "SET" or "RESET".

In RTL/2 based systems, an event may be SET by one task and received by one or more other tasks. The tasks "WAITING" for this event flag are removed from the WAITING list. These tasks then reset their copy of the flag and begin execution.

In a similar way servicing routines, operating very close to the processor hardware, use events to communicate with tasks working in an RTL/2 environement.

A software event is an executive resource and therefore their usage should be done with care, specifically ensuring that event numbers are not duplicated.

CHAPTER 3

SMT OPERATING SYSTEM DESCRIPTION

3.1 INTRODUCTION

SMT is a small real-time kernel used to control the operation of a set of application tasks written in RTL/2.

It is suitable for those applications which do not require the complexity of a larger operating system. SMT provide:

- inter-task signalling via event flags
- the scheduling of access to shared resources via facility flags
- an interrupt handling mechanism
- priority scheduling amongst tasks
- Input/output formatting procedures
- Error recovery and reporting mechanism

Prior to the modifications carried out, SMT had several limitations, namely:

- the system was limited to 32k-words of memory
- individual task images could not be loaded seperately
- the memory management facility on DEC PDP-11 computers could not be used

The need to overcome these limitations resulted in the project reported in this dissertation.

3.2 REAL-TIME PRIMITIVES

SMT implements the primitives that lie at the heart of any real-time kernel. SMT contains:

1. An interrupt handling system which enables 32 priority levels of interrupts to be implemented.

2. Inter-task and device-driver to task communication is provided via "EVENT" flags. Events are flags which occupy one of two states namely "SET" or "RESET".

3. The scheduling of access to shared resources are controlled by the use of "FACILITY" flags.

4. Delayed scheduling of tasks based on the real-time clock.

The task scheduling mechanism in SMT is based on the static priority scheduling of tasks.

3.3 RTL/2 ENVIRONMENT CREATION

The method that is used by SMT to create an RTL/2 environment is described in this section. In SMT the tasks and procedures that form the layer between RTL/2 tasks and the hardware are known as H-tasks or Hardware Tasks. S-Tasks or Software tasks run in a pure RTL/2 environment created by the H-tasks.

These two task types have specific functions to perform and each runs in a different environment.

3.3.1 H-tasks

These tasks respond to hardware interrupts. They are interrupt handlers that are used to service the interrupts from hardware devices or the system clock. H-tasks form a layer, with the kernel that lies between the user software and the hardware (see Figure 3.1).

The scheduling of H-tasks is performed by the 32 level PDP-11 interrupt system. These tasks are all mutually exclusive in that only one H-task can run at a time.

In SMT, once an interrupt has been signalled to the processor via the hardware structure, the S-task executing is suspended. The H-task used to service the interrupt is then given the highest priority the processor has, since one of the design features of SMT, is that the H-task executing will run to completion before another H-task can execute. Thus once an H-task has been entered, it has the same priority as any other H-task.

3.3.2 S-tasks

These are normal user written tasks. They are written completely in RTL/2 and run in a real-time environment. The S-tasks are scheduled by the kernel to run in priority order.

Each task is self-contained and has its own stack which is a personification of the state of the task.

S-tasks run in an environment set up by the H-tasks and the kernel.

3.4 TASK CHANGING PHILOSOPHY

3.4.1 Scheduling of S-tasks

The scheduling of tasks running under SMT is done on a priority basis. The processor will always be executing the task with the highest priority.

Whenever an operation occurs which may possibly alter the state of any task e.g. an "EVENT" is set or the task delay list is updated, a check is made whether the task running at the moment is the highest priority task in an executable state (i.e. it is not delayed or awaiting some resource). If not, the kernel will save the context of the present task and then will select which task to put in the run state using the task changing mechanism described in section 3.4.2 below.

3.4.2 The Task Changing Mechanism

A model for kernel operations was shown in Figure 2.1. The figure indicates the four states that a given user task may occupy. Only one task runs at a given time while the remaining tasks fall into one of the other states. The task to run next is the highest priority whose status is OK that is not suspended or stopped etc.

