

FOR REFERENCE

NOT TO BE TAKEN FROM THIS ROOM

**A CONCURRENT PROGRAMMING SYSTEM  
WITH A FLEXIBLE STRUCTURE**

by

TAMER ŞIKOĞLU

B.S. in Ch.E. Middle East Technical University, 1976

M.S. in C.S., Boğaziçi University, 1978

Bogazici University Library



39001100315459

14

Submitted to the Institute for Graduate Studies in  
Science and Engineering in partial fulfillment of  
the requirements for the degree of

Doctor

of

Philosophy

Boğaziçi University

1984

A CONCURRENT PROGRAMMING SYSTEM  
WITH A FLEXIBLE STRUCTURE

APPROVED BY

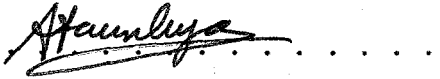
Doç.Dr. Tunç Balman  
(Thesis Supervisor)



Doç.Dr. Hamit Fişek



Doç.Dr. Avedis Hacınlıyan



DATE OF APPROVAL: 18.6.1984

## ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis supervisor, Doç.Dr. Tunç Balman for his continuous interest and guidance throughout the course of this study. Especially, his careful evaluation of this manuscript was of invaluable assistance in the preparation of this dissertation.

I would like to thank Mr. Vasil Kadifeli and Dr. Vahan Kalenderođlu for their suggestions during the course of the study. I also thank Doç.Dr. Avedis Hacınlıyan and Doç.Dr. Hamit Fişek for their careful evaluation of the manuscript.

I am grateful to Mrs. Mine Kalenderođlu for her patience and care in typing this thesis.

Tamer Şıkođlu



## ABSTRACT

This dissertation describes the design and implementation of a concurrent programming system called CPS/1100 which is a modelling language used in systems programming. Processor dedication and program trace capabilities are the distinguishing characteristics of the language. In addition to these features, the implementation scheme used in developing the language is such that it can easily be modified and extended to develop languages for specific applications. This flexible character of the language can be considered as the main contribution of the present work.

Structurally, CPS/1100 is an extension of a Pascal compiler by prescanner and postscanner routines. Prescanner routines provide syntax and semantic checks of the CPS/1100 statements and the control directives, while the postscanner routine modifies the assembler codes produced by the Pascal compiler. Concurrency primitives of Exec8, which is the system program of Univac 1100 machines, are utilized to maintain task creation, removal and control.

This dissertation provides an overview of concurrent programming and multitasking on Univac 1100 systems. Statements, structure and implementation details of CPS/1100 are explained. Typical concurrent programming problems are solved by using CPS/1100 to prove its capabilities. Finally, this study is evaluated.

## Ö Z E T

Bu tez çalışmasında sistem programlamada kullanılacak bir modelleme dili olan bir eşanlı programlama sistemi CPS/1100'ün tasarım ve gerçekleştirilmesi anlatılmıştır. Dilin işleyici tahsisi ve program izleme olanakları ayrıcalık gösteren özellikleridir. Bu özelliklerin yanısıra, CPS/1100 kolay değiştirilebilir ve özel uygulamalar için dil hazırlamak üzere geliştirilebilir bir yapıda gerçekleştirilmiştir. Dilin bu değiştirilebilir özelliği bu çalışmanın ana özelliği sayılabilir.

Yapısal olarak, CPS/1100 bir Pascal derleyicisinin öntarayıcı ve arttarayıcı yordamlar vasıtasıyla geliştirilmiş şeklidir. Öntarayıcı dizgeler CPS/1100 cümlelerinin ve kontrol yönlendiricilerinin ifadesel ve anlamsal kontrolünü yapar. Arttarayıcı ise Pascal derleyicisinin ürettiği bütünleyici kodları değiştirir. Univac 1100 makinalarının sistem programı olan Exec8'in eşanlılık yapıları iş yaratımı, yok edilmesi ve kontrolü için kullanılmıştır.

Bu tezde eşanlı programlama ve Univac 1100 sistemlerinde çokişli çalışma üzerine bir genel bakış verilmektedir. CPS/1100'ün cümleleri, yapısı ve gerçekleştirilme detayları açıklanmaktadır. Tipik eşanlı programlama problemleri CPS/1100 kullanarak çözülmüş ve CPS/1100 ile yapılabilecekler gösterilmiştir. Son kısımda, bu çalışma değerlendirilmiştir.

## TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
" ÖZET	v
LIST OF FIGURES	x
I. INTRODUCTION	1
II. AN OVERVIEW OF CONCURRENT PROGRAMMING AND CPS/1100	5
2.1 Historical Development of Concurrent Programming	5
2.2 Concurrent Programming: Concepts, Problems, and Solutions	7
2.3 Concurrent Software	19
2.4 Application Area of Concurrent Software	26
2.5 Description of CPS/1100	28
III. MULTITASKING ON UNIVAC 1100 SYSTEMS	31
3.1 Multiple Activity Programs and Activity Control	31
3.2 Activity Creation, Removal and Naming	33
3.3 Activation and Deactivation of a Named Activity	34
3.4 Waiting for Activity Completion	35
3.5 Contingency Registration and Interactivity Interrupt	35
3.6 Test-and-Set Instruction and Test-and-Set Clearing	37
3.7 Test-and-Set Queuing Registration and Deregistration, Test-and-Set Queuing and Activation	37
3.8 Activity/Processor Dedication	38

	<u>Page</u>
IV. CPS/1100 KEYWORDS, STATEMENTS, STRUCTURE AND IMPLEMENTATION DETAILS	40
4.1 Declarations	41
4.1.1 Process and Monitor Declarations.	41
4.1.2 Queue Type Declaration	41
4.1.3 Forward Declaration	42
4.1.4 I/O Initiation and I/O Termination Locks	42
4.1.5 I/O Initiation and I/O Termination Flags	42
4.2 Keywords and Statements	43
4.2.1 Startup With Taskname and Startup Slavetask Statements	43
4.2.2 Fork Statement	44
4.2.3 Release Statement	45
4.2.4 Await Statement	45
4.2.5 Wait and Signal Statements	46
4.2.6 Lock and Unlock Statements	46
4.2.7 Cobegin and Coend Statements	47
4.2.8 Clear Statement	48
4.2.9 Quit and Quit by Checking Statements	49
4.2.10 Suspend and Wait Forever Statements	49
4.2.11 Activate and Deactivate Slavetask Statements	49
4.2.12 Terminate Statement	50
4.2.13 Fetch Status of Slavetasks Statement	50
4.2.14 Dedicate Processor and Original Mode Statements	50
4.2.15 Cregion/Cend Statements	51
4.2.16 Snoopy on and Snoopy off Statements	51
4.2.17 Snapdump Statement	52
4.3 Implementation Details	52
4.3.1 Structure and Storage Allocation Mechanism of Pascal 8R1	52
4.3.2 Modifications on Pascal 8R1 to Provide Parallel Processing	57
4.3.3 Prescanner and Postscanner Routines of CPS/1100	65
4.3.3.1 Description of Prescanner Phase0	67
4.3.3.2 Description of Prescanner Phase1	68
4.3.3.3 Description of Prescanner Phase2	68
4.3.3.4 Description of Postscanner	71
4.4 Comparison of CPS/1100 Statements with Other Concurrent Languages	72

	<u>Page</u>
V. CASE STUDIES	74
5.1 Case Study One: Producer-Consumer Problem	74
5.1.1 Problem Statement	74
5.1.2 Method of Solution	74
5.2 Case Study Two: Dining Philosophers	79
5.2.1 Problem Statement	79
5.2.2 Method of Solution	79
5.3 Case Study Three: Quicksort with Processor Dedication	86
5.3.1 Problem Statement	86
5.3.2 Modified Quicksort Algorithm	87
5.3.3 Program Description	87
5.4 Case Study Four: Gaussian Elimination with Processor Dedication	90
5.4.1 Problem Statement	90
5.4.2 Gaussian Elimination Algorithm for Dual Processor Systems	90
5.4.3 Program Description	92
5.5 Case Study Five: A Model Operating System	96
5.5.1 Problem Statement	96
5.5.2 Model	96
5.5.3 Program Description	98
VI. DISCUSSION	110
6.1 The Use of CPS/1100	110
6.2 Extensibility and Portability of CPS/1100	111
6.3 Comparison of CPS/1100 with Other Concurrent Languages	112
6.4 Recommendations for Future Research	114
VII. CONCLUSION	117
APPENDIX A. MACRO PROGRAMMING LANGUAGE	121
APPENDIX B. CPS/1100 CONTROL CARDS	123
APPENDIX C. CPS/1100 STATEMENTS AND KEYWORDS	125



	<u>Page</u>
APPENDIX D. LISTINGS OF MACRO PROGRAMS	128
D.1 Prescanner Phase0	129
D.2 Prescanner Phase1	137
D.3 Prescanner Phase2	140
D.4 Postscanner	161
APPENDIX E. PROGRAM LISTINGS OF CASE STUDIES	167
E.1 Case Study One: Producer-Consumer Problem	168
E.2 Case Study Two: Dining Philosophers	191
E.3 Case Study Three: Quicksort	
E.3.1 Without Processor Dedication	202
E.3.2 With Processor Dedication	213
E.4 Case Study Four: Gaussian Elimination	
E.4.1 Without Processor Dedication	220
E.4.2 With Processor Dedication	225
E.5 Case Study Five: A Model Operating System	229
APPENDIX F. PROGRAM TRACE AND DUMP OUTPUTS	263
F.1 Program Tracing	264
F.2 Snapdump Output	266
REFERENCES	268

## LIST OF FIGURES

	<u>Page</u>
Figure 2.1 Graph Representation of Reusable Resources	17
Figure 2.2 Reduction of a Reusable Resources Graph	17
Figure 2.3 Block Diagram of the CPS/1100 Structure	30
Figure 4.1 Storage Allocation Map of Pascal 8R1	55
Figure 4.2 Activation Record Map of Pascal 8R1	55
Figure 5.1 Schematic Description of Producer-Consumer Problem	74
Figure 5.2 Representation of a Process	75
Figure 5.3 Representation of a Monitor	76
Figure 5.4 Forks on the Table	80
Figure 5.5 Representation of the Model Operating System	97

## I. INTRODUCTION

Concurrent programming has become an important topic for the efficient use of existing hardware facilities of computer systems in recent years. Conceptually, concurrent programming is a technique for expressing parallelism and for solving synchronization and communication problems of systems programming. This technique is used to increase computer efficiency and to manage environments where many things need attention at the same time. It also provides an abstract setting for studying parallelism without considering the machine details.

Concurrent programming was born with the third generation computers. Multiactivity (concurrent) system programs were essential to manage slow speed peripherals connected to fast central processor(s) of these machines. Studies performed by Dijkstra [1,2,3]\* have been the milestones on the subject in its infancy. In the seventies, concepts and data structures proposed by Hoare [4], by Hansen [5] and by Dijkstra [6] were used in the concurrent language development studies. The present challenge of programming methodology is a type of programming called "Distributed Programming" in which concurrent processes (activities) may communicate only by input/output operations.

It is observed that two different approaches exist in concurrent language development studies. The first, which may be called "professional approach", has a tendency either of enriching an already existing sequential language with concurrency features or to develop a language dedicated to specific hardware families. The enrichment of Algol and PL/1 with concurrency constructs and the development of a variety of machine dependent system programming languages are works belonging to this approach. The second approach which is mostly applied to academic studies, is aimed at developing machine independent language constructs

---

\* Numbers enclosed in brackets refer to the references at the end.

and new data structures. For this academic study presented in this dissertation, the methods of the professional approach are preferred.

In fact, existing concurrent languages are the results of extensive works which are usually carried out by software teams. The syntax and capabilities of these languages were strictly defined. Therefore, modifications on such languages can not be easily done. The absence of "processor dedication" and "program trace" capabilities are the major deficiencies of these languages.

At present, every medium or large scale computer system does not have user accessible concurrent programming languages. But, in almost all of these systems, there exist high level sequential languages and primitives (mostly assembler instructions) that provide concurrency on program control flow. Therefore, a practical method to develop a powerful and flexible concurrent language is to enrich a high level sequential language with concurrency primitives. This is the design philosophy of the concurrent language developed in the present study.

In Univac 1100 series computers, the operating system, Exec 8 [7,8], controls and manages operations in several modes including real-time. Users may write multiactivity (concurrent) assembler programs by using low level concurrency primitives of Exec 8. On the other hand, concurrent program modules can be written by using PL/1, Cobol and PLUS [10,11] the latter being a special system programming language. But, Univac Cobol [12] and PL/1 [13] compilers are not suitable for sophisticated concurrent programming applications. Although both PLUS and 1100 Assembler provide concurrent programming structures, they are not convenient for average programmers. The concurrent language developed in this study was designed to satisfy the needs of these average programmers.

This dissertation describes the design and implementation details of a concurrent programming system, CPS/1100, which is a "Pascal like" modelling language suitable for system programming. The syntax of CPS/1100 statements is similar to that of Pascal statements, because CPS/1100 is an extension of a Pascal compiler. CPS/1100 was developed as a multiphase translator. Its prescanner routines recognize "CPS/1100 control directives" and analyse the "input program stream". These

routines were written in Macro [14] which is a string manipulation language. The program stream processed by the prescanner routines is compiled by using Pascal 8R1 compiler [15] and then an assembler program text is obtained. This assembler program is further processed by the CPS/1100 postscanner routine to generate a multitask (concurrent) assembler program. After these steps, the program is assembled, linked and executed.

The "processor dedication" and the "program trace" capabilities are the distinguishing characteristics of CPS/1100. In addition to these features, the implementation scheme which permits easy modification and extension of the language for specific applications has been preferred. This flexible character of the language together with the "processor dedication" and "trace" capabilities can be considered as the main contribution of the present work to the field of concurrent programming.

Following this introduction, the second chapter gives an overview about concurrent programming; its historical development including the concepts and the problems related with concurrency and its relationship with the operating systems are explained. Synchronization techniques, deadlock and deadlock detection methods are stated. Explanations about objective and structural description of CPS/1100 conclude the second chapter.

Multitasking (concurrent programming) on Univac 1100 machines and the Exec 8 concurrency constructs for the "activity creation, control and removal" and for the "activity synchronization" are explained in Chapter three.

Chapter four is devoted to CPS/1100 statements, keywords and control directives. Implementation details of CPS/1100 are given in this chapter.

Chapter five consists of case studies about concurrent programming. Case one is a "monitor" implementation where the "producer/consumer" problem [16] is solved. Case two is the well-known "dining philosophers" problem [3] which is a resource allocation problem. Case three is the implementation of a parallel sorting algorithm. The "quicksort" algorithm [17] is modified for a dual processor system in this case study. Case four is the implementation of a parallel algorithm applied to a matrix operation [18]. Case three and case four are the demonstrative examples of the CPS/1100 "processor dedication" feature. Case five is a model for

a "timesharing" and "multiprogramming" operating system in which the "minimum response time" scheduling strategy is applied.

Chapter six is devoted to the discussion of the study. Comparison of CPS/1100 with some of the existing concurrent languages and recommendations for future research are given in this chapter.

Chapter seven contains the conclusion of the study.

Six appendices are devoted to presenting details.

## II. AN OVERVIEW OF CONCURRENT PROGRAMMING AND CPS/1100

An overview of concurrent programming is given in this chapter. Previous studies on concurrent programming are summarized in chronological order. Concepts and basic problems due to concurrency are introduced. Methods and data structures that have been developed for solving concurrency problems are briefly explained. Characteristics of existing concurrent languages and application areas of concurrent software are given. The design aims and structure of CPS/1100 are described in detail.

In this chapter and the succeeding ones, the terms process, activity and task are used to denote a computation in which the operations are carried out strictly one at a time. The term concurrent processes is used to specify that the processes overlap in time. The term resource is used to describe a set of objects (such as processor, memory, programs) that a process can use. The term synchronization is used to denote any constraint on the order in which operations are carried out. The term quasiparallel means that program control, in a single processor system, passes to another process only at places where the current process explicitly specifies its own suspension. The other terms related with concurrent programming are described within this chapter.

### 2.1 Historical Development of Concurrent Programming

In the mid 1950's computer architecture changed with the invention of large magnetic stores and asynchronous operating peripheral devices. Slow speeds of peripheral devices made asynchronous operations essential in computer systems. Operating systems became complex beyond human

comprehension and due to their size they became unreliable. Then a search for abstraction began.

In the late 1960's, high level languages were used in the programming of operating systems as a result of these abstraction studies [19,20]. Through the use of high level programming languages, system programmers were able to concentrate their productivity on the problem being solved rather than on machine details. They were able to produce readable texts, therefore testing and maintenance of programs became easy. The extraordinariness of these systems programming languages is the coordination and controlling capabilities of concurrent activities and the convenience of representing tables, queues, lists and similar data structures.

The developments on design and programming of operating systems were driven by the conceptual innovations on concurrent programming. In early 1960's the operating systems, such as Atlas (1961) and Exec II (1962) were not reliable because in those years the deadlock problem was not understood clearly enough [21,22]. Dividing a concurrent program into asynchronous modules with time independent behaviour was an important innovation and was first implemented in MIT's CTSS project [23].

In 1965, Dijkstra introduced the critical region concept [2]. Processes of concurrent programs usually operate on common variables for synchronization. Such common variables must be available to a single process at a time. Critical regions are used to prevent competing processes from using common variables simultaneously. Dijkstra proposed a special data type, named semaphore, for solving communication problems among processes. He used a hierarchical program structure [3] on THE operating system [1].

In 1971, Hoare introduced the concept of conditional critical region [4] by which execution of processes may be delayed until a shared variable satisfies some conditions. Process queues associated with shared variables were proposed by Hansen [24] to implement conditional critical regions. Hansen proposed a language notation monitor [5]. Hoare developed the monitor concept as a method on structured system programming [25].

In 1974, Concurrent Pascal [16,26,27,28,29] was defined and implemented. It was the first concurrent programming language based on



the monitor concept. Another language suitable for the design and testing of operating system algorithms, Simone, was implemented by Kaubisch [30]. Pascal has been used as basis language in the Simone project. Modula which is also based on the process and monitor concepts was proposed by Wirth in 1977 [31,32,33]. Pascal-plus, which is a realtime and simulation purpose language was implemented by Welsh and Busthard [34]. In those years Dijkstra proposed another control structure, called guarded commands, for process synchronization [6]. The coroutine approach has been used to simulate parallelism in some studies [35,36,37]. CSP/k was developed by extending Holt's SP/k system [38].

In the late seventies, efforts were concentrated to develop "program verification rules" [39,40] and new languages for distributed systems. Hoare proposed a language named Communicating Sequential Processes [41] and Hansen proposed another language named Distributed Processes [42] to satisfy the constraints of distributed storing, in 1978. On both languages Dijkstra's guarded commands [6] are adopted as sequential control structures.

The Ada programming language [43,44], which will probably be a widely used language in concurrent programming, was announced by U.S. Department of Defence in 1980. Edison [45,46], SR (Synchronizing Resources) [47], \*Mod [48], Platon [49], Implementation of Communicating Sequential Processes [50] are some of the recently developed concurrent languages. Efforts to design an ideal abstraction for parallel activities and concurrent programming still continue.

## 2.2 Concurrent Programming: Concepts, Problems and Solutions

Concurrent programming is a programming technique in which several parts of a program may be in execution at a given time. For many years, it has been used for writing parts of operating systems. But, it is only recently that appropriate high level concurrent programming constructs have been developed and used in writing operating systems.

Operating systems of third generation computers are collections of asynchronous, interruptible software modules which are called processes. The functionally fundamental module of an operating system, called the

kernel, schedules the CPU among processes, receives i/o interrupts, transmits them to the devices and supports interprocess communication. The kernel uses clock interrupts to distribute the CPU time among processes. It gives each process the appearance of having its own CPU. Each of these virtual CPUs behaves like a real CPU except that there is a variable rate of progress. Use of multiple CPUs is called multiprocessing. The kernel and the physical (single processor) machine simulate a multiprocessor non-existing machine which is called virtual machine. The kernel's responsibilities can be implemented by hardware or microprogramming. Such an implementation makes asynchronous interrupts indivisible to all software modules.

Control problems of concurrency arise in systems where tasks operate in a parallel manner. Typical control problems are nondeterminacy in the system, deadlocks in the system, mutual exclusion and synchronization of tasks. Determinacy of a system is the independence of the system's outcome to the relative speeds of the processes within the system. Time dependent errors exist in a nondeterminate system. Deadlocks may frequently appear in limited resource systems. The mutual exclusion problem arises when two or more processes request the same resource in a system. When a resource is shared by processes in an uncontrolled manner, the processes modify the internal state of the resource and this causes nondeterminacy or deadlock in the whole system. Synchronization problems appear whenever the processes are dependent upon each other. Such processes must have a means to communicate with each other.

Dijkstra's THE operating system [1] may be used to demonstrate the basic concepts of concurrent programming. THE system has a hierarchical organization with five levels. Level one, which is the lowest level, is the kernel. It implements a virtual CPU for each process, i.e. the timesharing facility. Level two implements virtual memories to the processes on higher levels. Level three enables processes on higher levels to communicate with the operator. Level four consists of one process per i/o device. Level five consists of one process per allowed user program.