A task change can occur due to three possible reasons (See Figure

Figure 3.1

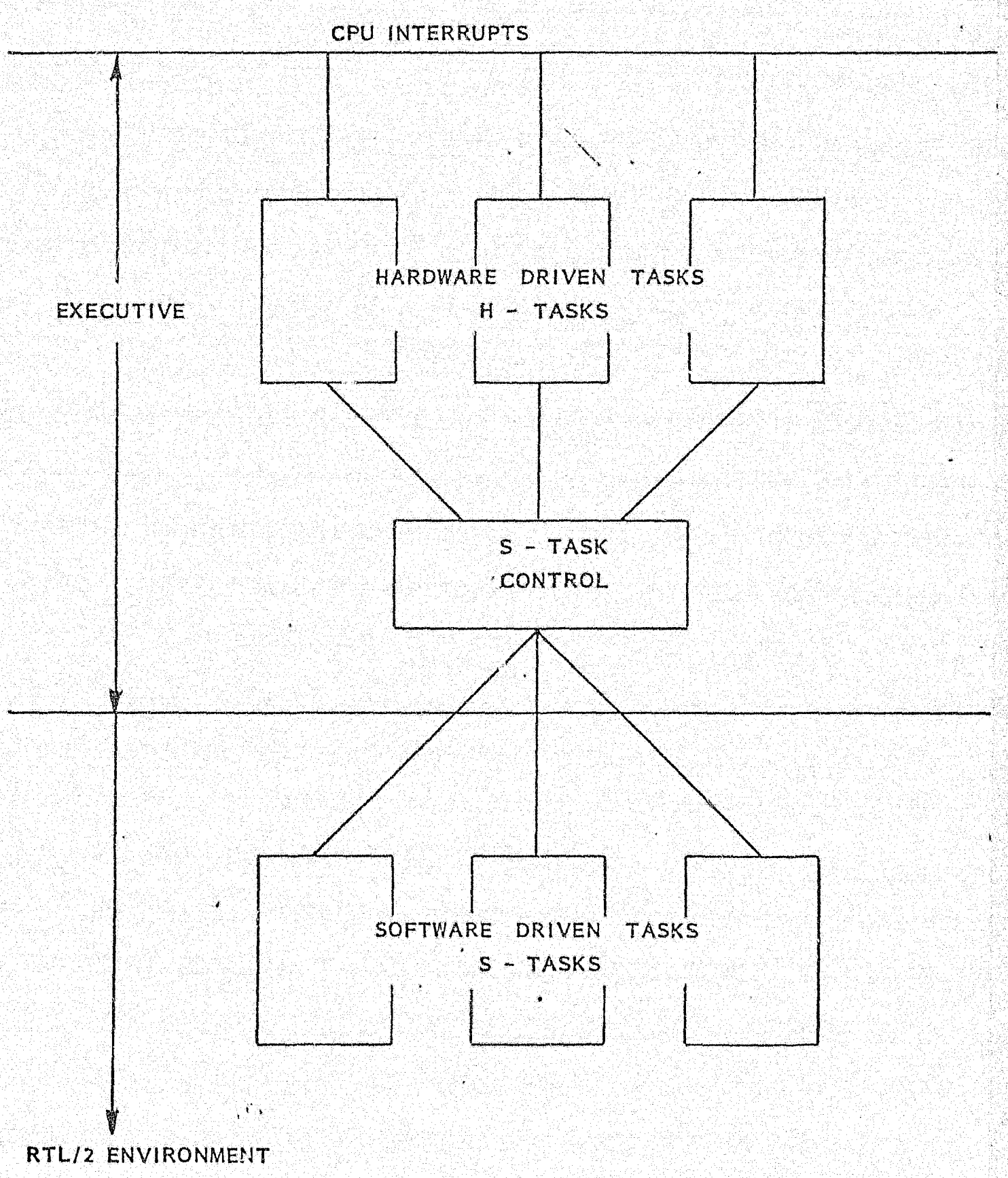
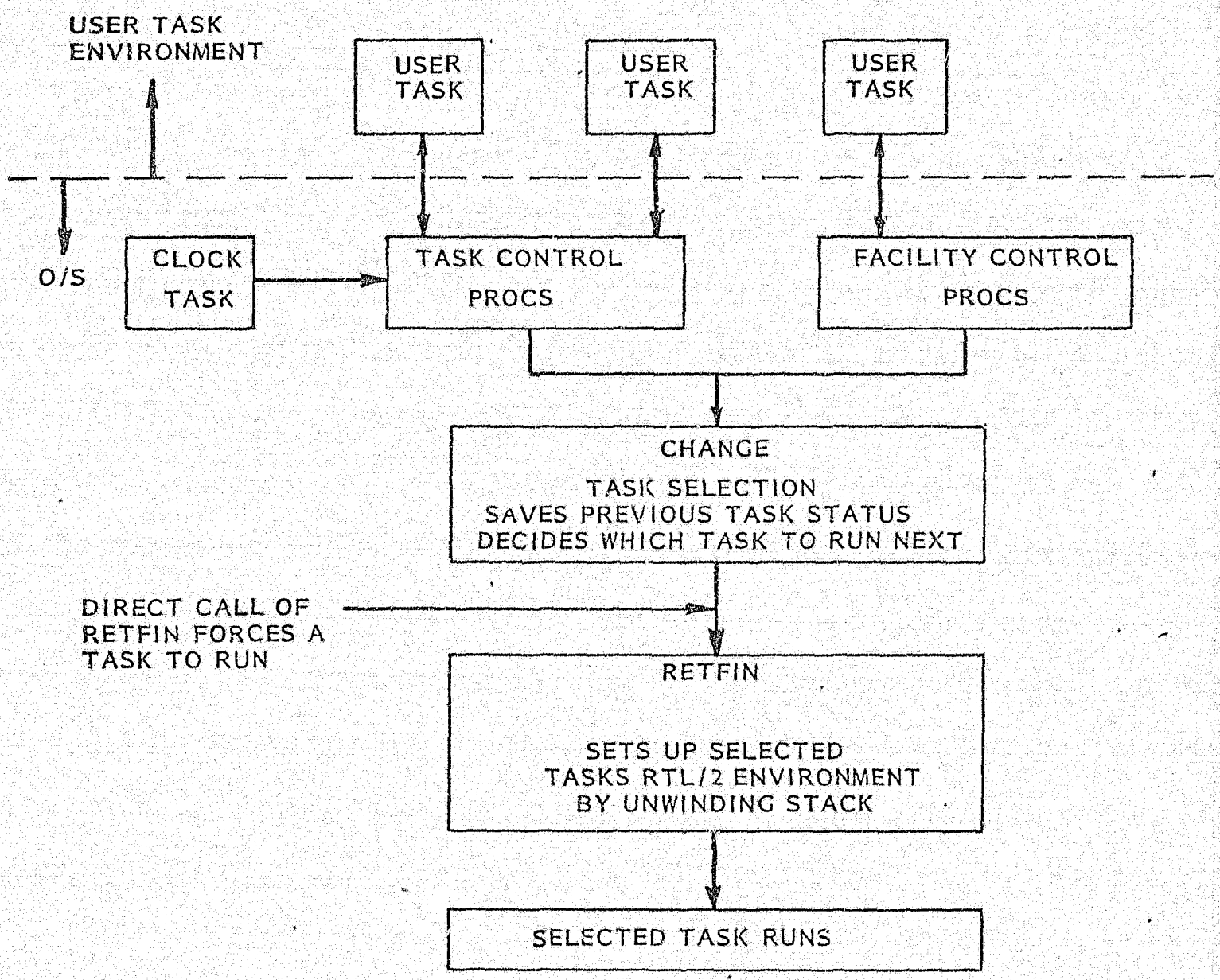

RTL/2 ENVIRONMENT CREATION
CPU INTERRUPTS
EXECUTIVE
HARDWARE DRIVEN TASKS
H - TASKS
S - TASK
CONTROL
SOFTWARE DRIVEN TASKS
S - TASKS
RTL/2 ENVIRONMENT

Figure 3.2



TASK CONTROL PROCS	FACILITY CONTROL PROCS
STOP	SECURE
START	RELEASE
DELAY	
WAIT	
WAITFOR	
SET	
RESET	

3.2):

1. A task control procedure was called.

2. A facility control procedure was called.

3. A clock interrupt occurred which forced the processor to execute the CLOCK task and then procedes to the SVC procedure CHANGE.

(see Figure 3.2, 3.3 and 3.4)

After the execution of any one of these procedures, the kernel then executes the SVC procedure called CHANGE which executes as an H-task. The procedure CHANGE merely selects which task to run after having first checked that the event queue is empty. If the event queue is not empty it calls the procedure RETEV which empties the event queue. If the event queue is empty it checks the status word called STAT of the highest priority task to see if all the conditions are OK to run it i.e. STAT=0 .... not stopped, or delayed or waiting for an event.

CHANGE continues down an array called STATADS which was set up in the startup task, SETFMQ, in priority order until it finds the highest priority task with it's STATUS word OK. CHANGE then saves the task number in a variable called CURTEX and then calls the procedure RETFIN which sets up the task environment to run the task.

A task may also be interrupted. When this occurs the task state is saved by a call of procedure PTORTL, which uses the hardware stack while executing the servicing routine. After the interrupt servicing routine has been completed, the routine may either terminate by putting an EVENT in the queue which effectively calls CHANGE or a call may be made of RETFIN which continues the execution of the interrupted task (see Figure 3.4).

Coming out of CHANGE the next task to run has been selected after looking at the status and priority of all the tasks.

On exiting from CHANGE the next procedure to execute will be RETFIN which sets up the RTL/2 environment to run the selected task i.e. moving all the task registers which were saved on the stack into the machine registers.

Once all the registers have been replaced the selected task may continue from where it left off.

3.5 CONCLUSIONS

The above concepts have been implemented in SMT and great deal of attention has been devoted to optimizing the efficiency of the implementation. This takes the form of various queuing mechanisms to minimize the amount of searching that must be done to find the next highest task which can be put in the "GO" state.

The result of these features is that SMT provided a well chosen set of programming primitives for use by application tasks. A further feature of SMT is that it has been efficiently implemented.