The outcome of a program, which consists of concurrently executing processes, depends on the speed of processes, i.e. some parts of the

program are time critical. If the outcome does not depend on the speed of the process, then the processes are called "independent processes".

When processes of a program have a shared data (or resource), they must access and update the data one by one. Otherwise, the final state of the shared data is unpredictable. As an illustration:

"Let, P1 and P2, which are the processes of a program, have a shared data, named CLOCK. P1 updates CLOCK while P2 clears it. P1 and P2 do not communicate with each other.

Process P1:

```
-
-
clock:= clock + increment;
-
-
```

Process P2:

```
-
-
clock:= 0;
-
-
```

The outcome of the program absolutely depends on the speed of the processes. If the execution sequence of processes is P1, P2 then the result stored in the variable CLOCK is zero, otherwise it is different from zero."

The result is unpredictable and may not be the desired one, in this example. Such disastrous cases, called race conditions, can occur whether the processes execute physically or logically in parallel. In the physical parallelism, each process has its own processor and is simultaneously running. But, in the case of logical parallelism, processes are timesharing on a single processor and one process can be running at a time.

Race conditions may occur at the hardware level. As an example:

"Two processors, a CPU and a channel (or an i/o processor) can access a memory location. CPU tries to update the memory location, while the channel tries to store information to that location."

In practice, memory interlocking circuits allow only one process at a time to access the memory. Thus, the confusion described above is prevented.

Time critical parts of the concurrent programs are called critical sections or critical regions. Mutual exclusive access to these sections must be guaranteed either by software or by hardware tools.

Another fundamental concept, conditional critical region, [4] is used to explain permissibility of an operation on a critical region. The original suggested form for the conditional critical region is:

with r when B do C

where

r: common resource or data

B: boolean expression which tests the permissibility of an operation on critical region

C: critical region.

Concurrent programming software provides services for creating multiple activities and communication between activities. It is significantly difficult to understand and debug multiactivity programs. They should be approached with caution, because several activities could simultaneously access the same data and programs may work one day but may fail the next. Therefore, the problems of concurrency originate from the difference between timing of the asynchronous activities. Several synchronization techniques are used to solve the problems arising from concurrency in systems. Any synchronization technique must ensure that determinacy of the system is preserved and each activity is completed in a finite time interval.

Semaphores are the oldest process synchronization tools [2]. A semaphore is a synchronization variable and only two operations on it are allowed. These operations allow a process to block itself to wait for a certain event and then to be awakened by another process when the event occurs. These operations are called P and V operations. P operation, also called wait operation, and V operation, also called signal operation, can be characterized as:

P(s): wait until  $s > 0$  and then subtract 1 from s.

V(s): add 1 to s.

where s is a semaphore variable.

Both P and V operations are indivisible. If the processes share CPU by time slicing, the P and V operations can be implemented via a software kernel. If the value of a semaphore is only 0 or 1 then such a semaphore is called a binary semaphore.

Locking scheme [51] is a synchronization method applicable to concurrent programming. A process may lock a shared data structure to ensure its inaccessibility while it is in a temporarily inconsistent state. If a process attempts to lock an object that has already been locked it must either wait, abort itself or preempt the other process. Processes unlock the data structure after they complete their work on it.

Lock and unlock primitives [38] are used to guarantee mutually exclusive access to the shared data or resources. Lock/unlock mechanism, functionally, is a gate for a critical region. Before entering a critical region, a program should lock its corresponding gate, and afterwards should unlock it.

```
lock (< lock identifier >);
```

```
critical region
```

```
unlock (<lock identifier>);
```

A typical implementation of lock/unlock mechanism can be seen in PL/I [13]. Lock/unlock statements and event variables of PL/I can be used to create critical regions. In PL/I, an event variable can be declared as:

```
declare (<event variable>) event;
```

A built-in PL/I function, completion, can be used to change the state of an event variable, such as:

```
completion (<event variable>) = true;
```

or

```
completion (<event variable>) = false;
```

The wait statement of PL/I can be used to suspend program control of a task until status of corresponding event variable become true.

```
wait (<event variable>);
```

The task which is in the wait state can not change the status of an event variable, therefore resetting of an event variable to false should be performed explicitly. A task in PL/I can be created by a call statement

```
call (<procedure name>(<parameter>) task (<taskname>)
      event (<event variable>);
```

Call statement with task and event options starts up a child process to the caller. At that moment, the event variable in the call statement is set to false. When the child process has completed its job, it can be terminated by executing an exit statement. This sets the event variable used in the call statement to true. Parent process can wait for the completion of the child process by executing a wait statement which uses the event variable that is used in the call statement.

A PL/I built-in function, status, can be used for interprocess communications. For example:

In process P1:

```
status (<event variable>) = integer variable ;
```

In process P2:

```
if status (<event variable>) > <integer variable> then ...
```

Monitor [25] is another synchronization tool for concurrent programming. A monitor can be thought of as a combination of a set of shared data and the critical sections for that particular set. A monitor provides appropriate facilities for guaranteeing mutual exclusion

and for blocking and waking up processes. A monitor may have more than one entry point. If a process enters into a monitor and finds that a required condition, related with shared data, is not true, then it enters into a wait state by executing a wait statement. When another process enters the monitor and finds the condition to be true, it executes a signal statement that removes a waiting process, if any, from the queue and wakes it up. The following Concurrent Pascal codes were used to implement a monitor which represents a clock keeping track of real-time. Explanations for some statements are given in quotation marks.

```

const  one min = 60.0; onehour = 3600;      "constant declaration"
        halfday = 43200; oneday = 8640000;
type   clock = monitor                    "monitor declaration"
var    seconds: real;
function entry value: real;                "defines the present"
        begin  value: = seconds end;        "value of time"
procedure entry correct (time:real);      "sets the time to"
        begin  seconds: = time end;        "a given value"
procedure entry tick;                      "increments the time"
        begin
            seconds: = seconds + 1.0;        "by one second"
            if seconds > = oneday
                then seconds:= seconds - oneday;
        end;
begin  seconds: = 0.0 end;                "sets time zero"

```

Processes enter a monitor to test or update a set of critical data, thus, they communicate with each other. Monitors are very useful and understandable structures for interprocess communication. Another way of interprocess communication is message passing mechanisms. Send and receive operations are used to specify desired i/o operations.

The send primitive is used to pass a message to another process. The receive primitive is used to block the user process until a message is accepted from the sender process. In most of the message passing

systems, the transmitted message is stored until received in some special buffer, sometimes called a mailbox. Many systems allow multiple-slot mailboxes in which messages are arranged on a first-in-first-out basis. In a zero-slot message transmission mechanism, if a send operation occurs first, the sender is blocked until a receive operation occurs. Conversely, if the receive occurs first, the receiver is blocked until the send occurs.

Coroutines [37] are process synchronization tools by which a quasi-parallel programming environment can be built. A coroutine starts with its first statement when it is activated the first time. It continues the execution until it suspends itself by transferring the control to another coroutine. The suspended coroutine can only be reactivated when an explicit transfer back to it is performed by one of the other coroutines. Then, it resumes its execution at the statement following the point of suspension.

Guarded commands and guarded regions [6] are the statements which are used to control program execution in recent parallel programming studies [41,42]. Guarded commands have the following syntax:

```
if B1:S1 | B2:S2 |.. end
do B1:S1 | B2:S2 |.. end
```

where  $B_i$  is a boolean,  $S_i$  is a sequence of statements and | denotes operator 'or'.

Meaning of the if statement is:

If some of the conditions  $B_1, B_2, \dots$  are true, then select one of the true condition  $B_i$ , and execute the statement  $S_i$  that follows it; otherwise stop the program.

Meaning of the do statement is:

While some of the conditions are true, select one of them arbitrarily and execute the corresponding statement.

The guarded regions have the following syntax:



```

when B1:S1 | B2:S2 |... end
cycle B1:S1 | B2:S2 |... end

```

Meaning of the when statement is:

Wait until one of the conditions is true and execute the corresponding statement.

Meaning of the cycle statement is:

Endless repetition of a when statement.

If several conditions are true within a guarded command or region, the statement that the machine will select is unpredictable.

Uncontrolled usage of lock/unlock or wait/signal mechanisms may cause deadlock in the system. Deadlock is a state in which two or more processes are waiting indefinitely for conditions which will never hold. It involves circular waiting such that each process is waiting for a condition which can only be satisfied by one of the others. But, since each process expects one of the others to resolve the conflict, they are all unable to continue. The deadlock problem was first recognized and analysed by Dijkstra [2]. A typical example of deadlock is explained below.

"Process P1 and P2 lock resources R1 and R2 respectively. After a while, process P1 tries to access the resource R2, while process P2 tries to access the resource R1. Both resources are not available to the requests of processes P1 and P2."

To analyse a deadlock problem in a system, the system is considered as a combination of a set of states and a set of processes, where each process is a function that maps states into states. A process is blocked in a state if it cannot continue execution when it is in that state. A process is deadlocked in a state if it is blocked in that state and in all future states which the system can reach. A state is safe if no process can map that state into a deadlock state. In a system with many different kinds of resources, deadlock is a difficult problem. If

a resource in a system can be used repeatedly by many processes, it can be called a permanent resource. If the resource is produced by one process and consumed by another, this resource can be called a temporary resource. A disk device and a message are examples for permanent and temporary resources respectively.

Necessary conditions for the occurrence of a deadlock with respect to permanent resources are the absence of a mutual exclusive access mechanism to the resource, or having a nonpreemptive scheduling strategy on the resource, or allowing piecemeal allocation of the resource, or circular waiting of processes to acquire each other's resources. Preemptive scheduling strategy is a useful way to prevent deadlock but it leads to a less efficient utilization of resources. Another way of deadlock prevention is to allocate all resources needed by a process in advance of its execution. For example, all tape unit requests of a process can be satisfied just before the execution of the process starts, however this leads to the inefficient utilization of resources. A way of deadlock prevention, due to circular waiting of processes, is sequentially, hierarchically, ordering requests. The requests and releases of hierarchal allocated permanent resource are subject to a fixed sequential ordering [52]. Also, temporary resources can be allocated in a hierarchal manner to prevent deadlocks. Message transmission between processes can be organized hierarchally with finite process levels. Processes at lower levels, called masters, can supply processes at higher levels, called servants, with messages. Servants can return answers to their masters in response to messages. Thus, messages are sent in one direction only and answers are sent in the opposite direction.

Special models of parallel executing systems, such as Petri nets and computational schemata [52] have been developed to analyse and formalize the deadlock problem. Reusable resource graph [38] is an example of the modelling approach for the deadlock problem. This method concentrates on specific problems related to resource sharing. Resources such as tape drives are called reusable, because after they have been used by one process, they can be re-used by another process. The physical devices of computer systems, such as memory, tape drives and disks, can be thought

of as reusable resources. A system can be represented as a graph having nodes for each process and resource. The units of a resource are shown by small triangles inside the resource nodes as illustrated below:

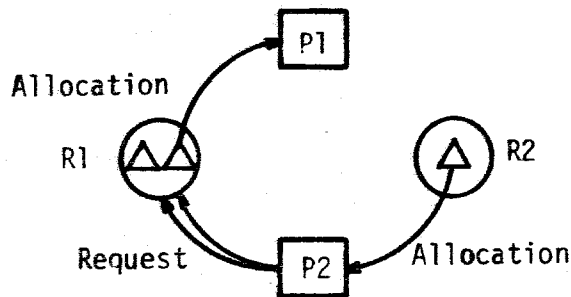
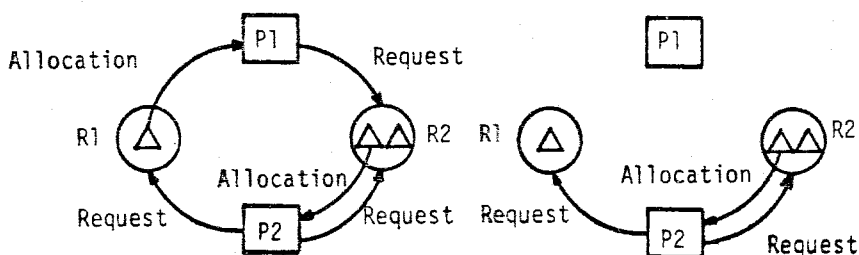


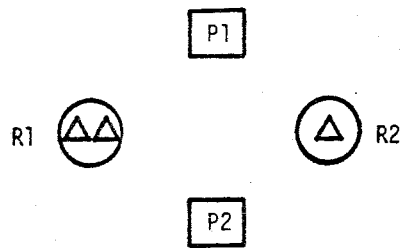
Figure 2.1 - Graph representation of reusable resources.

In this graph, resource R1 has two units and R2 has one unit. One of the units of R1 has been allocated to P1. P2 has acquired the unit of R2 and has requested both units of R1. P2 will be blocked until P1 releases its unit of R1. The reusable resource graph method is used to analyse such a graph to determine if there is a deadlock, i.e. to see if some processes can never be granted their requests. A graph can be reduced if the request of its processes can be granted. An example of a reduction operation is given below:



a) Initial state: One of the units of R2 has been allocated to P2, P2 has been requested one unit of R1 and one unit of R2. One unit of R1 has been allocated to P1. P1 has requested one unit of R2.

b) Process P1 has been satisfied by using one unit of R1 and one unit of R2. Therefore, the arrow linking R1 to P1 and P1 to R2 are removed to simplify the graph.



- c) Process P2 has been satisfied by using one unit of R1 and two units of R2. The arrows linking P1 to R1, R2 to P2 and P2 to R2 are removed. Therefore, the graph is completely reduced.

Fig. 2.2 - Reduction of a reusable resources graph.

There are no deadlocked processes if and only if the graph is completely reducible. The order of reductions is immaterial because the same final graph is obtained regardless of the order.

Deadlocks can be broken by using time-outs to reawake waits after a specified time interval. However, it is significantly difficult to pick a workable time-out interval. Another approach for deadlock protection is to use a locking strategy where locking rights of resources are determined in hierarchical manner. An alternative locking method, called predicate locks, is locking resources internally against external access when certain values related with resources satisfy predicates [51]. More on the theory of deadlock and related bibliography can be found in Holt's survey [53].

Communications between processes can be performed either by copying parameters within a common store or by i/o between separate stores. Two major classes, procedure calls and message passing, have been evolved for process interaction. Communication between processes may be synchronous or asynchronous. In the case of asynchronous communication, one of the processes can deliver a value and go on; at some later time, another process may pick up this value. Interprocess communication scheme of Concurrent Pascal is an example of asynchronous strategy. Concurrent Pascal uses monitors for process communication. In the case of synchronous, processes must send and receive values simultaneously. The synchronous communication technique was proposed as the communication

method in Communicating Sequential Processes and Distributed Processes. Classification and comparison of communication techniques are discussed by Straunstup [49].

### 2.3 Concurrent Software

The cheap processing power provided by present computer technology has led to a variety of proposals for concurrent programming methodology. In the last decade, many concurrent languages have been proposed, such as Concurrent Pascal, Pascal-plus, CSP/k, Modula, Simone, Communicating Sequential Processes, Distributed Processes, Edison, \*Mod, Ada, SR, Platon and most of these languages have been implemented for different machines. Some of them, such as Concurrent Pascal, Simone, Pascal Plus, Modula, Communicating Sequential Processes, Distributed Processes, Ada are worth discussing because of their structural and functional properties.

Concurrent Pascal [16] is an extension of sequential Pascal with concurrent processes, monitors and classes. It is primarily implemented for a PDP 11/45 machine. Concurrent Pascal compiler generates pseudo codes for a virtual machine that can be simulated in any other physical machine. An interpreter program may analyse and executes those virtual codes. Kernel, in the original Concurrent Pascal implementation, is an assembler program that multiplexes the processor among concurrent processes and gives them exclusive access to monitors. Concurrent Pascal consists of three kinds of system types: process, monitor and class. A process is a combination of a private data structure, a sequential program and access rights. Processes can only operate on their own private data. But, they can share certain data structures in monitors. The other system type monitor consists of shared data structures and procedures called monitor procedures. Synchronization operation are controlled by monitor procedures. A monitor also has an initial operation section for the shared data structure. Concurrent Pascal has a special data type, called queue, which can be used by monitor procedures to control execution of processes. A monitor can either delay the execution of a process and place it in a queue or resume the execution of

a process waiting in a queue. Only one process at a time can wait in a queue. A class defines a data structure and possible operations on it. The machine does not have to schedule simultaneous calls of class procedures in run time. A class can only be initialized once, and then, its parameters and private variables exist forever. Concurrent Pascal checks that the private data of a process is accessible only by that process. It also checks that the data structure of a class or monitor is accessed only by its procedures. There is one restriction due to storage allocation and deallocation mechanism, that a Concurrent Pascal program consists of a fixed number of processes, monitors and classes. These components and their data structures will exist forever after program initialization.

Simone [30] is also an extension of Pascal with three basic features: process, monitor and condition variables. Condition variables are used to synchronize processes by means of wait and signal operations. They can be declared within a monitor only. There is a queue for each condition variable in which two or more processes can wait for their activation. Processes are those parts of Simone programs which can be executed in quasiparallel fashion. In a quasiparallel system, there is by definition only one processor, the program control passes to another process only at places where the current process explicitly specifies its own suspension. As they are activated, processes are pushed onto a sequencing stack and commence execution. Eventually when the sequencing stack becomes empty, the first process on the time queue is moved onto the sequencing stack, simulated time is advanced to its restart time. When a currently executing process performs a signal operation, the reactivated process is moved from the condition variable queue to the sequencing stack and immediately resumes execution in place of the signalling process. The fundamental assumption on Simone store management is that all the processes will be active simultaneously. Therefore, the storage allocation mechanism is static and there exists no storage recovery operation.

Pascal-plus [34] is an extension of Pascal which can be used in either realtime or simulation mode. It has a block structure construct, called envelope, which provides modularity of programs. Essential

programming structures are processes, monitors and condition variables. Condition variables have single first-in-first-out queues. Pascal-plus, being a successor of Simone, provides simulation facilities. It consists of a simulation monitor on which processes may suspend themselves for a period of pseudo time. The programs ultimately intended for real-time use can be tested by simulation facilities of the language. In realtime mode, the actual realtime devices are connected to a system instead of simulation processes that correspond to those devices. The input/output handling of the language is dependent on implementations for specific machines. For example, input/output handling operations are expressed in terms of standard Pascal file operations in the original implementation of Pascal-plus. Pascal-plus allows a static memory allocation system for processes and monitors which are exploited during compilation of the program.

Modula [31] which is based on module concept is intended primarily for programming dedicated computer systems, including process control systems on smaller machines. It relies very strongly on Pascal. The module concept has been introduced as an essential supplement to the block concept. A module is a collection of constant, type, variable and procedure declarations called objects of the module. A module contains two lists of identifiers. A define list mentions all module objects that are to be accessible outside the module. A use list mentions all objects declared outside the module that are to be accessible inside. Identifiers in the define list are said to be exported, those in the use list are imported. Multiprogramming facilities are the processes, interface modules and signals which are added to the language called Sequential Modula. A process can not create other processes. Process creation is possible in the main program only. Modula supports the possibility of dynamic process generation, but it does not support the recycling of its store. Therefore, overall storage allocation mechanism of Modula is static. Synchronization operations are achieved by the use of signals. Signals can be sent, and a process can wait for a signal. Processes cooperate via common variables. An interface module, which is a kind of monitor, is a set of critical sections

where simultaneous execution by several process is excluded. The interface module allows more than one process to be in a critical region, provided that all but one are either waiting for a signal or sending a signal. The essential characteristic of Modula is the absence of interrupts. Modula is unusual in that it is designed to run on a bare machine without any operating system support.

Distributed Processes [42] is a language concept that is introduced for concurrent processes without common variables. These processes communicate and synchronize by means of procedure calls and guarded regions. A process on the language defines its own variables, common procedures and initial statements. It may call common procedures either within itself or within other processes. A process performs two kinds of operations, the initial statements and external requests made by other processes. The language is proposed for real-time applications controlled by microcomputer networks with distributed storage. There is no need for interrupts in a realtime program written in the proposed language. Fast response to external requests is achieved by dedicating a processor to each critical event in the environment. An attractive feature of the proposal is that it lets a process carry out computations between external calls of its procedures. Synchronization of individual processes is handled by means of guarded regions. As an illustration:

```

process    sem; s : int
proc      wait; when s > 0 :   s := s-1   end
proc      signal;   s := s+1
s := 0

```

The process `sem` is a semaphore implementation. It is initialized by the statement `s := 0` and wait/signal operations are defined by procedures. The when/end structure is a guarded region where the calling process waits until variable `s` is greater than zero. Processes are permanent in Distributed Processes language. Proposed mechanism for storage allocation is static. In the implementation of the language, parameter passing between two processes requires a single input operation



before the called procedure is executed and a single output operation when it terminates.

Communicating Sequential Processes [41] is a language proposal which has a brevity so that the primitive concepts are expressed with single character notation. In the language, guarded commands are adopted as sequential control structures. A simple i/o command is used for communication between processes. There is no automatic buffering in the i/o operations. Input commands may appear in guards. The language is implementable both in a machine which has a single main store and in a network of processors connected by i/o channels. A parallel command of the language specifies concurrent execution of its constituent processes. Communication occurs between two processes of a parallel command whenever an input command in one process specifies as its structure, the process name of other process, and an output command in the other process specifies as its destination, the process name of the first process, and the target variable of the input command matches the value denoted by the expression of the output command. The following statement is an illustration for process communication:

```
X :: *[c:character; west?c → east!c]
```

Process X copies characters from process west to process east.

The Ada programming language [43] was developed as the result of a procurement exercise by the U.S. Department of Defence. It is a powerful and elegant language with the features of data abstraction, exception handling, separate compilation and concurrency. The data abstraction construct of the language is called a package which consists of two parts, the package specification and the package body. The package specification lists all objects which are accessible outside of the package. Generic definition facility is an ability of Ada language to parameterize units over appropriate data types. Exception handling feature provides that the control is transferred to an emergency action when a specific exception condition is raised. Separate compilation feature permits the compilation of a program in distinct pieces. Ada allows concurrent tasks to be introduced into a program by using a

notation similar to that for packages. Intertask communication is handled by means of "randevous". Calls to transfer data are received by an accept statement with the called task. Nondeterminism is handled by using a select statement, each select alternative being preceded by an optional guard. Since dynamic memory allocation and deallocation mechanism may be quite slow, Ada allows the maximum space to be allocated to a collection of dynamic objects such as tasks.

In Univac 1100 systems, the vendor supported languages with concurrency features are Plus [10,11], PL/I [13] and Cobol [12]. Cobol is not worth discussing because it is inconvenient for system programming. But, in PL/I multitask programming, users may use the following statements:

- call with task and/or event options
- clear
- lock
- post
- stop with or without task option
- wait
- unlock

Synchronization of the processes can be achieved by using the following built-in functions:

- completion
- status

Task, event and lock constants or variables are used on synchronization operations. A task constant represents a symbolic name which is dynamically associated with a task. It is initially inactive and becomes active when it is associated with a task and is inactive again when the task is terminated. An event constant represents a symbolic name of a dynamically associated task. It is inactive initially, becomes active when it is associated with a task, and is inactive again when the associated task is terminated. A lock constant represents a symbolic name of lock. Locks are used to protect a certain data area. Lock/unlock

statements are used to create a critical region.

A new task may be created by executing a call statement with task and event options. For example:

```
call sub-task task (t1) event (e1);
```

The wait statement is used to suspend the control of a task until an event associated with the wait statement has been completed.

```
wait (e1);
```

Programmers can alter the completion state of an event by using post and clear statements:

```
post      statement sets an inactive event constant or variable  
           to complete
```

```
clear    statement sets an inactive event constant or variable  
           to incomplete
```

The completion built-in function is used to know whether the state of an event is complete or incomplete. The built-in function status is used to change the status value of event variables, therefore these variables are used as communication signals among tasks. But, Univac PL/1 compiler contains a variety of bugs which may be solved in new releases.

The Plus compiler of Univac [11] was a restricted software for many years. It was used on system programming and compiler writing studies under the control of Sperry Univac. In recent years, the Plus 5R1 compiler was released worldwide.

Plus is a language which consists of language constructs similar to PL/1 and Pascal. The powerful feature of Plus is the ability of accessing facilities which can only be used in assembler programming. Plus provides a facility, called inline, for generating assembler code sequences. Since the inline facility of Plus is actually part of the compiler, it has access to the compiler's information tables to retrieve needed information. The inline facility has on and off directives to facilitate the conditional and/or iterative generation of code.

On the other hand, Plus allows programmers to direct the compilation by using compile-time statements, such as copy, include, resume and define. The copy statement provides a mechanism by which line images from other resources are incorporated into the compilation. The include and resume statements permit the conditional compilation of source code. The define statement allows the parameterization of programs.

Modularity in Plus programming is maintained by using module and term statements. The environment facility of Plus allows a program to save processed declarations that can be read into other compilations later.

Multitasking in Plus programming is possible by using inline facility. For that reason, programmers, who try to write multitask programs by using Plus, must have an extensive knowledge of Exec 8 and Univac 1100 assembler programming.

#### 2.4 Application Area of Concurrent Software

Operating systems are the basic application fields of concurrent programming since concurrent programming grew out of problems associated with them. Timesharing, multiprogramming, distributed processing and similar sophisticated software systems can be realized after understanding concurrent programming concepts. Defects of operating systems are reduced to a minimum by using concurrent programming software and abstract data types. Concurrent programming will develop while it is used to increase availability and performance of parallel hardware facilities.

Concepts used on concurrent programming are very similar to concepts used on discrete event simulation. Data structures used in concurrent software may be used for scheduling and queuing events in simulation models. Concurrent software may be extended by adding special type priority lists and data structures to create a powerful simulation language. Simone is an example of this approach. Similarities between queuing network elements and concurrent language elements is stated by Sauer [54]. Various types of nodes on extended queuing models are

similar to semaphores and monitors in concurrent programming.

Concurrent programming is the only programming technique used on programming realtime systems which are the systems such as industrial plant control, military command control, air traffic control systems. Requirements of realtime systems are very special. A realtime program must respond to a variety of indeterministic requests from its environment. The programs cannot predict the order in which requests will occur. It absorbs and processes dozens of different asynchronous external signals and operator commands. Programs must respond to requests within certain time limits. Otherwise, input data to the program may be lost or output data of the program lose its significance. Concurrent software may reduce the programming effort and increase the reliability of programs which are applied to the control engineering field.

Time and space requirements of large scale mathematical problems may be reduced when the problems are solved in a multiprocessor media. Recently developed algorithms in the fields of sorting, polynomial evaluation and graph operations permits making a tradeoff between space, time and processors in a multiprocessor or a distributed system [55,56, 57,58,59].

Artificial intelligence subjects such as game playing and robotics are the application area of concurrent programming. Parallel algorithms for alpha-beta search or implementation of transportation tables in game trees improve the search speed. A discussion on parallel searching on game trees may be found in Marshlands' survey [60]. In another survey by Bonner [61], robot languages are classified and their extensibility, concurrency and modularity features are evaluated. A robot language would include the ability to operate different devices concurrently. Concurrent operations are performed either by wait/signal mechanism or parallel block execution. Wait/signal primitives are commonly used to synchronize processes in robot control programs. The parallel block execution method is preferred when concurrent processes are executed in different devices and they communicate with each other.

## 2.5 Description of CPS/1100

CPS/1100 was primarily designed to aid the writing and understanding of concurrent programs in various fields of computer science. The primary concern during the design and implementation phases was to produce a concurrent system suitable for writing model operating systems, real-time applications and discrete simulation.

CPS/1100 is designed to provide facilities to solve concurrent programming problems such as deadlock, mutual exclusion and nondeterminism. Special attention was devoted to facilitate the easy debugging of CPS/1100 programs. Two debugging aids are made available to users. Users may get a dump of process synchronization variables or may examine status of activities.

Processor dedication facility of CPS/1100 provides physical concurrency in program execution. Therefore, efficient parallel algorithms for mathematical and engineering problems can be programmed by using CPS/1100.

Structurally, CPS/1100 is an extension of the Pascal 8R1 compiler which was developed in Institute of Datalogy of University of Copenhagen. Pascal 8R1 compiler permits assembler code insertion into programs and separate routine compilation. Syntactically, CPS/1100 and Pascal have the same structure. CPS/1100 has a prescanner to perform syntactical and semantical analyses of its commands and control directives. Prescanning is achieved in three phases. Prescanner programs were written by using a high level string processing language, Macro. At the end of the prescanning operation, a program, which is a mixture of Pascal and assembler codes, is created by CPS/1100. These assembler codes are used to call executive requests [62] for exception handling, processor dedication, activity creation, removal and control. Also, test-and-set type assembler instructions are used to implement and maintain critical regions. Afterwards, the Pascal 8R1 compiler generates an assembler program text instead of an object code stream. This assembler stream is then processed by the CPS/1100 postscanner which is also a Macro program. The postscanner recognizes the messages inserted during prescanning. It processes the

assembler code stream according to these messages. The final form of the program is a Univac assembler program which can only access the Exec8 and Pascal 8R1 libraries. This final form of the program is assembled and linked to produce an executable element. Block diagram of the CPS/1100 structure is given in Figure 2.3.

The basic restriction of CPS/1100 software is that procedure activation records are not allocated and deallocated dynamically. Therefore, CPS/1100 does not permit processing of multilevel routines. In other words, CPS/1100 allows only single level routines and its memory allocation mechanism is static.

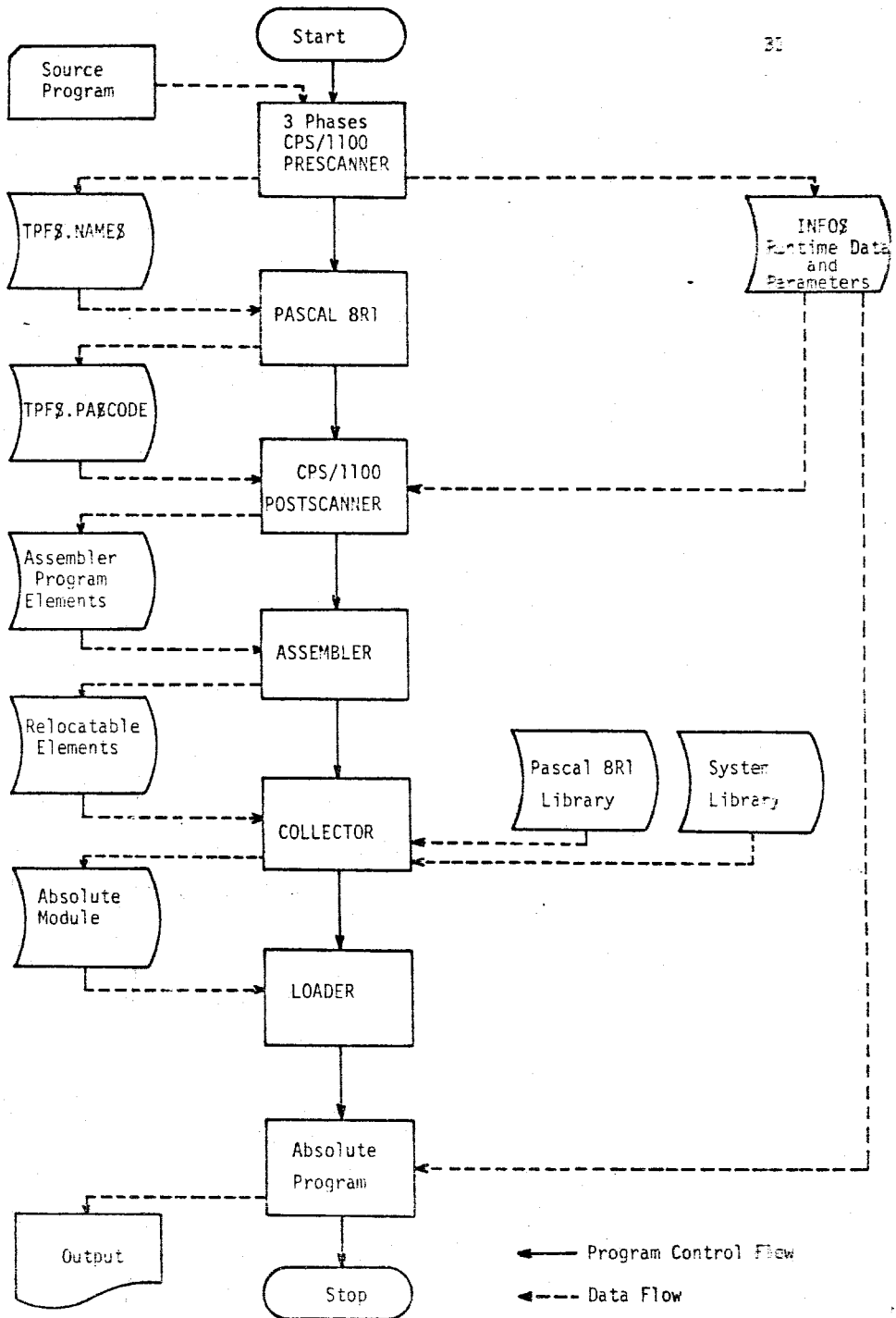


Figure 2.3 - Block Diagram of the CPS/1100 structure.



### III. MULTITASKING ON UNIVAC 1100 SYSTEMS

Generally the name activity, which corresponds to the name process, is used to denote a program control flow in Univac terminology. The operating system of Univac 1100 machines, Exec8, provides services for creating multiple activities and for communication between activities. Several activities could simultaneously be in execution in programs. Activities created by Exec8 are identified in either of two ways: by an 'activity-id' or by an 'activity-name'. A program initially consists of a single activity. This is the initial activity of the program. Other activities can be created by using a fork\$ request. Each child activity may have either the minor register set, which includes X8-X11, A0-A5 and R1-R5 registers, or the major register set, which includes all user registers. Details of multitasking are given in this chapter.

#### 3.1 Multiple Activity Programs and Activity Control

The operating system of Univac 1100 machines, called Exec8, provides services for creating, removing and controlling multiple activities. Activities may be identified by an activity-name, or by an activity-id or both. An activity-id is a number in the range of 1 to 35. An activity-name contains a maximum of three characters. An activity can be named by using the executive request name\$. Activity names are used to activate or to interrupt the execution of tasks. Activity-ids are used to wait for the termination of other activities.

A program initially consists of a single activity called the initial activity. Other activities of a multitask program are created by using an executive request fork\$. The entire program can be terminated by using the executive request abort\$, while a single activity can be terminated by using exit\$.

Activities can be controlled by `await$`, `act$`, `dact$`, `twait$`, `int$` and `aded$` requests. An activity may be delayed until the completion of other activities via the `await$` request. This request can be used by an activity to suspend its execution until the completion of specific activities. The waiting activity and the activities being waited for are distinguished by their activity-ids. `Act$` and `dact$` requests may be used to activate and deactivate activities. `Dact$` request suspends the activity that requests it. `Act$` request causes the resumption of a named activity. An activity that issues an `act$` request must have an activity name assigned through the use of a `name$` request. Activities may interrupt each other by using the `int$` request. `Int$` request creates a contingency condition, which is an abnormal occurrence, in a indicated activity. Control of interrupted activity is directed to a predefined routine when an interactivity interrupt is handled via `int$` request. A routine, called contingency routine, must be registered via `all$` request by the interrupted activity. Another necessary condition to achieve an interactivity interrupt is that the interrupted activity name.

Test and set mechanism can be used to create and control critical regions. An activity may lock a critical region by using a test and set instruction. Another activity may reference the critical region by using a test and set instruction and if the region is locked, that activity enters a busy wait state (periodic checking) until the first activity unlocks the critical region.

Timed waiting is another technique used in synchronization operations. The execution of an activity can be delayed for a specified time interval by using the `twait$` request.

Activities may have either a subset of user registers, called minor register set, or all of the user registers according to selected parameters during their creation.

Test-and-set queuing mechanism can be used to control interactions between activities. In this mechanism, an activity sets a test-and-set flag and then deactivates itself by clearing the test-and-set flag, until another activity sets the test-and-set flag and activates the first activity by clearing the test-and-set flag. `Exec8` uses a queue for each test-and-set flag if a test-and-set queuing registration is

performed. The registration can be performed by calling the executive request `tsqrg$`.

Contingency handling mechanism of Exec8 is applicable to many exception conditions, other than interactivity interrupts, such as overflow/underflow, i/o errors. Exec8 stores information about interrupted activity into an information packet when a contingency condition is raised. At that moment, the interrupted activity has a high internal priority, called 'in contingency mode'. The activity leaves contingency mode when it calls an executive request. Error-address field of contingency packet, which contains the location counter when the activity is interrupted can be used to resume the execution of the activity.

### 3.2 Activity Creation, Removal and Naming

A `fork$` request with a parameter word can be used to create a new activity. The `s2` field, bits 24-29, of parameter word contains the activity-id which is unique and in the range of 1-35. If the `s2` field is zero then the activity does not have any activity-id. The `h2` field of the parameter word, corresponding to bits 0-17, is the address of first instruction to be executed by the new activity. The `s3` field of parameter word, corresponds to bits 18-23, must be zero to indicate that all user registers are active. The parameter word must be loaded into `A0` register before the executive request `fork$` is called.

Following instructions are sufficient to create an activity.

```

L   A0, (011, newactivity) . Load the parameter word
ER  FORK$                  . Issue an executive
-                               request to activate
-                               the process indicated
-                               by the parameter word
newactivity
-
-

```

The executive request `exit$` should be used to remove an activity from the system.

The `name$` request is used to assign a name to an activity. It converts an 18 bit, i.e. three characters, name to a one word activity name. The 18 bit user defined name must be unique. It must be loaded into the `A0` register before the `name$` request is called.

An activity is named and that name is stored at a specific location by the following codes:

L, U	A0, 'ABC'
ER	NAME\$
S	A0, activityname

### 3.3 Activation and Deactivation of a Named Activity

The executive request `act$` activates an activity which is already named via a `name$` request. It resumes the execution of the named activity after the activity deactivate itself by using a `dact$` request. The name of the activity must be loaded into the `A0` register before the executive request `act$` is called. If the activated task is currently active it will not be deactivated after its next `dact$` request.

In the following example, an activity is activated:

L	A0, activityname
ER	ACT\$

A named activity may deactivate itself by using `dact$` request. `Dact$` request prevents the execution of the activity until it is activated by another activity. If other activities have issued `act$` request for an activity before it performs a `dact$` request, deactivation does not occur and execution continues. If an activity remains deactivated after all activities have terminated, an `await$/dact$` ambiguity error is reported by `Exec8`.

An activity deactivates itself through the following instruction:

ER	DACT\$
----	--------

### 3.4 Waiting for Activity Completion

The execution of an activity can be delayed until the completion of one or more activities by using the `await$` request. Initial activity of a program can not call `await$` request because it does not have an activity-id. A bit mask is used to indicate the activities being waited for. Bit 0 of the mask must always be zero. The activity-ids of the activities being waited for, must be used to indicate the bit positions which would be set to 1. The mask word must be loaded into `A0` register just before calling of `await$` request as illustrated below:

```
L      A0,mask word
ER     AWAIT$
```

### 3.5 Contingency Registration and Interactivity Interrupt

The executive request `iall$` can be used to register a contingency routine. A mask word is used to indicate the types of contingencies that may be handled by the contingency routine. Bit 34 of the mask word must be set to 1 to handle interactivity interrupt type contingency. H2 field (bits 0-17) of the mask work contains contingency routine address. The mask word must be loaded into `A0` register before requesting `iall$`. The first two words of contingency routine is called the contingency packet. The contingency packet consists of information about contingency processing. The address, that is the location counter at the moment interrupt occurred, is stored in the H2 field of the first word of contingency packet. Interrupted activity can be resumed by transferring control to the successor of that address. The contingency packet is immediately followed by the codes of a contingency routine. An activity in a contingency routine is in contingency mode until it calls an executive request.

Another executive request, `creg$`, is used for contingency registration. `Creg$` request allows the contingency packet to be separated from the contingency routine. Further details on contingency registration are given in the Univac Executive System manuals [8,62].

The executive request `int$` can be used to handle interactivity interrupt. An activity that may be interrupted must register contingency before it is interrupted. If an interrupted activity is already deactivated via an `await$` or `dact$`, it is reactivated to handle the interrupt. If an interrupted activity has been deactivated via a test-and-set queuing request, it will be interrupted after activation is performed by another activity via test-and-set clear and activate request. The name of activity being interrupted is loaded into `A0` register before `int$` requested as illustrated below:

#### Interrupting Activity

```
-
L,U      A0, name-of-the-interrupted-activity
ER       INT$
-
```

#### Interrupted activity:

```
-
L,U      A0,activityname
ER       NAME$ . Activity naming
L        A0,(02000000,contingency-routine)
ER       IALL$ . Contingency registration
-
-
-
-
-
ER       EXIT$
Contingency-routine
RES      +2 . Contingency packet
-
-
-
J        somewhere . Jumping
```

### 3.6 Test-and-Set Instruction and Test-and-Set Clearing

Test-and-set instruction (ts) is designed to cooperate asynchronous tasks via a common location for the tasks, called test-and-set cell. TS instruction checks bit 30 of test-and-set cell whether it is 0 or 1. If bit 30 of the cell is zero, the next instruction of the program is executed and this bit is set to 1. If bit 30 is set to 1, an interrupt occurs and the processor does not service this activity for a while. The activity receives control back and checks bit 30 of the cell again. A test-and-set cell can be created via a special assembler procedure `tscell`. Bit 30 of a test-and-set cell can be reset to zero by using `csts` executive request. `ts` and `csts` pair can be used to create critical regions as illustrated below:

```

cell          -
              TSCELL          . test-and-set cell creation
              -
              -
              TS      cell
              -          . Critical region
              -
              CSTS      cell

```

### 3.7 Test-and-Set Queuing Registration and Deregistration, Test-and-Set Queuing and Activation

The executive request, test-and-set queuing registration, `tsqrq%` is used to register automatic queuing of test-and-set conflicts within a program. On the contrary, test-and-set queuing deregistration request, `tsqcl%`, suppresses queuing operations on test-and-set conflicts, but activities already on queues will be serviced by clear test-and-set and activate, `cstsa`, requests.

The assembler procedure `cstsq` clears a test-and-set cell and deactivates the requesting activity. The cell referenced by that request must be declared before. A test-and-set cell must be processed by a `ts`

instruction before it is referenced in a `c&tsq` request. Otherwise, the result of the `c&tsq` request may cause an ambiguity on the program execution.