OPERATING SYSTEM & USER TASK INTERACTION

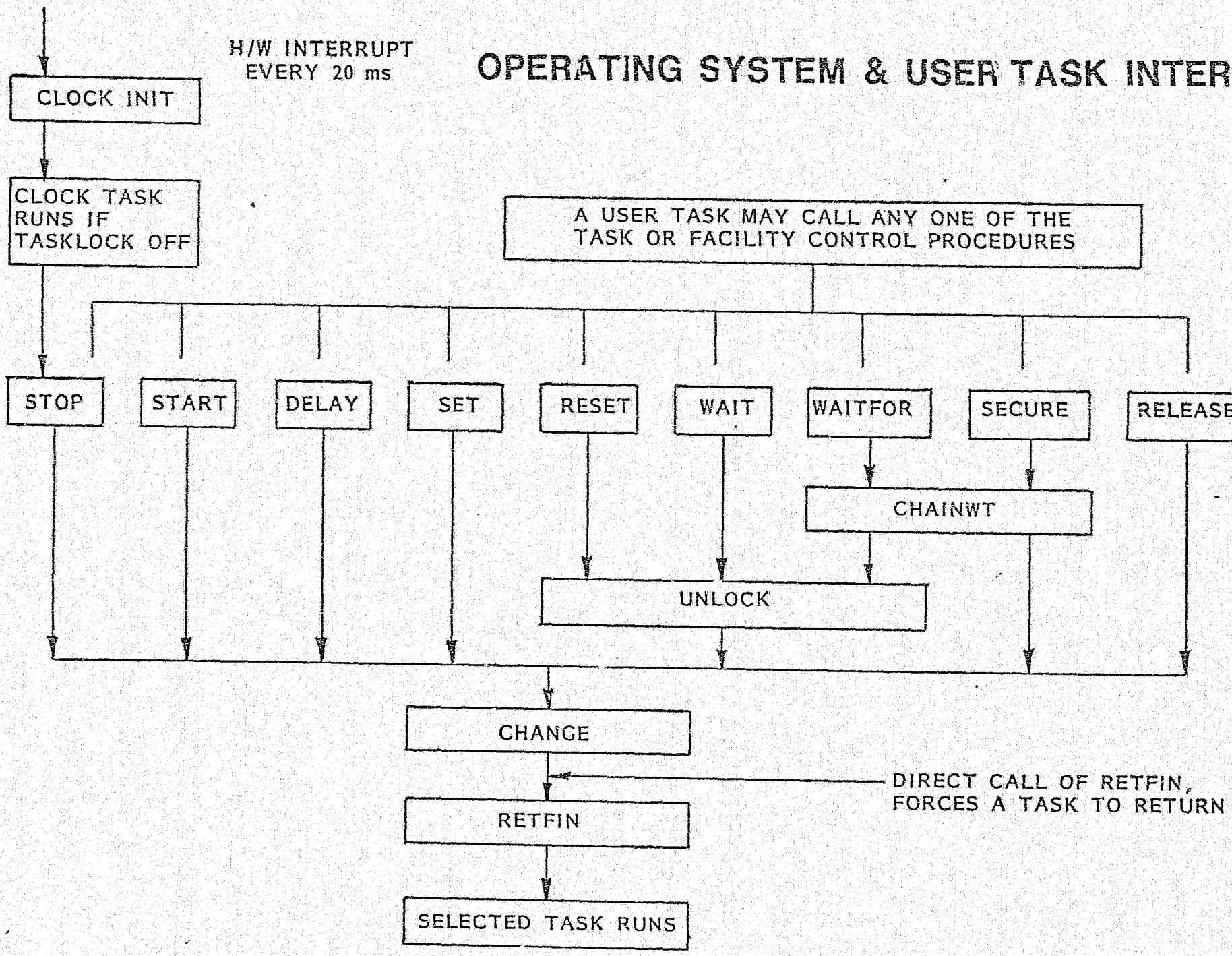