The assembler procedure `c&tsa` clears a test-and-set cell and activates one activity, queued to the cell, which was deactivated by a `c&tsq` request. The cell must be set via a `ts` instruction before `c&tsa` request is used. `c&tsa` causes a search on a specific queue for an entry created by `c&tsq` request. If such an entry is found, it is activated and the test-and-set conflict is cleared.

The following example is a solution for a 'producer-consumer' problem by using `c&tsq` and `c&tsa` requests:

Activity A1, of program A, is passing information via a common data bank to activity B1 of program B. The following subroutines will provide the data protection and activity synchronization. CELL is in a common bank and it has test-and-set and queuing registered.

Within activity B1:

```

Lock data area (TS CELL)
Check for input
If input exists, remove it, clear (C&TS CELL), return
If no input, clear lock and deactivate (C&TSQ CELL)

```

Within activity A1:

```

Lock data area (TS CELL)
Deposit input for B1
Clear lock and activate (C&TSA CELL)
Return

```

### 3.8 Activity/Processor Dedication

The executive request `aded&` dedicates a processor to a specific activity. A mask word should be used to indicate the processor on which the `aded&` requesting activity is to be processed. The  $2^n$  th bit indicates



processor  $n$  of a multiprocessor Univac 1100 system. The mask word must be loaded into the  $A0$  register before  $aded\&$  is requested. At most one processor may be requested. A mask setting of zero, releases an activity. If the requested processor is down or nonexistent, the activity will be terminated in error.

Clearing mask and then calling  $aded\&$  request after an activity/processor dedication operation causes the return of the activity to normal processing mode in which activities may be serviced by any available processor.

#### IV. CPS/1100 KEYWORDS, STATEMENTS, STRUCTURE AND IMPLEMENTATION DETAILS

A set of keywords was developed to enhance parallel programming and process synchronization in CPS/1100. A complete list of keywords is given in Appendix C. CPS/1100 is structurally divided into three phases: prescanning, pascal compilation and assembler code modification. A series of Macro programs is used in the prescanning and assembler code modification phases.

In the design phase of CPS/1100, the syntax of the Pascal language was preferred as the syntax of CPS/1100 for the following reasons:

- CPS/1100 was an extension of Pascal; it should obey the syntactical rules of Pascal as much as possible.
- One of the design aims of CPS/1100 was to develop a concurrent language for average programmers. Therefore, it would be better that CPS/1100 has the same syntax of Pascal.

The open/close parenthesis characters are used as delimiters in some of CPS/1100 statements, while the semicolon character is used to denote a end-of-statement. Therefore, a close syntactical similarity can be observed between statements of CPS/1100 and statements of Pascal 8R1 and PL/1. For example, process and monitor declarative statements of CPS/1100 have the same syntax as the external routine declaration of Pascal 8R1. The lock/unlock statements of CPS/1100 are very similar to the lock/unlock statements of PL/1.

In this chapter of dissertation, detailed information about CPS/1100 statements, structure and implementation is given.

## 4.1 Declarations

### 4.1.1 Process and Monitor Declarations

A CPS/1100 program is composed of a main and several external procedures. External procedures are declared as either monitors or processes. Examples for process and monitor declarations are given below:

```
procedure forks (var fentry:labels; var philid:integer); monitor;  
procedure phils (var pentry:philent); process;
```

Rules:

1. All process and monitor declarations must exist in the main program.
2. Only declarations of processes or monitors that will be called in a certain process are required within that process.
3. Monitor or process declarations must be placed before variable declarations in the main program or procedures.

### 4.1.2 Queue Type Declaration

Queue variable type is used only for condition variables of a CPS/1100 program. A queue variable can be used only for process synchronization purposes. An example for queue type declaration is given below.

```
var syncvar : queue;
```

Rule:

1. The maximum length of a queue variable is 12 characters.

### 4.1.3 Forward Declaration

Forward declaration placed after an argument in a start-up or fork statement prevents the transfer of the value or address of that specific argument to the procedure called. As an example:

```
fork with taskname (xx) procname (argument1, argument2 forward,...);
```

The forward keyword in the example prevents the transfer of the value or the address of the argument2 to the called procedure procname. This approach is similar to the 'optional parameter passing' mechanism on procedure calls.

### 4.1.4 i/o Initiation and i/o Termination Locks

Queue type variables are used to create a critical section around i/o statements in slavetasks. A lock command which uses i/o initiation lock is placed just before a read/write statement in a slavetask routine by CPS/1100. An unlock command which uses i/o termination lock is placed to the line that succeeds a read/write statement in a slavetask routine. These variables are common to model and slavetask routines. They may be used for synchronization purpose. Following statements are examples for the declarations of lock variables:

```
aaa : ioinitiation lock of queue;
bbb : iotermination lock of queue;
```

Rule:

1. Variables must be declared in the CPS/1100 main routine.

### 4.1.5 i/o Initiation and i/o Termination Flags

These variables are used in wait and signal statements if slave-tasks perform input/output. CPS/1100 model tasks can control slave-tasks by these flags when slavetasks perform i/o operations. These

variables are common to model and slave activities. The following statements are examples for the declaration of these flags.

```
aaa : ioinitiation flag of queue;
bbb : iotermination flag of queue;
```

Rule:

1. Variables must be declared in the CPS/1100 main routine.

## 4.2 Keywords and Statements

### 4.2.1 Startup with Taskname and Startup Slavetask Statements

A startup with taskname statement creates a new activity with an activity-id associated with the taskname given. The calling activity enters a wait state until it is resumed by a release statement in the called procedure. In the meantime, the new activity provides for storage requirements of arguments, of local variables and of link data segment. It also sets a specific global variable associated with that procedure as the activation record pointer. The following statement creates an activity and transfers control to procedure gauss.

```
startup with taskname (task1) gauss (a,b,c);
```

Rules:

1. A taskname given to a specific activity must be unique. However the same taskname can be used for another activity if the programmer is sure that the previous activity has exited.
2. A startup statement must be used on the first call of any procedure and used only once.

A startup slavetask statement creates an activity with an activity name. Activity name can be a string up to three characters in length and has to be declared in the CPS/1100 control card of the slavetask.

Startup slavetask statement must be the first executable statement of the CPS/1100 main routine. The main activity enters a wait state, when a slavetask is started, until the name of the started slavetask is stored in a specific activity name field by the created activity. In the following statements 20 slavetasks are created.

```

i:= 1;
while i = < 20 do begin
           startup slavetask (i); i:= i+1;
           end;

```

Rule:

1. All slavetasks must be started up in the CPS/1100 main routine before any activity is created.

#### 4.2.2 Fork Statement

Fork statement creates a new activity with an activity-id associated with the taskname given. When the new activity enters into the called procedure, it sets a specific index register as the activation record pointer of the called procedure by obtaining the record address from a global variable which has been set by a startup statement. An example for the fork statement is given below:

```

fork with taskname (task09) gauss (a,b,c);

```

Rule:

1. The taskname given to a certain activity must be unique. However the same taskname can be used for another activity if the programmer is sure that the previous activity has exited.

### 4.2.3 Release Statement

Release statement causes the father activity, that is in a wait state after a startup statement, to resume its control again. Format of the release statement is given below:

```
release (<proc-id>);
```

where proc-id is the name of the procedure called in startup statement. For example, let the procedure gauss be called by a startup statement in the main routine and the control is suspended. New activity resumes the control of main activity by executing the following statement in procedure gauss:

```
release (gauss);
```

Rule:

1. Release statement must be used only once in a procedure.

### 4.2.4 Await Statement

This statement suspends the execution of an activity while specified activities exist in the system. The following statement suspends the execution of the calling activity while the activity named task09 exists in the system.

```
await (task09);
```

An activity may suspend its execution while more than one activity exists in the system with an await statement as shown below:

```
await (task09,task21,abcd);
```

Rule:

1. Taskname(s) in an await statement must have been used in a startup, fork or cobegin statement before it is used in an await statement.

#### 4.2.5 Wait and Signal Statements

Wait statement is used to suspend the execution of an activity and queue it to a test-and-set cell. An activity that executes a wait statement is deactivated and placed in a queue associated with the test-and-set cell. The following statement is an example for wait statement use:

```
wait (msg);
```

where msg is a queue type variable.

Signal statement is used to resume the execution of an activity previously placed in a test-and-set queue. It causes a search in the queue associated with the specified test-and-set cell and the first activity in the queue is reactivated. For example the following statement reactivates the activity that executed wait(msg) statement:

```
signal (msg);
```

Rule:

1. The condition variables referenced in wait and signal statements must have been declared as queue type variables in the main CPS/1100 routine.

#### 4.2.6 Lock and Unlock Statements

Lock and unlock statements are used to create critical regions. An activity that executes a lock statement sets a test-and-set cell so that other activities, that access the cell, wait until that cell is cleared by an unlock statements. Unlock statement clears the specified test-and-set cell. In the following example, variable gate is a queue type variable.

```
lock(gate);
<Critical Region>
unlock(gate);
```



Rule:

1. The condition variables referenced in lock and unlock statements must be declared as queue type variables in the main CPS/1100 routine.

#### 4.2.7 Cobegin and Coend Statements

Cobegin/coend and associated await statements are used to simultaneously commence the execution of new activities which are started up in the program segment specified by cobegin/coend statements. Cobegin statement starts an activity with a given name and deactivates it. Coend statement causes reactivation of that activity and exits it from the system. Startup and fork statements between cobegin and coend statements create new activities. These new activities must be suspended via await statements on the called routines until the coend statement is executed. In the following example producer and consumer processes are created and then their controls are suspended by await statements. Execution of both processes are allowed by execution of coend statement in the main CPS/1100 routine.

```
*COMPILE  OPTIONS=ISRLM,ELTNAME=TPF$.MAIN
  program main;
      -
  var commence : queue;
      -
      -
  begin
      cobegin (commence);
          fork with taskname (task01) producer;
          fork with taskname (task02) consumer;
      coend;
      -
      -
      -
  end.
```

```
*COMPILE  OPTIONS=ISRLMV,ELTNAME=TPF$.PRODUCER
  program define (producer);
  procedure producer;
  begin
    await (commence);
    -
    -
    -
  end;
```

```
*COMPILE  OPTIONS=ISRLMV,ELTNAME=TPF$.CONSUMER
  program define (consumer);
  procedure consumer;
  begin
    await (commence);
    -
    -
    -
  end;
```

Rule:

1. Nested cobegin/coend usage is forbidden.

#### 4.2.8 Clear Statement

It stores zero to a queue type variable. For example:

```
clear (flag);
```

statement stores zero to the queue type variable flag.

Rule:

1. The variable referenced in a clear statement must be declared as a queue type variable in the CPS/1100 main program.

#### 4.2.9 Quit and Quit by Checking Statements

Quit statement causes an activity to exit from the system. But quit by checking causes an activity to exit from system if the referenced queue type variable has been cleared. The following statements are examples for unconditional and conditional quit statements:

```
quit;  
quit by checking (flag);
```

Rule:

1. Queue type variable referenced in quit by checking statement must be declared in the main CPS/1100 routine.

#### 4.2.10 Suspend and Wait Forever Statements

Suspend statement causes an activity to delay its execution for a specified time period. Wait-time period is in the range of 2-30000 miliseconds. Wait forever causes an activity to delay its execution for an indefinite time period. In the following statements execution of an activity is delayed by suspend and wait forever statements.

```
a:= 2500; suspend(a);    or    suspend (2500);  
wait forever;
```

#### 4.2.11 Activate and Deactivate Slavetask Statements

Activate statement resumes execution of a slavetask which has just started up or already deactivated by an external activity. Deactivate statement interrupts execution of a slavetask and suspends execution until it is reactivated externally. As an example:

```
i:= 7;  
deactivate slavetask (i);  
-  
l:= 7;  
activate slavetask (l);
```

Rule:

1. The variable referenced in an activate or deactivate slavetask statement must be an integer identifier.

#### 4.2.12 Terminate Statement

This statement causes termination of all activities which are created by CPS/1100. Program execution may be ceased via a terminate statement.

#### 4.2.13 Fetch Status of Slavetasks Statement

It can be checked whether a slavetask has exited from the CPS/1100 system or not by using the fetch status of slavetasks statement. If the activity in the specified slavetask routine has exited from the system, the corresponding element of a certain boolean array is set to true by that statement. Otherwise the element of the boolean array remains as false. In the following statement, status of slavetasks in the system are retrieved.

```
fetch status of slavetasks;
```

Rule:

1. User is responsible to declare a boolean array with the name of status in the routine in which the fetch statement is used. The size of that boolean array must be equal to the number of slavetasks in the CPS/1100 run.

#### 4.2.14 Dedicate Processor and Original Mode Statements

Dedicate processor statement dedicates the specified processor to the requesting activity. Original mode statement causes the return to normal processing mode in which activities may be processed by any

available processor. In the following program segment processor1 is dedicated to an activity for a while:

```
-
-
dedicate processor(1);
```

```
-
-
-
original mode;
-
-
```

#### 4.2.15 Cregion/Cend Statements

These statements are used to create a critical region where Pascal 8R1 library routines are called. A critical region is mandatory around library entry jumps because programs in the library are not re-entrant. More than one activity must not execute library program codes at a time, otherwise programs produce unpredictable results. In the following example, a string is printed in a cregion/cend critical region.

```
cregion;
```

```
-
```

```
-
```

```
write('xxxxxxx');
```

```
cend;
```

#### 4.2.16 Snoopy on and Snoopy off Statements

Snoopy on statement calls a utility program to trace assembler routines. Snoopy off statement turns off the trace operation. In the following statements a program segment is traced:

```

-
-
snoopy on;
<Portion of CPS/1100>
<program to be traced>
snoopy off;
-
-

```

Rule:

1. Only one activity at a time must be traced in a program.

#### 4.2.17 Snapdump Statement

Snapdump statement causes a dumping of the queue type variables. The following statement is an example snapdump call:

```
snapdump (dump-id);
```

Rule:

1. Dump-id must be a name of maximum 6 character in length. It is used for identification purpose in the dump lists.

### 4.3 Implementation Details

#### 4.3.1 Structure and Storage Allocation Mechanism of PASCAL 8R1

Pascal 8R1 compiler generates assembler instructions instead of machine code of the program being compiled. It stores the assembler program into a specific program file element, tpf\$.pa\$code. The assembler codes must be assembled by Univac 1100 assembler processor that creates a relocatable element. That element, therefore, must be linked with the Pascal 8R1 library which consists of routines for allocating and deallocating

storage for procedures, i/o handling facilities, communicating with other software processors and tracing facilities. The library also consists of routines for built-in mathematical functions.

External compilation is allowed in Pascal 8R1 software. Externally compiled routines must be collected to obtain the executable element. A sample program stream with an externally compiled procedure is given below:

```
@RUN,A run-id, account-number, project-id
```

```
@BU*PASCAL.S,ISRLM
```

```
  program main;
  procedure extpas (var inp:integer); external;
  var a : integer;
  begin
    -
    extpas(a);
    -
    -
  end.
```

```
@ASM,UN TPF$.PA$CODE,TPF$.MAIN
```

```
@BU*PASCAL.S,ISRLMV
```

```
  program define (extpas);
  procedure extpas (var abc:integer);
  var kk,ll:integer;
    -
    -
  begin
    -
    -
    -
  end;
```

```
@ASM,UN TPF$.PA$CODE,TPF$.EXTER
```

```
@MAP,I ,TPF$.MAIN
```

```
  IN TPF$.MAIN
```

```
  LIB BU*PASCAL.
```

```
  END
```

```
@XQT   TPF$.MAIN
```

```
    <data>
```

```
@FIN
```

The routine starter element (RS) of Pascal 8R1 library starts the execution of a program by allocating a run time stack. The size of the stack is dependent on the options used on the processor calling card. Program control is transferred to the user program codes after the processor calling options are checked by RS routine. The first executable code in the main program is:

```
LMJ    A1,PENT$
```

```
<3 words package>
```

Program control is transferred to pent\$ entry of the RS routine of the library by lmj jump instruction. At this moment, index registers x9 and x10 contain the address of start\$ which is the tag that refers to start address of the activation record of main routine. Locations between start\$+136 and stack top are allocated to the variables declared in the main routine. The following storage map may be useful to demonstrate the storage allocation mechanism of Pascal 8R1 (Figure 4.1).

In the RS routine, the data area for the variables declared in the main routine is allocated and next available position in the stack is pointed as the stack top. Program control returns to the main program.

When an external procedure is called, the program control is transferred to the procedure. Then a jump to the pnet\$ entry of RS routine is performed by the following instruction.

```
LMJ    A1,PENT$
```

```
<3 words package>
```

At this moment register A0 contains the address of the activation record for the incarnated procedure. Arguments of the procedure has been transferred to the locations in the activation record by the instructions created for procedure calls. The following map illustrates the activation record link fields (Figure 4.2).



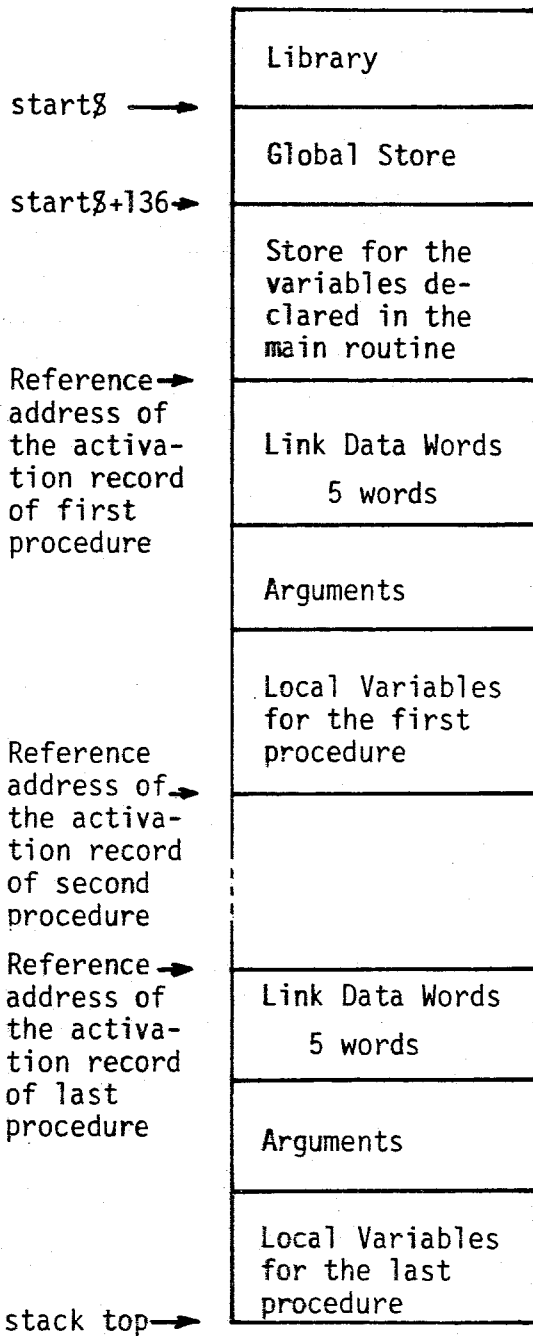


Figure 4.1 - Storage allocation map of Pascal 8R1

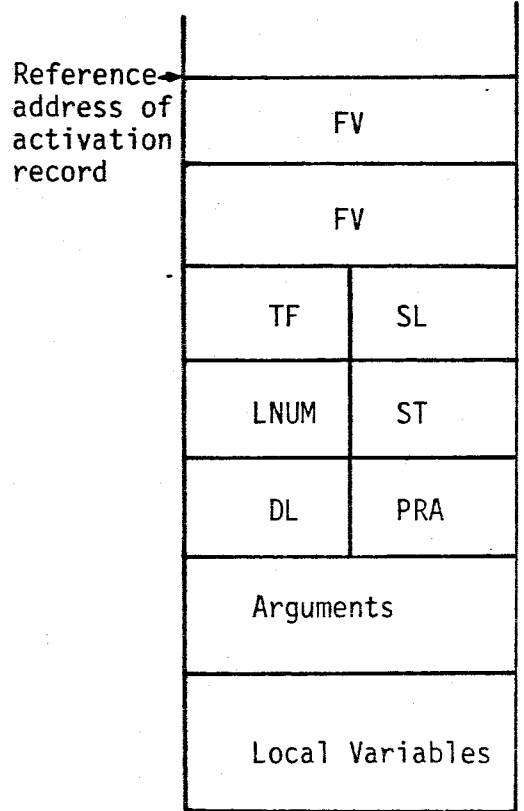


Figure 4.2 - Activation record map of Pascal 8R1

where,

- |    |                                   |       |                                  |
|----|-----------------------------------|-------|----------------------------------|
| FV | : Function Return Value (2 words) | LNUM: | Line Number of Procedure Calling |
| TF | : Trace Flag                      | ST    | : Stack Top                      |
| SL | : Static Link                     | PRA   | : Procedure Return Address       |
| DL | : Dynamic Link                    |       |                                  |

The following tasks are performed when program control is transferred to the `pent$` entry of RS routine:

1. TF, SL, LNUM, DL, PRA fields of the activation record of the called procedure are filled.
2. Index register `x10` is set as the reference pointer of the activation record.
3. Reference address of the activation record is saved in a library global variable `CX10$`.
4. Register `A0` is set as stack top pointer.
5. The size required by procedure local variables is checked. If it is not enough, the stack is expanded.
6. Level of procedure calling, which is a library global information, is updated.
7. Function return value field is filled with nil.
8. Location devoted to the local variables is filled with nil.
9. Traceword is checked for tracing process.
10. Certain library global variables are updated.
11. Program control is returned to caller routine.

When an external procedure is called, the following tasks are performed to transfer values or addresses of the arguments:

1. Line number of the procedure calling statement is stored into the register `R15`.
2. Current stack top value is stored in register `A0`. Afterwards, it is used as reference address of the activation record of the incarnated procedure.
3. Values or addresses of the arguments are stored into the locations that succeed the fifth word of the activation record.
4. Activation record address of the caller routine is loaded into register `x9`. This address is used as the static link value later.

5. Location counter of the caller routine is loaded into register x11 and program control is transferred to the called procedure.

The assembler code equivalents of the explained actions upon a procedure call is given below:

```

L,U      R15, <pascal program line number>
L,H2     A0,3,x10
L,U      ax, <address of argument1>
S        ax,5,A0
-
-
L        ax, <address of argument i>
S        ax,n,A0
-
-
L,U      x9, <address for static link>
LMJ      x11, <name of the called procedure>

```

where ax is an A or x register  
n is an integer with the value of 4+i.

#### 4.3.2 Modifications on Pascal 8R1 to Provide Parallel Processing

In sequential processing lmj instruction is used to jump to a procedure. In parallel processing er forks\$ request must be used instead of lmj instruction:

In sequential processing  
LMJ x11, entryname

In parallel processing  
L aa,A0  
L A0,(00xx01,entryname)  
ER FORKS\$  
L A0,aa

where aa is a register which is used to preserve content of A0 register. R15 register is used in implementation.

xx is the activity-id of the created activity.

In CPS/1100 programs, activation record of a procedure is allocated once and is not released until the program is finished. A flag is used to prevent reallocation of the activation record of a procedure. When a procedure is started up, the following instructions are executed.

1. TS <test-and-set cell for the called procedure>
2. L,U R15, <pascal program line number>
3. L,H2 A0,3,x10
- 
- 
- {Argument transfer}
- 
- 
4. L,U x9, <address for static link>
5. L R15,A0
6. L A0,(00xx01, name of the called procedure)
7. ER FORK\$
8. C\$TSQ <test-and-set cell for the called procedure>
9. L A0, R15

Instructions in line 1 and in line 8 create a critical region and cause suspension of activity control until the created activity, by fork\$ request, performs a signal operation on a CPS/1100 defined queue type variable. In line 2, statement number is stored in register R15 for tracing purpose. Activation record reference address of the called procedure is loaded into A0 register, in line 3. Argument transfer operation is performed by using this reference address. In line 4, the address that will be used as the static link in the called procedure is loaded into x9 register. In line 6, xx field is the activity-id of created activity and the adjacent 01 value denotes that the new activity uses the major registers set, i.e. all

the user accessible registers. Name of the procedure called is loaded into the H2 field, i.e. lower half, of the A0 register while 00xx01 is loaded into the H1 field. R15 register is used as a temporary storage to preserve the content of A0 register between the lines 5 and 9. Executive request fork\$, in line 7, creates an activity according to the specifications stated in register A0 and forks it to the procedure address which was stored in the H2 field of A0 register.

The newly created activity consists of a set of registers with the same contents of its creator activity after the creation operation. It processes the following instructions when it enters into the called procedure.

1. L           A0,R15
2. TNZ        <flag>
3. J           ENTER
4. L           x10, <address of global variable>
5. J           PROCST
6. ENTER
7. LMJ        A1,PENT\$  
      <3 words package>
8. SZ         <flag>
9. S          x10, <address of global variable>
10. PROCST

R15 register has been used to preserve the contents of A0 register by the creator activity. In line 1, the content of A0 is recovered. Flag used in line 2 is a CPS/1100 defined location which is initially nonzero. This flag is used to control activation record allocation. If it is nonzero, activity control is transferred to the label enter. In line 7, control is transferred to the entrypoint pent\$ of the RS routine of Pascal 8R1 library. When the activity control returns back to line 8, the activation record is allocated in memory. In line 8, the flag is set to zero. In line 9, content of x10 register, which is the reference address of the activation record, is stored into a CPS/1100 defined global variable.

When another activity is transferred to the procedure via a fork statement, it tests the flag, in line 2, and jumps to line 4. It retrieves the reference address of the activation record from the global variable and it loads this into register x10. Control passes on to the label procst in line 10.

The activity that used startup statement is in wait state because of the associated c&tsq request. Control of that activity should be resumed by a release statement which is placed in an appropriate line in the called procedure. The appropriate line in the procedure may be the first executable statement in the procedure or the end of variable initialization section of the procedure. A release statement and the corresponding assembler codes are illustrated below:

```
release (<procedure name>);
```

```
TS      <test-and-set cell for the procedure>
```

```
C&TSA  <test-and-set cell for the procedure>
```

Fork statement is used to create an activity and to transfer new activity to a procedure which is already allocated in memory via a startup statement. Assembler code correspondants of a fork statement is given below:

```
L,U      R15, <pascal program line number>
```

```
-  
{Argument transfer}
```

```
-  
L        R15,A0
```

```
L        A0,(00xx01, name of the called procedure)
```

```
ER       FORKS
```

```
L        A0,R15
```

Slavetasks are lowest priority activities that may be used to simulate user jobs executing under an operating system. Parameter passing to a slavetask is not allowed. Slavetasks may only be started up, activated or deactivated by other activities. The following instructions correspond to the startup slavetask (i) statement:

1. L                   x10, globvar
2. L,H2               reg1,3,x10
3. L,U                x9, <address for static link>

where globvar is the address of the global variable that contains the reference address of the most recently allocated activation record, reg1 is a register in which the reference address of slavetask procedure that will be created is temporarily stored.

4. L                   reg2, <address of variable i>
5. L                   label 1

where reg2 is the register in which slavetask index i is stored, label 1 is the section where the slavetask index range check is performed and control is transferred to the corresponding section to prepare a register for activity creation.

6. CØ§i
7. L                   AØ,(ØØØØØ1,SLAVETASK i)
8. SZ                  INTACTNAMEi
9. L,U                reg3,INTACTNAMEi
10. J                  label3

Codes in line 6 to 9 are repeated imax, that is the maximum number of slavetasks in the program, times. In line 6, AØ register is prepared for making a fork§ request. Slavetasks do not have activity-ids, for this reason the upper half of the AØ register is set to ØØØØØ1. Intactnamei is a CPS/1100 defined location in the common store of activities. Internal activity name of a slavetask is stored in the corresponding intactnamei word and the address of intactnamei is stored in a register. Label3 is the label of a section where fork§ executive request is used to create the slavetask.

11. label 1
12. TG,U              AØ,1
13. TG,U              AØ,imax+1

```

14. J          label2
15. ANU,U     A0,1
16. J          *§+1,A1
17.   +C0§1
      +C0§2
      -
      -
18.   +C0§imax
19. label2
20. LMJ       x11,PXCS§

```

In lines 12 to 14, the range of variable *i*, which is the content of *A0* register at the moment, is checked. If it is out of range, control is transferred to the *pxcs§* entry of the RS routine of Pascal 8R1 library and program is abort. In lines 14 to 18, control is transferred to label *C0§i*.

```

21. label3
22.   ER       FORK§
23.   TZ       ,*reg3
24.   J        §+4
25.   L,U     A1,10
26.   ER       TWAIT§
27.   J        §-4
28.   L        x10,globvar

```

In line 22 a new activity is created and transferred to the *slavetaski* procedure. Content of the address in *reg3* register which is the *intactnamei* is tested for zero. If *intactnamei* is zero, it means activation record for the *slavetaski* procedure is not allocated and *slavetaski* is not named yet, control is suspended for 10 msec and then *intactnamei* is tested again. When *intactnamei* is found to be nonzero, *x10* register is updated to contain the reference address of the most recent activation record.

The created activity that is transferred into the *slavetask* procedure executes the following instructions:



1. L           A0,reg1
2. LMJ         A1,PENT\$  
      <3 word package>
3. L           A0,(0200001,CONTNGi)
4. ER         IALL\$
5. L           A0,('<3 character activity name>')
6. ER         NAME\$
7. ER         IDENT\$
8. S           A0,INTACTNAMEi
9. ER         DACT\$
10. ER        LEVEL\$

In line1, reference address of activation record of the slavetask procedure, which is temporarily stored in the reg1, is loaded into A0 register. Control is transferred into pent\$ entry of RS routine to allocate the activation record. In line 3 and 4, activity is registered for interactivity interrupt. Contngi is the label where contingency packet and contingency routine is placed. Executive request iall\$ registers the activity only for interactivity interrupt due to the mask value of 0200001. In line 5, a name which is 3 characters in length is loaded into the A0 register. An internal name for the activity is created by the Executive System in line 6. In lines 7 and 8, the system generated internal name is retrieved by ident\$ request into A0 register and it is stored in location intactnamei. In line 9, slave activity deactivates itself and it waits for an external reactivation. When this happens, it executes er level\$ instruction to decrease the internal priority of the activity to the lowest level in the system.

A slavetask can be activated by another activity through the execution of an activate slavetask (i) statement. The following instructions are executed for activity i whenever an activate slavetask statement is requested.

```

L           A0,INTACTNAMEi
ER         ACT$

```

A slavetask can be deactivated externally. This operation is performed by creating a contingency interrupt on the slave activity. The following instructions are executed when a deactivate slavetask (i) statement is processed by an activity.

```
L          A0,INTACTNAMEi
ER        INT§
```

These statements cause an interrupt to the slavetask<sub>i</sub>. Control of the activity processing on the slavetask<sub>i</sub> procedure is transferred to the address labelled as contng 1, where the following instructions are executed.

```
1.  CONTNGi
2.   RES      2
3.   S        A0,SAVEA0i
4.   L,H2     A0,CONTNGi
5.   A,U      A0,1
6.   S        A0,RTADDRi
7.   L        A0,SAVEA0i
8.   ER       DACT§
9.   ER       LEVEL§
10.  J        *RTADDRi
```

The memory locations reserved in line 2 is the contingency packet that is filled when the contingency mode is raised. Content of A0 is saved into a memory location in line 3. In line 4, lower half of the first word of the contingency packet, which is the value of location counter when the interruption occurred, is loaded into the A0 register. Content of A0 is incremented by one, thus next instruction that will be executed by the activity is found. The address of the next instruction to be executed is stored in location rtaddr i in line 6. In line 7, the old content of A0 register is recovered. The activity leaves contingency mode and deactivates itself by executing dact§ request in line 8. When that activity is activated externally, it drops its priority via a level§ request. Control of the activity is then returned to the next instruction in the calling program.

Wait and signal primitives are implemented in CPS/1100 in the following way:

```

1.  cell1      T&CELL
2.  cell2      T&CELL
3.              ER      TSQRG&
   -
   -
4.              TS      cell1
5.              C&TSA   cell1
   -
   -
6.              TS      cell2
7.              C&TSQ   cell2

```

In lines 1 and 2 two locations cell1 and cell2 are declared as test-and-set cells via the system procedure t&cell. In line 3, test-and-set and queueing registration is performed. In lines 4 and 5 a signal operation is processed on cell1. In lines 6 and 7 a wait operation is processed on cell2.

Lock and unlock primitives are implemented in the following way:

```

8.              TS      cell
   {Critical Region}
9.              C&TSA   cell

```

Implementation details of other CPS/1100 statements can be found on Macro routines in Appendix D and assembler code listing for case one in Appendix E1.

#### 4.3.3 Prescanner and Postscanner Routines of CPS/1100

The prescanner routines perform the syntactic and semantic analyses of a CPS/1100 program source. Primarily, these routines recognize and interpret the control directives encountered in a source program and

modify the source stream by inserting assembler and Pascal statements. On the other hand, the postscanner routine processes the assembler program generated by the Pascal compiler. Both the prescanner and the postscanner phases of CPS/1100 were implemented using Macro, which is described in Appendix A.

The prescanner is driven by three Macro programs named phase0, phase1 and phase2. Phase0 of the prescanner performs the syntactic analysis of CPS/1100 control directives and statements. The input to this phase is the original CPS/1100 program text and the output is stored in a temporary file to be used in the other phases of prescanning.

Phase1 of the prescanner analyzes the parameters given to the slave-tasks of a CPS/1100 program. During this phase an analysis of i/o operation flags is performed according to user declarations. The slavetasks are also determined during this phase. Flags, the number of slavetasks and the characteristics of slavetasks are written into a temporary file for use during phase2.

Prescanner phase2 inserts messages and assembler statements that correspond to CPS/1100 statements into its output stream. The output of this phase (which is passed onto the Pascal compiler) is stored in file element `tpf%.name%`. Messages and assembler instructions in the program text are ignored by the Pascal compiler in analysis phase. They are directly inserted into the assembler program produced by the Pascal compiler.

The postscanner of CPS/1100 modifies the assembler codes produced by the Pascal compiler according to the messages that were inserted into the source stream during phase2 of the prescanner.

Details of scanner programs are given in the succeeding sections of this chapter. Complete listings of those programs are given in Appendix D.

#### 4.3.3.1 Description of Prescanner PhaseØ

In this macro program, procedure readln is used to read a line from the input stream and to tokenize the line into an array. If the first two characters in the line is recognized as the string /c, then this line is assumed to be a comment line and it is skipped. The procedure nexttoken is used to retrieve the next token in the input stream line. If all of the tokens of the line are processed, this line is written into the program file element scratch\$.scratchl and then the procedure readln is called again.

In the main body of program phaseØ, the program file element scratch\$.scratchl is opened. The first line of the input stream is read by using the procedure readln and the line is tokenized by using the procedure nexttoken. The variable tok is used to store the current token. The variable tok is compared with the first keywords of the CPS/1100 statements. If a match occurs, a routine which corresponds to the CPS/1100 statement is called. For example, the procedure proccps is called when the content of the variable tok is equal to the string \*compile. In this case, succeeding tokens are analysed by matching with the fields of a CPS/1100 control directive by using the procedure proccps. An error message is displayed on the output stream when a syntax error is found in the input stream. In a similar way the other procedures in the program are used to analyse CPS/1100 statements in the input text. CPS/1100 statements and the corresponding procedures are listed below.

<u>CPS/1100 Statement</u>	<u>Macro Routine</u>
startup and fork	procstrtup
wait and wait forever	procwait
await	procawait
activate and deactivate	procactive
dedicate	procdedic
original mode	procorig
snoopy on and off	procsnoopy
quit and quit by checking	procquit
fetch status	procfetch

<u>CPS/1100 Statement</u>	<u>Macro Routine</u>
cobegin and signal and release and lock and unlock and suspend and clear	procother

#### 4.3.3.2 Description of Prescanner Phase1

In this program, a set of macros is used to recognize the input/output locks and flags which provide synchronization between model and slavetasks of model operating systems implemented by using CPS/1100. A temporary file info $\phi$  is opened and the variables which are used to store the names of the input/output locks and flags are initialized with a null string.

Macro routine cpsslave is used to fetch the slavetask parameters, i.e. run parameters of the user jobs in a model study, and to store these parameters into the arrays which are declared with the names of user, progsiz, priority, starttime. Number of user jobs in the model is counted by this Macro routine.

The Macro routine cps is used to convert the CPS/1100 control directives between model routines to a format that is very similar to the conventional Univac 1100 processor assignment.

The Macro routines ioinitlock, iotermlock, ioinitflag, ioterm are used to recognize the ioinitiation lock, iotermination lock, ioinitiation flag, iotermination flag respectively. The names of these locks and flags are stored in the corresponding variables.

When the end of text is recognized by the macro routine end-of-text, the number of user jobs and the names of locks and flags are written into the file info $\phi$ . Afterwards, the parameters of the user jobs are written into the file info $\phi$  and then the file is closed.

#### 4.3.3.3 Description of Prescanner Phase2

The assembler code correspondants of CPS/1100 statements and certain messages used in postscanning phase are produced by this program.

The first record of the file info%, where the number of slavetasks and i/o locks and flags has been stored, is read. The variables and control flags of the program are initialized.

The rest of the program is a collection of Macro routines which are triggered when the string series described in their picture definitions are found in the input text. Names and functions of these Macro routines are listed below:

<u>Name(s) of macro(s)</u>	<u>Function</u>
await	checks the arguments of await statements whether the activities corresponding to the arguments exist in the array activity-names. It produces assembler code correspondants of the await statement.
m1	initializes a series of variables when the keyword procedure has been encountered.
m2,m5	syntax macros, they are used in the picture definitions of some other Macro routines.
proc-dc1	recognizes process and monitor declarations, stores the name of procedures in an array.
var-dc1, var-dc2	recognize variable declarations. It stores queue type variables in an array.
m3	clears certain flags when a close parenthesis is encountered.
m4	recognizes the keyword var.
macbeg	produces codes to define the test-and-set cells and the fields to save A0, A1 registers. It also generates a set of codes for each slavetask to control their execution.
fork1, fork2	generates codes which corresponds to the fork statement of CPS/1100. Identifier in the taskname field is checked whether it is already defined in a startup statement or not. If it is not defined an error message is produced.
cobegin, coend	produce corresponding codes for cobegin and coend statements.
startup1, startup2	produce a message for postscanning phase. They produce codes that correspond to the startup statement.
wait, signal	produce codes for wait and signal statements.

<u>Name(s) of macro(s)</u>	<u>Function</u>
dedicate, normalmode	produce codes that correspond to dedicate processor and original mode statements.
program	generates a set of statements to define a slavetask routine.
progdef	recognizes the name of a model task routine, check whether this name is already declared or not. It sets certain control flags.
release	generates codes which correspond to the release statement.
lock, unlock	produce codes for lock and unlock statements.
suspend, suspend2	recognize suspend statements with two different formats. These routines generate codes corresponding to suspend statement.
cps2, cps, endoftext	produce a series text editing commands that erase the comment lines, produced by the Pascal 8R1 compiler.
snapdump, snoopyon, snoopyoff	produce codes which correspond to the snapdump, snoopy on, snoopy off statements.
clear, quitby, quit	produce codes corresponding to the clear quit by checking, quit statements.
slvstartup	generates a group of assembler instructions to create an activity which will be transferred to a specific slavetask routine during program execution.
activate, deactivate, terminate	produce assembler codes which correspond to the activate slavetask, deactivate slavetask and terminate statements of CPS/1100.
initiotermio	inserts a set of codes to the preceding and succeeding lines of a write or writeln statement in a slavetask. Model task and slavetask are synchronized by the aid of these codes when a slavetask initiates and completes an i/o operation.
endpoint	inserts a set of assembler codes to the end of slavetask routines. These codes are used to suspend slavetask control for an undefined time period.
fetching	generates codes corresponding to the fetch status command.



<u>Name(s) of macro(s)</u>	<u>Function</u>
forkey	produces a message that will be used by the postscanner program.
waitforever	generates codes corresponding to the wait forever statement.
cregion, cend	produces codes which correspond to the cregion and cend statements.

#### 4.3.3.4 Description of Postscanner

This program is composed of a set of Macro routines which modify the codes produced by the Pascal 8R1 compiler and insert assembler codes according to the messages produced during prescanner phase2. In the following list names and the functions of the macros are given.

<u>Name(s) of macro(s)</u>	<u>Function</u>
m0	recognizes the location counter sign $\$ (l)$ in the input text and sets a flag.
m1	updates the statement by which the run-time stack is allocated.
m2	inserts a group of codes to the preceding lines of the pent $\$$ entry in the input text. Jumping to the pent $\$$ entry is controlled by these codes.
m3	fetches the entry name of assembler routines.
m4, m5	recognize startup and fork messages produced by the prescanner phase2. They reset a group of control variables.
m6	deletes load modifier and jump (lmj) instruction and sets certain control flags.
m7	deletes load and load address (l and l,u) instructions which correspond to the arguments used together with a forward keyword in the CPS/1100 program text. It also updates some control variables.
m8	deletes store (s) instructions which correspond to the arguments used together with a forward keyword.
m9	modifies the assembler lines by which the activation record address of routines are loaded into the A0 register.

<u>Name(s) of macro(s)</u>	<u>Function</u>
m11	updates certain control variables when a delete argument message is recognized.
m12	updates a control variable when a slave-task entry message is recognized.
formac, dsmac, dlmac	modify codes corresponding to the for statement of Pascal. They are basically used to store the values of loop counters into a predefined table.
storeR13	loads the value 1 into R13 register and stores that value to the address when the s R15,<address> instruction is recognized.
suskey	updates a control variable when a suspend keyword message is recognized.
suskey2	generates codes to load the suspension period into A1 register.
R15ignore	deletes L R15,address statements
cregion, cregionoff	update a control variable.

#### 4.4 Comparison of CPS/1100 Statements with Other Concurrent Languages

A group of statements of CPS/1100 is functionally similar to the statements of other concurrent languages. Comparison of the CPS/1100 statements and the statements of other concurrent languages is given below in a tabular form:

<u>Keyword of Statement</u>	<u>Explanations</u>
monitor/process	exist in Concurrent Pascal, Pascal-plus, Simone, Modula and Ada, but in different formats. The format used in CPS/1100 is just like the format of external routine declaration of Pascal 8R1. Functionally these declarations are used to define scope rules as in Concurrent Pascal.
queue	same name is used in Concurrent Pascal for the same type of variables. Condition variables of Simone and Pascal-plus have the same function.