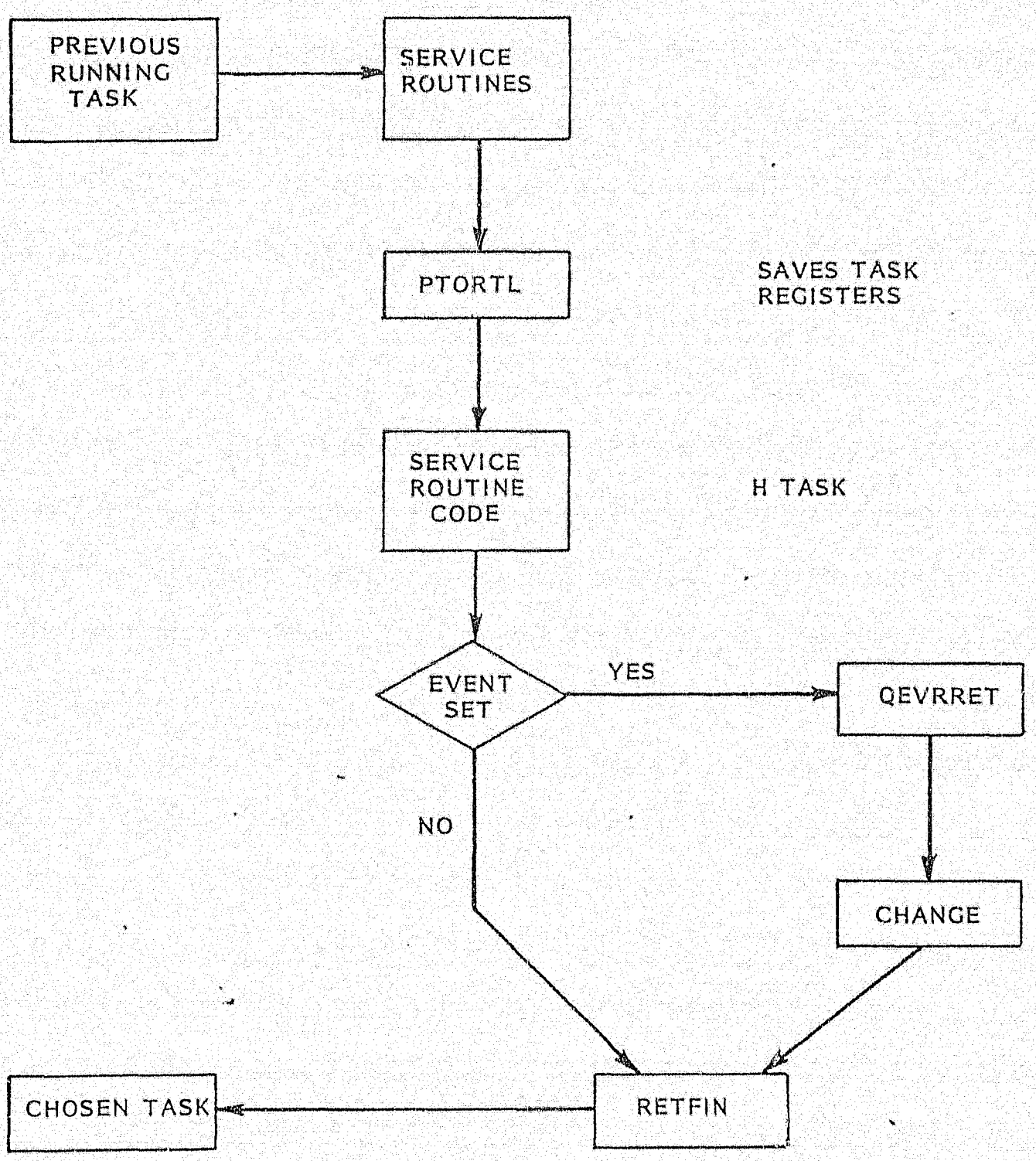