<u>Keyword or Statement</u>	<u>Explanations</u>
forward	optional parameter passing is performed by using different entry points of procedures in Concurrent Pascal, Simone, Pascal-Plus and Ada.
startup	init statement of Concurrent Pascal is functionally equivalent to this keyword of CPS/1100. In most of the concurrent programming languages dispatcher like facilities start up the execution of new processes.
fork	same type of statements exist in CSP/k and in Simone.
wait, signal	exist almost in every concurrent language with different formats.
lock, unlock	functionally same commands used in PL/1. Exist almost in every concurrent language with different formats.
cobegin, coend	exist in CSP/k.
quit	exist in Simone, Pascal-plus and in CSP/k, but conditional quit does not exist.
suspend, terminate	exist in Simone and Pascal-plus.

The other CPS/1100 statements such as deactivate/activate slavetasks, dedicate processor, original mode, snoopy on/off do not exist in other concurrent languages, as these statements were developed by taking advantage of the facilities available under the Exec8 operating system.

## V. CASE STUDIES

### 5.1 Case Study One: Producer-Consumer Problem

#### 5.1.1 Problem Statement [16]

A buffer is shared by two processes so that one of them sends a message to the buffer while the other receives the message from the buffer. Operation of both processes are independent of each other.

Define basic data structures and write CPS/1100 procedures to represent the events described above.

#### 5.1.2 Method of Solution

Let the two processes stated in the problem be called the card reader and printer processes respectively. Operations and the system components can be represented by the following figure.

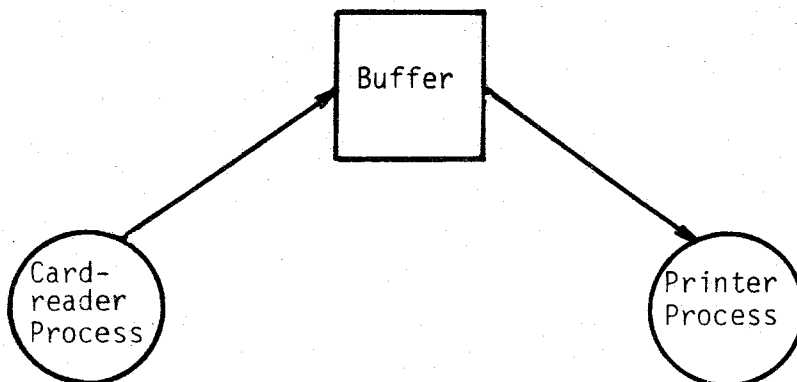


Figure 5.1 - Schematic description of producer-consumer problem.

Card reader and printer processes may be represented by the following data structure in the program implementation.

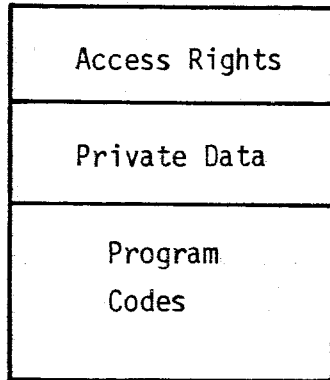


Figure 5.2 - Representation of a Process.

The access rights are conceptual equivalents of the arrows on the Figure 5.1. The private data for card reader and printer processes are the message sent and the message received. Program codes perform the sending and receiving of messages. The following CPS/1100 routine is the implementation of the above data structure for the card reader process:

```

program define (cardproc);
type labels = (receive, send, init);
/c
/c      Access Rights
/c
procedure buffer (var entry: labels; var inp: integer; var out: integer); monitor;
/c
/c      Private Data
/c
procedure cardproc;
var entry    : labels;
    i, inp, out: integer;
/c
/c      Program Codes
/c

```

```

begin
    release (cardproc);
    await (starter);
    i:=1;
    while true do
        begin
            inp:= i*1;
            lock (lockv);
            entry:= send;
            fork with taskname (c1) buffer (entry,inp,out forward);
            await(c1);
            i:=i+1;
        end;
    end;
end;

```

Buffer in the system can be represented by a monitor type data structure. Shared data section of the monitor is a location where the sent and received messages are stored temporarily. The following figure illustrates the structure of a monitor data type:

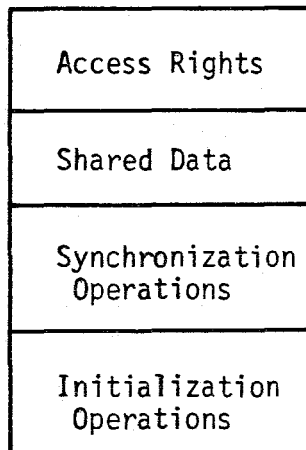


Figure 5.3 - Representation of a monitor.

CPS/1100 routine for the buffer is given below:

```

program define (buffer);
type labels = (receive, send, init);
procedure buffer (var entry: labels; var inp: integer; var out:integer);
/c
/c      Shared Data
/c
var   full    : boolean;
        contents: integer;
/c
/c      Synchronization Operations
/c
begin
    case entry of
        receive:
            begin
                unlock (lockv);
                lock (receiver);
                if not full then wait (receiver);
                    else unlock (receiver);
                out:= contents;
                full:= false;
                lock (sender);
                signal (sender);
                quit;
            end;
        send:
            begin
                unlock (lockv);
                lock (sender);
                if full then wait (sender);
                    else unlock (sender);
                contents:= inp ;
                full    := true ;
                lock (receiver);
            end;
    end;

```

```

        quit;
    end;
/c
/c    Initialization Operations
/c
    init;
        begin;
            full:= false;
            release (buffer);
            wait forever;
        end;
    end;
end;

```

The shared variable `full` defines whether the buffer is full or not. The other shared variable `contents` is used to store the information sent and received by the processes. Queue type variables `sender` and `receiver` are used in synchronization operations.

An activity created in the main routine is transferred to `init` label of the buffer procedure. This activity sets the boolean `full` to false and deactivates itself via a `wait forever` statement. An activity created by the card reader process is transferred to the label `send` of the procedure `buffer`. Control of that activity is suspended if the boolean variable `full` is true. If `full` is false or the activity is woken up via a `signal (sender)` statement, the information sent by the card reader process is stored into the variable `contents`. Boolean `full` is set to true and if there is an activity suspended by `wait (receiver)` statement it is activated by executing a `signal (receiver)` statement. The activity created by the card reader process leaves system. This allows the execution of the card reader process if it is suspended by an `await` statement. An activity created by printer process is transferred to the label `receive` of the procedure `buffer`. Control of that activity is suspended if the boolean variable `full` is false. If `full` is true or the suspended activity is activated via a `signal (receiver)` statement, the information stored in the variable `contents` is transferred to the printer process. The boolean



variable full is set to false and if there is an activity suspended by wait (sender) statement, it is activated via a signal statement. The activity created by the printer process leaves system and allows the execution of the activity suspended by an await statement in the printer process routine.

Operations performed while transferring information to the monitor and transferring the control of the created activity into a specified label in the procedure buffer must be indivisible. Indivisibility is created by critical regions that start before fork statements in the card reader and printer processes and end after receive and send labels in the monitor buffer.

The main routine of the program is not worth to discuss, since it only creates activities for card reader and printer processes. Complete CPS/1100 and corresponding assembler program listings for this case study are given in Appendix E1.

## 5.2 Case Study Two: Dining Philosophers

### 5.2.1 Problem Statement [3]

Five philosophers sit around a table. Each philosopher alternates between thinking and eating. In front of each philosopher there is a plate with spaghetti. When a philosopher wishes to eat, he picks up two forks next to his plate. There are, however, only five forks on the table. So a philosopher can only eat when none at his neighbours are eating.

Assume that all philosophers are thinking in the initial state and spaghetti plates are continually replenished. Define thinking and eating periods of philosophers. Write a deadlock free program to represent events described above.

### 5.2.2 Method of Solution

The following figure is the table described in the problem.

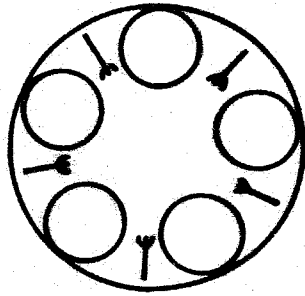


Figure 5.4 - Forks on the table.

Philosophers are simulated by `phils` procedure and the forks which are resources are represented by the `forks` monitor. An activity for each philosopher is created and transferred to the specific labels in the procedure `phils`. Those activities are suspended via `await` statements until the `coend` statement is executed in the main routine. A queue type variable is declared for waiting/signaling operations of each philosopher. Following codes describe the main routine of the program:

```

program main;
/c
/c    Access Rights
/c
procedure forks (<argument declarations>); monitor;
procedure phils (<argument declarations>); process;
/c
/c    Variable Declarations
/c
begin
  startup with taskname (F00) forks (<arguments>);
  startup with taskname (P00) phils (<arguments>);
  cobegin (starter);
/c
/c    Forking to philosopher 0
/c
  pentry := phil0;
  fork with taskname (P0) phils(pentry);

```

```

/c
/c   Forking to philosopher 1
/c
    pentry := phil1 ;
    fork with taskname (P1)phils(pentry);
    -
    -
    -
/c
/c   Forking to philosopher4
/c
    pentry := phil4 ;
    fork with taskname (P4) phils (pentry);
    coend;
    wait forever;
end.

```

Philosophers live in a thinking and eating loop. An activity that simulates the actions of a philosopher is suspended for a certain period of time, i.e., thinking period. At the end of that period the philosopher attacks to get his forks which are the shared resources with his neighbours. Entry to monitor forks is defined in a critical region. Only one philosopher at a time may pick up or put down his forks. An activity is created to simulate each pickup or putdown operation of each philosopher. When a fork is requested by a philosopher, the requesting philosopher is entered into a wait state until one of his neighbour philosophers wakes him up. Process phils is given below:

```

program define (phils);
type philent = (phil 0, phil1 , phil2 , phil3 , phil4 , pinit);
type labels = (pickup, putdown, finit);
procedure phils (var pentry: philent; var i: integer);
/c
/c   Access Rights
/c
procedure forks (var fentry: labels; var i: integer); monitor;

```

```

/c
/c Variable Declarations
/c
begin
  case pentry of
    phil0:
      begin
        await (starter);
        phid0 := 0;
        while true
          do begin
/c
/c           Thinking time is 3 time units
              suspend (3);
/c
/c           Pickup Operation
              lock (entrylock);
              lock (philq0);
              fentry := pickup;
              fork with taskname (F0) forks(fentry,phid0);
              wait (philq0);
              await (F0);
/c
/c           Eating time is 8 time units
              suspend (8);
/c
/c           Putdown Operation
              lock (entrylock);
              fentry := putdown;
              fork with taskname (F0) forks (fentry,phid0);
              await (F0);
          end;
    end;
end;

```

```

phil4:
    begin
        await (starter);
        phid4 := 4;
        while true
            do begin
                -
                -
                -
                -
            end;
        end;
pinit:
    begin
        release (phils);
        wait forever;
    end;
end;

```

When a philosopher attacks to get his forks, he may receive them if both forks are available. The activity that simulates a pickup operation decrements the variables that represent the number of forks available for the neighbouring philosophers. It resumes the control of the activity that simulates the requesting philosopher via a signal statement. The activity then unlocks the monitor entry and leaves system. If both forks are not available when they are requested, an element of the boolean array, that indicates whether the requesting philosopher in wait state or not, is set to true. In this case, the activity representing the pickup operation leaves the system.

Philosophers try to put their forks down via an activity that simulates a putdown operation. This activity increments the variables that represent the number of forks available for neighbouring philosophers. It checks whether neighbouring philosophers are in wait state and both of their forks are available. When a neighbour philosopher is found

as waiting for forks and there are two available forks for him, a pickup operation is applied for that philosopher. After this, the activity that simulates the putdown operation exits the system and the requesting philosopher thinks for a while again.

Listing of monitor forks is given below. Complete listing of CPS/1100 program for case study two can be found in Appendix E2.

```

program define (forks);
type labels = (pickup, putdown, init);
procedure forks (var fentry: labels; var i:integer);
var frk : array [0..4] of 0..2;
      waiting : array [0..4] of boolean;
begin
  case fentry of
    pickup:
      begin
        case i of
          0: lock (philq0);
            -
            -
          4: lock (philq4);
        end;
        if frk [i] <> 2
          then begin waiting [i]:= true;
              case i of
                0: unlock (philq0);
                  -
                  -
                4: unlock (philq1);
              end;
              unlock (entrylock);
              quit;
            end;
          else begin

```

```

        waiting [i]:= false;
        frk[(i+4)mod5]:=frk[(i+4)mod5]-1;
        frk[(i+1)mod5]:=frk[(i+4)mod5]-1;
        case i of
            0: signal (philq0);
                -
                -
            4: signal (philq4);
        end;
        unlock (entrylock);
        quit;
    end;

putdown:
    begin
        frk[(i+4)mod5]:=frk[(i+4) mod5]+1;
        frk[(i+1)mod5]:=frk[(i+1) mod5]+1;
        if (frk[(i+4) mod5] = 2) and
            (waiting [(i+4) mod5])
        then begin
            waiting [(i+4) mod5]:= false;
            frk[(i+3) mod5]:= frk[(i+3) mod5]-1;
            frk[i]:= frk[i]-1;
            case (i+4) mod5 of
                0: signal (philq0);
                    -
                    -
                4: signal (philq1);
            end;
            if (frk[(i+1) mod5]= 2) and
                (waiting [(i+1) mod 5])
            then begin
                Pickup forks for philosopher (i+1) mod 5
                and wake him up
            end;
        quit;
    end;

```

```

    end;
  init:
    begin
      for i:= 0 to 4 do begin
        frk [i] := 2;
        waiting [i] := false;
      end;
      release (forks);
      wait forever;
    end;
  end;
end;

```

### 5.3 Case Study Three: Quicksort with Processor Dedication

#### 5.3.1 Problem Statement

Quicksort [17] is an efficient sorting algorithm. Its performance may be improved by exploiting parallelism. Any sorting algorithm may be modified for a dual processor system in the following way:

1. Divide the array to be sorted into two halves
2. Sort the two halves simultaneously
3. Merge two halves.

The first arrangement operation of the quicksort algorithm is used to divide the array into two subarrays. Those subarrays can be sorted concurrently in two different processors of a dual processor system. Merging operation is not essential for quicksort algorithm.

Modify the quicksort algorithm for a dual processor system and write its program.



### 5.3.2 Modified Quicksort Algorithm

Let  $x$  be the array to be sorted and  $n$  be the number of elements in the array. Choose a pivot from a specific position within that array such as the first element of array. Rearrange the array  $x$  and place the pivot element into a position  $j$  such that, each of the elements in position  $1$  to  $j-1$  is less than or equal to the pivot element and each of the elements in position  $j+1$  to  $n$  is greater than or equal to the pivot element. Sort the subarrays, which corresponds to the elements  $1$  to  $j-1$  and  $j+1$  to  $n$  of array  $x$ , by dedicating a processor to each of them.

### 5.3.3 Program Description

Recursion on the quicksort algorithm must be eliminated by using a stack since CPS/1100 does not permit recursive procedure calls. This stack is used to store pointers for lower and upper bounds during rearrangement operations.

Order of comparisons in sequential quicksort algorithm is  $n \ln_2 n$  on average. It is decreased to  $(n \ln_2 n + (n/2) \ln_2 (n/2))/2$  in the modified quicksort algorithm. Efficiency of the modified algorithm depends on how well the pivot is chosen. But the order of comparison in the modified algorithm on worst case is equal to the average case of sequential algorithm.

In the CPS/1100 program for this case study, three process, divide-two, quick1, quick2 are declared in the main routine. The unsorted array  $x$  is read. Processes are started up by the main activity and the address of array  $x$  and the size of it is transferred to the process dividetwo as arguments. Control of the main activity is suspended until the sorting operation is completed. The main routine is illustrated with comments in the following program outline.

```
program main;
/c
/c   Access Rights
/c
```

```

procedure dividetwo (<argument declarations>); process;
procedure quick1 (<argument declarations>); process;
procedure quick2 (<argument declarations>); process;
/c
/c    Variable Declarations
/c
begin
/c
/c    Read Array x
/c
    startup with taskname (T00) dividetwo (<arguments>);
    startup with taskname (T01) quick1 (<arguments>);
    startup with taskname (T02) quick2 (<arguments>);
/c
/c    Wait until sorting completion
/c
    quit;
end;

```

Dividetwo process chooses the first element of array x as the pivot element. It rearranges the array x and places the pivot into a position j, so that the pivot element divides the array into two subarrays according to the values of the elements. Dividetwo process creates two activities and transfers their control to the quick1 and quick2 routines. It suspends its execution until quick1 and quick2 processes both complete their duties. Two different routines, quick1 and quick2 are used to sort two subarrays, although these routines are the same of each other, since the Pascal 8R1 compiler does not produce reentrant codes. The dividetwo routine is explained with comments in the following program outline.

```

program define (dividetwo);
procedure dividetwo (<argument declarations>);
/c
/c    Access Rights
/c

```

```

procedure quick1 (<argument declarations>); process;
procedure quick2 (<argument declarations>); process;
/c
/c    Variable Declarations
/c
begin
    release (dividetwo);
/c
/c    Choose a pivot element, arrange array x
/c
/c    and divide it into subarrays
/c
    fork with taskname (T00) quick1 (<arguments>);
    fork with taskname (T07) quick2 (<arguments>);
    await (T06, T07);
/c
/c    Print the sorted array x
/c
/c
    quit;
end;

```

Quick1 process dedicates processor 0 while quick2 process dedicates processor 1 for their execution. Quick1 sorts one of the subarrays by successive rearrangements. Quick2 performs the same task on the other subarray. Both of the activities exit the system after they finish sorting. The following statements explain quick2 process:

```

program define (quick2);
procedure quick2 (<argument declarations>);
begin
    case entry of
    q2sort:
        begin
            dedicate processor (1);

```

```

/c      Sort one of the subarrays
/c
        quit;
        end;
q2init:
        begin
            release (quick2);
/c      Initialization operations
/c
        end;
        end;
end;

```

Complete program listing for this case study is given in Appendix E3. Dedicate processor statements are not used in the program listed in Appendix E3.1. Sorting is performed by two processes on a single processor. In Appendix E3.2, only the program segments, where processors are dedicated, and the obtained results are given. In this case, due to the absence of a second processor in the machine that the program was run on, one of the sorting activities is aborted with error type 04 and error code 077. For this reason one subarray was sorted.

#### 5.4 Case Study Four: Gaussian Elimination with Processor Dedication

##### 5.4.1 Problem Statement [18]

Gauss elimination method, which is well known, is used to solve linear equation systems. This method can be improved for dual processor systems and thus execution time may be decreased.

##### 5.4.2 Gaussian Elimination Algorithm for Dual Processor Systems

Operations of the gaussian elimination method may be divided into two parts, the reduction operation on the pivot row and the reduction

operation of the rows that succeed the pivot row. The second operation can be handled by two similar activities within different processors. Sequential gaussian elimination algorithm may be modified in the following manner:

1. Create an activity and dedicate the second processor to it. Suspend that activity.
2. Select the pivot row,  $i$ , starting from first row.
3. Reduce the pivot row so that the diagonal element is equal to 1. If the pivot row is the last row, stop elimination.
4. Wakeup the second activity.

Tasks performed by first activity:

5. Select row  $k$  as  $i+1, i+3, i+5$  and reduce these rows such that the element  $k, i$  is equal to zero.
6. If the last row is reached check whether the second activity has finished its duty. If the second activity has not finish its duty, enter a wait state otherwise jump to step 3 incrementing  $i$  by one.

Tasks performed by second activity:

7. Wait until the first activity sends a signal for waking up.
8. Select row  $l$  as  $i+2, i+4, i+6, \dots$  and reduce these rows such that the element  $l, i$  is equal to zero.
9. If the last row is reached, register that the duty of second activity is complete and send a signal to wake up the first activity.
10. Jump to step 7.

### 5.4.3 Program Description

The main routine reads the input matrix which has  $n$  rows and  $n+1$  columns. Last column of the matrix consists of the right hand sides of the equalities. Main activity initializes the process gauss. Two reduction activities are created. Control of the main activity is suspended until elimination is finished. First reduction activity wakes up the main activity when it finishes its duty. Then, the main activity calculates the unknown variables by using the reduced matrix. Main routine is explained by comments in the following program outline.

```

program main;
const  n = 20; np1 = 21;
type  entries = (entry1, entry2, init);
        arrtype = array [1..n, 1..np1] of real;
procedure gauss (entry: entries; var A: arrtype); process;
var  A      : arrtype;
        i,j,k: integer;
        -
        -

begin
/c
/c  Read the input matrix A
/c
startup with taskname (g0) gauss (init, A forward);
-
-
fork with taskname (g1) gauss (entry1, A);
-
-
fork with taskname (g2) gauss (entry2, A);
-
-
wait(waitwd);

```

```

/c
/c   Calculate the unknown variables
/c
end.

```

There are three entries in the routine gauss: `init`, `entry1` and `entry2`. The activity, which is transferred to entry `init` initializes the boolean variable `permission` as false. Boolean variable `permission` shows whether the second reduction activity is in a wait state or not. The first reduction activity, which is named `g1`, is transferred to label `entry1`. It starts to reduce the pivot row `i` while the second activity is transferred to the label `entry2`. Activity `g1` sets the boolean variable `permission` to true in a critical region. It reduces the rows `i+1`, `i+3`, `i+5`, etc. At this time, the second reduction activity `g2` checks the variable `permission` in a critical region. It either enters a wait state or leaves the critical region depending on the value of the boolean `permission`. If `permission` is true or the second activity, `g2`, is woken up then activity `g2` starts to reduce the rows `i+2`, `i+4`, `i+6` and so on. Activity `g2` sets the boolean `permission` to false and checks the value of pivot row index `i`. If the pivot row is the last row of the matrix, activity `g2` exits the system otherwise it checks boolean `permission` for further reduction operations. When activity `g1` finishes a reduction pass it checks boolean `permission` to see whether activity `g2` has finished its duty or not. If activity `g2` has finished its reduction pass then activity `g1` increments the row index `i` by 1 otherwise enters a wait state. When activity `g1` reduces the last row of the matrix, it exits the system. Following program outline explains the routine gauss.

```

program define (gauss);
const n = 20; npl = 21;
type entries = (entry1, entry2, init);
         arrtype = array [1..n, 1..npl] of real;
procedure gauss (entry: entries; var A: arrtype);
var i,j,k,l,m : integer;
         permission : boolean;
         p,q,r      : real  ;

```

```

begin
  case entry of
    entry1:
      begin
        dedicate processor ( $\emptyset$ );
        for i:= 1 to n
          do begin
/c
/c          Reduce pivot row i
/c
          if i = n then
            begin lock (waitwd);
              signal (waitwd);
              quit;
            end;
            lock (act2);
            permission:= true;
            signal (act2);
            k:= k+1;
            while k < n
              do begin
/c
/c          Reduce row k
/c
              k:= k+2;
              end;
              lock (act1);
              if permission then wait (act1)
                else unlock (act1);
              end;
          end;
        end;
    entry2:
      begin
        dedicate processor (1);
        -
        while true

```



```

do begin
    lock (act2);
    if not permission then wait (act2);
        else unlock (act2);
    l:= i+2;
    while l < n
        do begin
/c
/c          Reduce row l
/c
            l:= l+2;
            end;
            lock (act1);
            permission:= false;
            signal (act1);
            if i > (n-1) then quit;
        end;
init:
    begin
        permission:= true;
        -
        -
        quit;
    end;
end;
end;

```

The program listing can be found in Appendix E4.1. Dedicate processor statements are not used on the listed program. Equations are solved in a single processor system and the results are printed. In Appendix E4.2, only the program segments, where processors are dedicated to distinct reduction activities, are given. In that case program is aborted with error type 04 and error code 077 which corresponds to the error message processor requested on `aded$` call was unavailable at the time of the `er aded$`, or was downed by the system operator.

## 5.5 Case Study Five: A Model Operating System

### 5.5.1 Problem Statement

Assume that a timesharing and multiprogramming operating system can manage only pascal programs. Memory requirements, priorities and start-times of the user jobs are given on the job control cards as parameters. The system has the following properties:

1. A line frequency clock generates regular clock interrupts in every predefined time interval.
2. User programs may be time shared according to different scheduling strategies.
3. The system maintains three separate queues of user programs.

One is the ready queue of all programs in memory that are capable of immediately utilizing the CPU. The second queue is the blocked queue of all programs that can not compete for the CPU because they are on secondary storage. All members of blocked queue can proceed directly to the ready queue when loaded into main memory. The third queue is the i/o queue of all programs that are waiting for i/o processing.

4. Once every predefined time duration, if the blocked queue is not empty, the oldest job in the ready queue is swapped out of main memory. A job in the blocked queue that is selected according to a priority rule is loaded into the ready queue.

Write a CPS/1100 program for this model operating system.

### 5.5.2 Model

CPU and i/o unit are the servers and ready, blocked and i/o queues are the three queues in the system. This model is schematically described by the following figure.

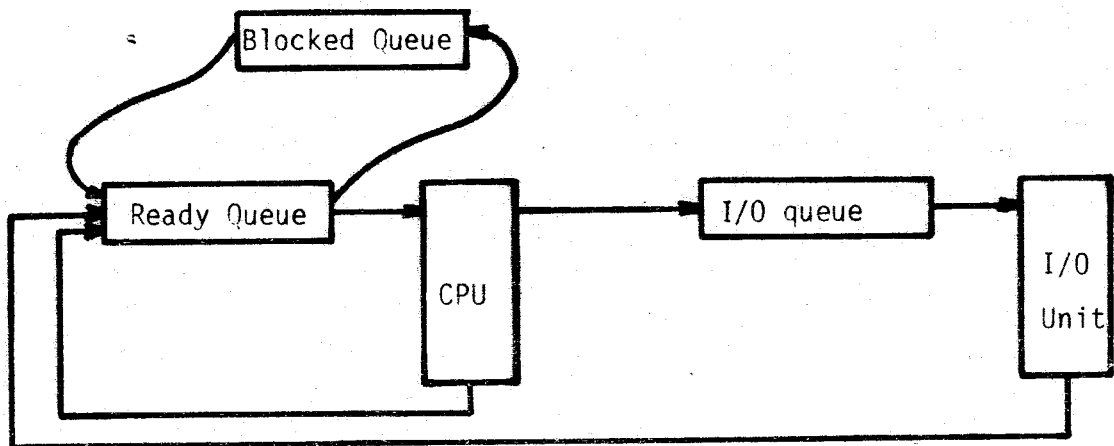


Figure 5.4 - Representation of the model operating system.

Actions in the system are listed below:

1. A job processing in CPU is interrupted and is moved to the end of ready queue at every regular clock interrupt. At these intervals, the ready queue is rearranged according to a specific scheduling rule. First job in the ready queue is moved to the CPU for processing.
2. A job deactivates itself while informing the operating system that it has started an i/o operation. At this instant, the requesting job is transferred to the i/o unit queue. The first job in the ready queue is fetched and it is moved to the CPU for processing. The i/o queue is handled on a first come first served basis. The first job in the i/o queue gets the i/o unit first.
3. A job reactivates itself while informing the operating system that it has finished its i/o operation. This job is transferred to the end of ready queue. Ready queue is rearranged according to a specified scheduling rule. If i/o processing queue is not empty, the first job in the i/o queue is transferred to the i/o unit.
4. Oldest job in the ready queue is swapped to the end of the blocked queue at every predefined time period. The blocked

queue is rearranged according to a scheduling rule. A number of jobs are transferred to the end of the ready queue so that unused memory in the system is reduced to a minimum. The ready queue is also rearranged during this operation.

### 5.5.3 Program Description

Processes arrange and model operating system (mos) are declared in the main CPS/1100 routine. Slavetasks, which simulate user jobs in the model, are started up. Processes arrange and mos are also initiated. An activity, that simulates the system clock, is created and transferred to the clockstart label of the mos routine. Another activity that simulates regular clock interrupts, is created and directed to the regularint label of the routine mos. The activity that simulates the actions performed when an i/o operation is started in a user job is created and directed to the label ioinitiate. The activity that simulates actions when an i/o operation is completed in a user job is created and transferred to the label ioterminate in the mos routine. Outline of the main routine is given below:

```

program main;
/c
/c      Type declarations
/c
procedure mos (<argument declarations>); process;
procedure arrange (<argument declarations>); process;
/c
/c      Variable declarations
/c
begin
    i:=1; while i < <number-of-slavetasks>
        do begin
            startup slavetask (i);
            i:= i+1;
        end;

```

```

startup with taskname (A00) arrange (<arguments>);
startup with taskname (T00) mos (<arguments>);
-
-
mosentry:= clockstart;
fork with taskname (t01) mos (mosentry);
-
-
mosentry:= regularint;
fork with taskname (t02) mos (mosentry);
-
-
mosentry:= ioinitiate;
fork with taskname (t03) mos (mosentry);
-
-
mosentry:= ioterminate;
fork with taskname (t04) mos (mosentry);
-
-
end.

```

The program segment under the label clockstart simulates the system clock. Execution of this segment is in an endless loop in that segment. Control of the clock activity, that represents the time slice used in the model, is suspended for a while. The clock is then updated and a signal operation on the queue variable clockintrpt is performed. This program segment is listed below:

```

clockstart;
  begin
    -
    -
    while true do begin

```

```

suspend (50);
lock (clocklock);
clock: = clock + 50;
unlock (clocklock);
if clock >= 30 000 then terminate;
lock (clockintrpt);
signal (clockintrpt);
end;

```

Actions performed after each regular clock interrupt are programmed in the regularint segment of the mos routine. An activity in that segment waits for a clock interrupt. It is woken up when a signal operation is performed on the queue type variable clockintrpt. This activity checks whether the CPU is idle and the ready queue is empty. If both conditions occur, then the activity enters a wait state by using the queue type variable emptyrq. Otherwise it may start a swapping or a CPU timesharing operation in the system.

The activity in the regularint segment checks the following conditions for swap operation:

1. Blocked queue is not empty
2. Swappable memory space, which is the portion of memory used by the jobs waiting for CPU, is equal to or greater than the memory requirement of the first job of the blocked queue.
3. The predefined time interval between two swap operation is consumed.

The activity, whether it performs a swap operation or not, checks the time for a CPU sharing operation. If the difference between clock value and the time, at which last CPU switching is performed, is equal to or greater than the predefined time interval for timesharing then a CPU switching operation is performed. Afterwards, the activity being in an endless loop enters a wait state until a signal operation is performed on the queue type variable clockintrpt. The following program outline shows the actions performed after each regular clock interrupt:

regularint:

```

-
-
  while true
    do begin
      lock (clockintrpt);
      wait (clockintrpt);
      if (rqtail = rqhead) and
        (controltable.ccu = 0)
        then begin
          lock (emptyrq);
          wait (emptyrq);
        end;
      lock (masterlock);
      lock (clocklock);
      if (bqhead<>bqtail) and
        (memreq[bqlist[bqhead]] <= memswappable) and
        ((clock - lastswap) >= 500)
        then begin
          lastswap:= clock;
/c
/c
          Swap operation
        end;
      if ((clock - timedecrement) >= 50) or (timedecrement=0)
        then begin
          timedecrement:= clock;
/c
/c
          CPU switching operation
        end;
    end;

```

CPU switching is performed in the following manner:

If any job is already occupying the CPU that job gets deactivated. Then, the status of all user jobs are checked in order to determine

whether the current CPU user already exists in the system. The deactivated job is placed at the end of the ready queue. The current clock value is registered for the purpose of searching the oldest job in the ready queue. Jobs in the ready queue are rearranged by creating an activity and forking it to the arrange routine. After the arrangement operation, the variable that represents swappable memory space is updated and the CPU is registered as idle. The first job in the memory queue is transferred to CPU. Pointers of ready queue and the variable that represents swappable memory space are updated. The new job in CPU is activated. The program segment that performs CPU switching is given below:

```

if (rqtail<>rqhead)
  then begin
    if (ccu<>0) and ((quantum [ccu] mod quantmax) = 0)
      then begin
        deactivate slavetask (ccu);
        fetch status of slavetasks;
        if status [ccu]
          then begin
            rqlist [rqtail]:= ccu;
            rqwait [rqlist[rqtail]]:= clock;
            rqtail:= (rqtail+1) mod 20;
            lock (arrangelock);
            arrangentry:= arrglist;
            lock (arrflag);
            fork with taskname (A02)
              arrange (arrangentry,rqlist,rqhead,
                    rqtail,quantum);
            wait (arrflag);
            await (A02);
            unlock (arrangelock);
          end;
        memswappable:= memswappable+memreq[ccu];
        ccu:= 0;

```



```

        if rqtail = rqhead then cpuidle := clock;
    end;
    end;
    if (rqtail<>rqhead) and (ccu = 0)
        then begin
            ccu:= rqlist [rqhead];
            rqlist [rqhead]:= 0;
            rqhead:= (rqhead+1) mod 20;
            memswappable:= memswappable - memreq [ccu];
            activate slavetask (ccu);
        end;

```

Swapping operation is performed in the following manner:

If ready queue is not empty the oldest job in the queue is removed to the end of the blocked queue. Variables corresponding to unused memory and swappable memory space are updated. The blocked queue is rearranged. Ready queue is reorganized by shifting some elements to successive positions to recover the gap created by the oldest job. These operations are repeated until the memory requirement of the first job of the blocked queue is satisfied. Program outline for this segment is given below:

```

    while (rqtail<>rqhead) and
        (memreq [bqlist [bqhead]]>= memunused)
        do begin
/c
/c      Find the oldest job j
/c
        memunused:= memunused + memreq [rqlist[j]];
        memswappable:= memswappable - memreq[rqlist[j]];
        bqlist[bqtail]:= rqlist[j];
        bqtail:= (bqtail + 1) mod 20;
/c
/c      Rearrange blocked queue and ready queue
/c
    end;

```

If unused space in memory is equal to or greater than the memory requirement of the first job in the blocked queue, this job is appended to the end of the ready queue. The current clock value is registered as the time of joining to ready queue. Ready queue tail pointer is updated and the ready queue is rearranged. Variables denoting the unused memory and the swappable portion of memory are also updated. Block queue head pointer is changed. Those operations are repeated successively until either the blocked queue is empty or memory request of the first job in the blocked queue is greater than the unused space in memory. Following program outline describes these actions:

```

while (memreq[bqlist[bqhead]] <= memunused) and
      (bqhead <> bqtail)
  do begin
    rqlist[rqtail] := bqlist[bqhead];
    rqwait[rqlist[rqtail]] := clock;
    rqtail := (rqtail + 1) mod 20;
  /c
  /c   Arrange ready queue
  /c
    memunused := memunused - memreq[bqlist[bqhead]];
    memswappable := memswappable + memreq[bqlist[bqhead]];
    bqlist[bqhead] := 0;
    bqhead := (bqhead + 1) mod 20;
  end;

```

When a user job begins an i/o operation, it performs a signal operation on the queue type variable that is defined as i/o initiation flag in the main CPS/1100 routine. The user job then deactivates itself. At this moment, the activity that is already suspended in ioinitiate segment of mos routine is woken up by a signal operation. This activity places the user job at the end of the i/o queue. It transfers the first job in ready queue to CPU for processing and updates the head pointer of the ready queue. If the ready queue is empty, the CPU is registered as idle. Meanwhile, if the i/o unit is empty,

the first job in the i/o queue is placed into the i/o unit and head pointer of i/o queue is updated. The user job in the i/o unit is activated for i/o operation. The following program segment describes the actions outlined above.

ioinitiate:

```

-
-
  while true
    do begin
      wait (ioinit);
      lock (ioinit);
      -
      -
      dioqlist [dioqtail]:= ccu;
      dioqtail:= (dioqtail+1) mod 20;
      if rqhead<>rqtail
        then begin
          ccu:= rqlist [rqhead];
          rqlist[rqhead]:= 0;
          rqhead:= (rqhead+1) mod 20;
          activate slavetask (ccu);
        end
      else begin
        ccu:= 0;
        cpuidle:= clock;
        end;
      if cioj = 0 then
        begin
          cioj:= dioqlist [dioqhead];
          dioqlist [dioqhead]:= 0;
          dioqhead:= (dioqhead+1) mod 20;
          activate slavetask (cioj);
        end;
      end;
    end;

```

When a user job in the i/o unit completes its i/o operation, it performs a signal operation on the queue type variable that is defined as i/o termination flag in the main CPS/1100 routine. This user job deactivates itself. Consequently, the activity that is woken up by a signal operation inserts the user job at the end of the ready queue. It rearranges the ready queue and updates the variable that keeps track of the swappable memory space. Then, a signal operation is performed on the queue type variable emptyrq to state that the ready queue is not empty. If i/o queue is empty, the i/o unit is registered as idle, otherwise the first job in i/o queue is removed to the i/o unit and the head pointer of i/o queue is updated. The job in possession of the i/o unit is activated. Program outline for the described actions is given below:

```

ioterminate:
-
-
while true do
  begin
    wait (ioterm);
    lock (ioterm);
-
-
    rqlist[rqtail]:= cioj;
    rwait[rqlist[rqtail]]:= clock;
    rqtail:= (rqtail+1) mod 20;
  /c
  /c Arrange ready queue
  /c
-
-
    lock (emptyrq);
    signal (emptyrq);
-
-
    memswappable:= memswappable + memreq[cioj];

```

```

    if dioqhead = dioqtail then cioj = 0
        else begin
/c
/c           Remove the job into i/o unit
/c           and activate it. Update i/o queue
/c           head pointer
/c
/c
        end;
-
-
end;

```

The activity which is created and transferred to the label mosinit of mos routine, initializes local variables and arrays of the routine mos. It reads memory requirements, priorities and start-times of user jobs from a file named infotable. This information is originally extracted from CPS/1100 control directives of the user jobs. Characteristic parameters of these jobs are registered in the arrays starttime, memreq and priority. Jobs are arranged according to their start-times, and their priorities if required. A group of jobs are transferred into the ready queue until the memory is full. The remaining jobs are filled into the blocked queue. The following program outline describes actions on initialization of routine mos.

```

mosinit:
    begin
/c
/c           Variables and arrays are initialized
/c
    reset (infotable);
    readln (infotable);
    for i:= 1 to <number of slavetasks>
        do begin
            read (infotable,memreq[i],priority[i],starttime[i]);
        end;

```

```

/c
/c   Sort jobs according to their starttimes and sort them
/c
/c   according to their priorities, if required.
/c
/c   Fill ready and then blocked queue
/c
-
-
  end;

```

User jobs in this model study are single level Pascal programs. Characteristics such as size, priority of a user job are given on its control directive. A sample user program with its control card is given below:

```

*compile options=isrlmv,eltname=tpf$.user3,user=uc,priority=5,starttime=0000
  program main;
  var a,b,j: integer; c:real;
  begin
    b:=3;
    while true
      do begin
        for j:=1 to 999999
          do begin
            a:=b; c:= 5.23*18.2*0.03/(3.8*8.8*0.44);
            if (j mod 444) = 0
              then begin
                writeln (a,c);
              end;
            b:= a;
          end;
        end;
      end;
    end;
  end;

```

The complete program listing for this case study is given in Appendix E5. Minimum response time strategy is applied for CPU switching in the model. Quantum size is selected as five. In this case study 18 user jobs were managed by the model operating system. Diagnosis for switching and swapping operations and the queue dumps are given on the output.

## VI. DISCUSSION

In the present work, a concurrent language has been designed, implemented and tested. Its main virtues are:

- Flexibility
- Processor dedication capability
- Program trace capability
- Easy usage.

More details are available in a section of Chapter two, "Description of CPS/1100".

The present chapter is mainly devoted to discuss the use of CPS/1100, extensibility and portability of CPS/1100, comparison of CPS/1100 with other concurrent languages, and recommendations for future research.

### 6.1 The Use of CPS/1100

CPS/1100 is a powerful language with distinguishing characteristics from the other concurrent languages. These characteristics make CPS/1100 a favourable language in operating system modelling and concurrent programming studies for average programmers. The attractive features of the language are stated below:

#### A. Flexibility of the Language:

CPS/1100 is modifiable and extensible because of its special structure. The structure of CPS/1100 permits any modification on the syntax and the semantics of the language. Therefore, capabilities of



the language can be improved by modifying scanner routines of CPS/1100. Thus, new languages could be developed for specific applications. An average programmer may easily modify the language, because a high level string manipulation language has been chosen for implementing the scanner routines.

#### B. Processor Dedication Capability:

This capability makes CPS/1100 a powerful language, since real parallelism on program control flow can be obtained with this feature. Thus, programmers can make a trade-off between space, time and processors while they analyze their problems.

#### C. Program Trace Capability:

Program tracing capability of CPS/1100 is very useful for concurrent program debugging, since errors in concurrent programs are not reproducible and may occur time to time. Errors in CPS/1100 programs can be fixed by using the program trace capability.

#### D. Easy Usage of the Language:

CPS/1100 syntax is similar to the syntax of the Pascal language. Therefore, any programmer who knows Pascal programming can write programs in CPS/1100.

These distinguishing features of CPS/1100 are discussed, by comparing it with other concurrent languages, in the succeeding sections of this chapter.

### 6.2 Extensibility and Portability of CPS/1100

CPS/1100 uses EXEC8 facilities to create, remove and control activities. Those features of EXEC8 that are not normally available to users have not been employed. However, it is possible to extend CPS/1100 by making privileged EXEC8 instructions available to it in a straightforward manner by simply adding new prescanner and postscanner routines.

Fortran routines can not be called in the present form of CPS/1100. This restriction stems from restrictions in the Pascal 8R1 library, and may be overcome by simply modifying this library. Such a change will make CPS/1100 available for a wide range of numerical applications.

It is possible to extend CPS/1100 and make it a powerful realtime language by adding new statements and console communication features into it.

The lack of portability in CPS/1100 arises from the fact that Macro and Pascal 8R1 are software written specifically for Univac 1100 series. This characteristic is quite natural for any piece of software which heavily interacts with the operating system.