Figure 3.3

Figure 3.4

S-TASK CHANGE DUE TO AN INTERRUPT



CHAPTER 4

MEMORY MANAGEMENT PHILOSOPHY IMPLEMENTED

4.1 INTRODUCTION

To achieve the objective of introducing memory management into SMT two areas had to be studied in detail namely:

1. the DEC PDP-11 memory management unit (mmu) hardware.

2. the SMT operating system method of operation.

With a full understanding of these two areas, memory management could be introduced without increasing the kernel overhead (see Figure 4.1).

4.2 DEC PDP-11 MEMORY MANAGEMENT HARDWARE PHILOSOPHY

The DEC memory management unit (mmu) employs the technique of memory segmentation and relocation. The 32k word virtual addressing area is segmented into 8 pages. Two registers per page exist in the mmu to fully specify each page, namely

1. Page Address Register (PAR) which contains the relocation constant of the page. This constant maps the virtual page to the physical page in memory.

2. Page Descriptor Register (PDR) specifies the attributes of the page e.g. the page size, whether it is read only or read/write etc.

The DEC addressing bus has two options, namely 18 bit or 22 bit addressing. Either option may be used with the memory managed version of SMT. The mmu takes the 16 bit virtual address and adds to it a further 2 (or 6) bits, whose value is dependent on the constant in the corresponding page PAR register, to produce an 18 (or 22) bit physical memory address word (see Figure 4.2).

For the modifications to SMT, it was decided to swop the PAR constants only and not the PDR constants. Thus the PDR's are fixed at startup time, with all the pages having read/write capability. The reason for not swopping the PDR constants, was to minimise the increase in overhead due to the memory management facilities. If the PDR constants were also swopped at task change time, the overhead would have been doubled. This can be carried out without jeopardising the security of the system. When installing the PAR constants of a task in the mmu, part of the adjacent task will also be in scope. Because of the security of RTL/2, the user need not worry about the task executing instructions of the adjacent task. Furthermore, any user of SMT has to have a full understanding of the kernel and its mode of operation and therefore the hardware protection offered by the mmu is of little use.

The mmu has 2 modes of operation namely "kernel mode" and "user mode". In kernel mode all the instructions available are allowed whereas in user mode the software operates in the restricted mode and certain functions such as the HALT and RESET instruction cannot be used. It was decided that the processor should always operate