### 6.3 Comparison of CPS/1100 with Other Concurrent Languages

The design philosophy of CPS/1100 is different from that in Distributed Processes, Communicating Sequential Processes, Modula, Pascal-plus and Ada. The concurrency and synchronization constructs of CPS/1100 are high level statements having low level correspondents in EXEC8. The objective of CPS/1100 study was to raise and embed the multitasking features of EXEC8 into a high level programming system. However, the design of other concurrent languages were dictated by the programming needs. Extensive studies were performed by software designers to satisfy the needs of professional software developing studies for embedded computer systems, in the Ada project. The design philosophy of Modula was to build up a language that has powerful data abstraction features and that runs without support of an operating system. The objective of Distributed Processes and Communicating Sequential Processes was to define programming languages for distributed memory systems. Therefore, the essential difference between CPS/1100 and the other concurrent software is that, CPS/1100 has concurrency features which are raised from EXEC8, the supporting software, while the constructs of other languages were designed according to needs.

Extensibility of CPS/1100 is its most powerful aspect compared to the other software such as Modula, Pascal-plus, Ada and Concurrent

Pascal. Any modification or improvement on these concurrent languages necessitates extensive work on both design and implementation phases. But, extensions on CPS/1100 are rather simple. CPS/1100 is a software that can be reshaped rather easily.

Automatic program debugging and tracing is a bottleneck for concurrent programming software. Tracing facilities of EXEC8 are raised and embedded into CPS/1100. This is the advantage of CPS/1100 when compared with the other languages such as Modula and Concurrent Pascal.

CPS/1100 and Modula are two extreme cases due to their structures among the concurrent languages. CPS/1100 is fully supported by an abstract machine which is the combination of Univac 1100 hardware and its operating system EXEC8. In contrast, Modula runs on a bare machine.

Externally compilation of CPS/1100 routines is achieved by CPS/1100 control directives. Externally compilation aspect of CPS/1100 nearly resembles the modular structures of other languages such as Ada's packages and Modula's modules.

The common characteristic of CPS/1100, Concurrent Pascal, Modula and Distributed Processes is that all these languages have a static memory allocation mechanism. Dynamic memory allocation of concurrent objects in those languages is ignored because of the effect of memory reorganizations on the overall efficiency of the programs.

The similarity between CPS/1100 and Concurrent Pascal is consistence of access right declarations. In both software, a process can not call another process or monitor without declaring it in a particular section of the program.

CPS/1100 and Plus, both being Univac software for system programming, have similar data structures. Concurrency features of EXEC8 are accessible in both software. But, users are responsible from almost every thing about creation and controlling of activities in Plus programming. Some parts of Plus programs must be coded in Univac 1100 assembler to work on multitasking mode. For that reason, Plus users must know Univac 1100 assembler programming. In contrast, the way of creating and controlling activities in CPS/1100 is well defined. An average programmer who knows Pascal programming may use CPS/1100.

As a result, CPS/1100 has a distinguishing structure and it fills a gap in the available Univac 1100 software.

#### 6.4 Recommendations for Future Research

The following extensions on CPS/1100 are recommended:

1. Accessing the realtime facilities of EXEC8:

CPS/1100 may be furnished with realtime facilities, such as `rt$` and `nrt$` requests, to execute activities in realtime mode. In realtime mode, activities respond externally generated input stimuli within a finite and speciable delay. Thus CPS/1100 can be applied to the programming of engineering control problems. For example robotics or industrial process units may be connected to a Univac 1100 system and can be controlled by CPS/1100 programs.

2. Accessing EXEC8 private data area and routines:

CPS/1100 may be extended so that it becomes a system programming language. It can make use of the console communication and i/o handling requests of Exec8, with simple modifications. It may access all hardware facilities of the machine and private area of Exec8 in the memory if protection on the executive system is suppressed. Protection on Exec8 area can be removed by a simple system generation operation. Then, new statements which correspond to the privilege set of Univac 1100 assembler instructions may be inserted to CPS/1100. Also, new statements to manipulate Exec8 internal tables or routines can be developed.

3. Allowing dynamic storage allocation to CPS/1100 routines:

Although, implementation of dynamic storage allocation mechanism in CPS/1100 is not so easy, such an extension permits the building up of large scale operating system models via CPS/1100.

4. Creating a CPS/1100 library:

Such a library may consist of a set of Pascal and assembler routines which correspond to basic data structures such as queue lists and stacks. Graphic processing and plotting routines may be added to the library. This library would be helpful for better model implementations, program abstraction and coding.

5. Accessing Fortran program libraries:

This feature can be added by modifying the Fortran interface routine in the Pascal 8R1 library. As a result of this extension, a wide range of numerical and scientific application can be programmed in CPS/1100 with spending less effort.

6. Reentrant code creation:

Pascal 8R1 multipass compiler does not create reentrant program codes. If new releases of this compiler produce reentrant codes, then it may be possible to execute more than one activity on the same program codes. This will provide a powerful feature to CPS/1100 whereby a single routine may be used for a set of processes or monitors. As an example, a single routine may be used to simulate a set of user jobs in a model study.

7. Having generic units:

Data types and routines in CPS/1100 programs may be parameterized by producing their copies, if CPS/1100 has generic producing capability. That capability may be obtained by redesigning and recoding the prescanner routines. Such an improvement would increase the abstraction power of CPS/1100.

Compile time checking is the best way of preventing time dependent errors in multiactivity programs. Therefore, CPS/1100 may be furnished via compile time check routines. Algorithms to detect possible deadlocks in the program may be implemented in those routines.

The present form of CPS/1100 uses a common memory for interprocess communication. New keywords may be added to CPS/1100 to provide process communication with only i/o operations. With this kind of extensions, CPS/1100 would be suitable for programming in distributed computer organizations.

## VII. CONCLUSION

In this work, design and implementation details of a concurrent programming language, which is structurally distinguished from the other concurrent languages, has been described. The merits of CPS/1100 were demonstrated by programming five typical concurrent programming problems with this language.

In case study one, a monitor type datastructure was implemented by using the wait/signal and lock/unlock commands of CPS/1100. The wait/signal pair is used for process synchronization while the lock/unlock commands were used to provide mutually exclusive access to the monitor.

In case study two, a resource allocation problem was solved. The essential difference between concurrent and sequential programming techniques, that is the programmer's ability of thinking in a parallel manner, was demonstrated by this case study.

In case study three, it was showed that some sequential algorithms can easily be adopted for concurrent programming. It was proved by this case study that elegant algorithms can be written by modifying already existing sort, search algorithms and by gaining the control on processors in multiprocessor environments. This case study was an example for using the processor dedication command of CPS/1100.

In case study four, it was illustrated that the processor dedication facility can be used to develop efficient and improved algorithms in widely used mathematical and engineering applications, especially in matrix operations.

In case study five, the suitability of CPS/1100 as a teaching aid and its modelling aspects were displayed. A model of a timesharing-multiprogramming operating system was implemented. Real Pascal programs

were used as user programs in the model. In this case study, it was emphasized that a hierarchical programming system can be implemented by using slavetasks and model tasks of CPS/1100. Activate slavetask deactivate slavetask and fetch status commands were demonstrated in the model study. The contingency registration and handling scheme was used to synchronize slavetasks and model tasks. On the other hand, wait/signal, lock/unlock commands were effectively used to synchronize processes. The case study was a non-trivial concurrent programming example.

In this dissertation, it has been suggested and demonstrated that the prescanning/compilation/postscanning structure is an elegant and convenient way of developing flexible concurrent software. In order to produce an easily modifiable concurrent language, a high level language was chosen for implementing the scanner. Set, token and picture constructs of Macro language were extremely suitable for scanner implementation. Modifications of CPS/1100 syntax and code generation, which were essential during the implementation phase, were performed easily by adding new routines or by updating existing routines of the scanner programs. As an example, the fetch status statement was not considered during the design phase, but it was easily added to the CPS/1100 command repertoire. Also, address assignment capability of the suspend command was easily implemented by simply adding a new macro routine into the prescanner program.

It is quite apparent that the processor dedication facility of CPS/1100, which can be considered as the most distinguishing characteristic from other concurrent languages, provides better control and physical parallelism on program execution flow. Thus, efficient algorithms were developed in the third and fourth case studies by using processor dedication command.

Trace capabilities of CPS/1100 was used to debug programs step by step. The snapdump command of CPS/1100 was used to get dumps of common data area where process synchronization variables were stored. The benefits of debug facilities were experienced during program developments for case studies.



In summary, CPS/1100 was designed and implemented as a concurrent programming language with a flexible structure. The language was enriched with processor dedication and program trace capabilities. It has been proved by the case studies that the design aims were achieved in the implementation of CPS/1100.

APPENDICES

## APPENDIX A. MACRO PROGRAMMING LANGUAGE

Macro is a language which provides powerful and structured text transformation facilities suitable for extending and translating programming languages under Univac 1100 executive system. Macro language may be used in any of the following areas:

1. Language extension, e.g. preprocessor for PL/I
2. Language translation, e.g. Algol to PL/I translation
3. Text generation, e.g. text searching and report generation
4. Text editing, e.g. systematic editing of texts
5. Data validation, e.g. format as value range checks.

Macro system consists two separate program modules. A compilation module, which converts Macro programs into an internal format, and a processor module, which performs actual text transformation. Macro program can be compiled and used without the need for a collection.

A program, known as macro, of the Macro language has two logical parts, a picture and a body. Picture is a specification of the form of the string that the macro will recognize. The body is a series of statements that are executed to generate an output text from input text.

Control cards for Macro compilation and processing are given below: Further information about Macro language can be found on Macro Reference Manual [14].

```
@UNIVAC*MACRO.MDC      file.. source, file.omn
@UNIVAC*MACRO.MDP      file. text, file.omn, file.out
```

where file. source contains Macro program  
file. omn contains omnibus element after compilation  
file. text contains input text  
file. out contains output text produced after processing.

## APPENDIX B. CPS/1100 CONTROL CARDS

Control cards for a sample CPS/1100 is given below. Blanks between fields are allowed. Each control card must be given in one line. Comment lines are identified by /c character in columns 1 and 2.

```
@CPS.S,S      ,CPS.S/0,
*COMPILE OPTIONS=.ISRLM, ELTNAME=TPF$.xxx
```

Main routine

```
*COMPILE OPTIONS= ISRLMV, ELTNAME= TPF$.yyy
```

Process or monitor routine

```
*COMPILE OPTIONS= ISRLMV,ELTNAME=TPF$.zzz, USER=uuu, SIZE=nnn,
    PRIORITY=k, STARTTIME=hhmm
```

Slavetask, if any

```
@CPS.S,N      SCRATCH$.SCRATCH1, CPS.S/1, SCRATCH$.SCRATCH2
@cPS.S,N      SCRATCH$.SCRATCH2, CPS.S/2,
@cPS.X
```

A CPS/1100 program may consists of a main and several process, monitor and slavetask routines. Options of Pascal 8R1 calls are valid on \*COMPILE directives of CPS/1100. V option is mandatory for all routines except the main routine. Element names stated as xxx, yyy, zzz may be a string that is maximum 12 characters in length. User name uuu is a string maximum 3 alphanumeric characters in length.

nnn in SIZE field on slavetask \*COMPILE directive must be an acceptable integer number. In PRIRORITY field must be a single digit integer number and hhmm in STARTTIME field must be a four digit number.

First @CPS.S card is used to call first phase of prescanner, CPS.S/0. The other two @CPS.S cards are used to call the other two phase of prescanner which are CPS.S/1 and CPS.S/2. @CPS.X card maintains Pascal compilation, postscanning, linking and execution of the program.

## APPENDIX C. CPS/1100 STATEMENTS AND KEYWORDS

<u>Statement or keyword</u>	<u>Function</u>
monitor and process	Used for scope rule definitions
queue	type declaration; used to declare process synchronization variables
forward	used for optional parameter transfer
ioinitiation lock, iotermination lock	used to create a critical region around i/o statements in slavetask routines
ioinitiation flag, iotermination flag	used to provide synchronization between model and slave task when an i/o operation in a slavetask is initiated and completed
startup with taskname	creates a new activity with an activity-id associated with the taskname given, allocates the activation record of the procedure to which the control of new activity is transferred, suspends control of the parent activity until the new activity executes a release statement
startup slavetask	creates a new activity with an activity-name associated with the slavetask, allocates the activation record of the slavetask procedure
fork with taskname	creates a new activity with an activity-id associated with the taskname given, transfers the control of this activity to a procedure which is already called by a startup statement
release	resumes the control of the parent activity of the requesting activity that the parent activity has entered in a wait state by performing a startup statement

<u>Statement or keyword</u>	<u>Function</u>
await	suspends the control of an activity while the specified activities in the statement exist in the system
wait	suspends the control of an activity by using the test-and-set and queuing mechanism
signal	resumes the control of an activity, which is already in wait state, by using the test-and-set and queuing mechanism
lock	sets a specific bit of a test-and-set cell if the bit is not set or suspends the control of an activity for a specific time interval if the bit is set
unlock	clears the specific bit of a test-and-set cell if the bit is set
cobegin	creates a specific activity and suspends its control
coend	removes the activity which is created by a cobegin statement
quit	removes an activity from the system
quit by checking	conditionally quit operation
suspend	suspends the control of an activity for a defined time period
clear	stores zero to a queue type variable
wait forever	suspends the control of an activity for an undefined time period
deactivate slavetask	suspends the control of a slavetask
activate slavetask	resumes the control of a slavetask which has been suspended by a deactivate slavetask statement
terminate	ceases the execution of whole activities



<u>Statement or keywords</u>	<u>Function</u>
fetch status of slavetasks	checks whether a slavetask has exited from the system or not. Status of slavetasks are stored into a specific boolean array
dedicate processor	dedicates a specific processor to an activity
original mode	returns back the system into normal processing mode
cregion	executes a ts instruction on a pre-defined test-and-set cell
cend	clears the test-and-set cell which is set by a cregion statement
snoopy on	calls the snoopy trace utility
snoopy off	turns off the trace operation
snapdump	dumps the contents of the queue type variables.

## APPENDIX D - LISTINGS OF MACRO PROGRAMS

- D.1 - Prescanner Phase0
- D.2 - Prescanner Phase1
- D.3 - Prescanner Phase2
- D.4 - Postscanner

## D.1 - PRESCANNER PHASE0

```

@CPS=C,SR CPS,MACRO0,CPS=S/O
MACRO COMPILER LEVEL 7R1 07/27/83 06:06:27
100 1
110 2
120 3 // ***** //
130 4 // * //
140 5 // * //
150 6 // * CPS/1100 PRESCANNER * //
160 7 // * //
170 8 // * PHASE 0 * //
180 9 // * //
190 10 // * //
200 11 // * CPS/1100 STATEMENTS ARE SYNTACTICALLY * //
210 12 // * ANALYSED IN THIS PHASE INPUT STREAM OF THIS * //
220 13 // * PHASE IS A CPS/1100 PROGRAM. OUTPUT STREAM * //
230 14 // * IS PLACED INTO THE ELEMENT 'SCRATCH1' OF A * //
240 15 // * TEMPORARY FILE 'SCRATCHS'. * //
250 16 // * //
260 17 // * //
270 18 // ***** //
280 19
290 20
300 21
310 22 DEBUG ON;
320 23 SET ALPHA <'A' 'Z' >;
330 24 SET ID_CHAR <'A' 'Z' '0' '9' '_' '\.' '\$' '\#' '\%' '\'/>;
340 25 SET DIGIT <'0' '9'>;
350 26
360 27
370 28
380 29 TOKEN OP<'(>;
390 30 TOKEN CP<'>;
400 31 TOKEN MCPS<'*COMPILE'>;
410 32 TOKEN ID < ALPHA(ID_CHAR) >;
420 33 TOKEN BLANK IGNORE <' ' >;
430 34 TOKEN NUM<DIGIT(DIGIT) >;
440 35 TOKEN FOTX<'EOF'>;
450 36 TOKEN COM<'>;
460 37 TOKEN SZ<'SIZE='>;
470 38 TOKEN OPT<'OPTIONS='>;
480 39 TOKEN PR<'PRIORITY='>;
490 40 TOKEN ELTN<'ELTNAME='>;
500 41 TOKEN STT<'STARTTIME='>;
510 42 TOKEN COMMENT<'/'>;
520 43
530 44
540 45 A=S'NUL';
550 46 I=0;
560 47 TOK=S'NUL';
570 48 DFCLAPE R(!);
580 49 ERR=' ** ERROR ** SYNTAX ERROR ';
```

590		50	PROCEDURE READLN;
600		51	ST: I=0;
610	1	52	EAD A;
620	1	53	IF LENGTH(A) EQ 0
630	1	54	TH N BEGIN
640	1	55	CLOSE WRITE INTO 'SCRATCHS',SCRATCH1';
650	2	56	STOP;
660	2	57	END;
670	1	58	IF AC1,23 EQC '/C' THEN GO TO ST;
680	1	59	TOKENIZE A INTO R;
690	1	60	END;
700		61	
710		62	
720		63	PROCEDURE NEXTTOKEN;
730	1	64	ST: I = I+1;
740	1	65	IF I GT SIZE(R)
750	1	66	THEN BEGIN
760	1	67	WRITE A INTO 'SCRATCHS',SCRATCH1';
770	2	68	ALL REA LN;
780	2	69	I=1;
790	2	70	END;
800	1	71	IF TRIM(R(I)) EQC ' ' THEN GO TO ST;
810	1	72	TOK = TRIM(R(I));
820	1	73	END;
830		74	
840		75	
850		76	
860		77	PROCEDURE PROCCPS;
870	1	78	IF TOK EQC '*COMPILE*' THEN
880	1	79	BEGIN
890	1	80	CALL NEXTTOKEN; T=TOK; CALL NEXTTOKEN; T = T&TOK;
900	2	81	IF T EQC 'OPTIONS=' THEN
910	2	82	BEGIN
920	2	83	CALL NEXTTOKEN;CALL NEXTTOKEN;
930	3	84	IF TOK EQC ',' THEN
940	3	85	BEGIN
950	3	86	CALL NEXTTOKEN;T = TOK;CALL NEXTTOKEN;T=T&TOK;
960	4	87	IF EQC 'ELT AME=' T EN
970	4	88	BEGIN
980	4	89	CALL NEXTTOKEN;
990	5	90	TYT= TOK&T,S3;
1000	5	91	IF ( TYT EQC '*TPFS*' ) AND
1010	5	92	( TOK NEC '*TPFS,NAMES*' )
1020	5	93	T EN BEGIN
1030	5	94	CALL NEXTTOKEN;
1040	6	95	IF TOK NEC ','
1050	6	96	THEN RETURN;
1060	6	97	ELSE BEGIN
1070	6	98	CALL NEXTTOKEN; T=TOK;CALL NEXTTOKEN; T = T&TOK;
1080	7	99	IF T EQC 'USER=' THEN
1090	7	100	BEGIN



1610	3	152	CALL NEXTTOKEN; CALL NEXTTOKEN;
1620	4	153	IF TOK EQC ')' THEN BEGIN
1630	4	154	CALL NEXTTOKEN;
1640	4	155	RETURN;
1650	5	156	END;
1660	5	157	END;
1670	4	158	END;
1680	3	159	END;
1690	2	160	END;
1700	1	161	ELSE IF TOK EQC 'SLAVETASK'
1710	1	162	THEN BEGIN
1720	1	163	CALL NEXTTOKEN;
1730	2	164	IF TOK EQC '('
1740	2	165	THEN BEGIN
1750	2	166	CALL NEXTTOKEN; CALL NEXTTOKEN;
1760	3	167	IF TOK EQC ')' THEN BEGIN
1770	3	168	CALL NEXTTOKEN;
1780	3	169	RETURN;
1790	4	170	END;
1800	4	171	END;
1810	3	172	END;
1820	2	173	END;
1830	1	174	WRITE ' ** ERROR ** SYNTAX ERROR ON FORK/STARTUP STATEMENT ';
1840	1	175	END;
1850		176	
1860		177	
1870		178	
1880		179	PROCEDURE PROCWAIT;
1890	1	180	CALL NEXTTOKEN;
1900	1	181	IF TOK EQC '('
1910	1	182	THEN BEGIN
1920	1	183	CALL NEXTTOKEN; CALL NEXTTOKEN;
1930	2	184	LOOP: IF TOK EQC ')' THEN BEGIN CALL NEXTTOKEN; RETURN; END;
1940	2	185	ELSE IF TOK EQC ','
1950	2	186	THEN BEGIN
1960	2	187	CALL NEXTTOKEN; CALL NEXTTOKEN;
1970	2	188	GO TO LOOP;
1980	3	189	END;
1990	3	190	END;
2000	2	191	END;
2010	1	192	WRITE ' ** ERROR ** SYNTAX ERROR ON AWAIT STATEMENT ';
2020	1	193	END;
2030		194	
2040		195	
2050		196	
2060		197	
2070		198	PROCEDURE PROCOTHER;
2080	1	199	LASTTOK = TOK;
2090	1	200	CALL NEXTTOKEN;
2100	1	201	IF (LASTTOK EQC 'LOCK') AND (TOK EQC 'OF') THEN RETURN;
2110	1	202	IF TOK EQC '('

2120	1	203	THEN BEGIN
2130	1	204	CALL NEXTTOKEN; CALL NEXTTOKEN;
2140	2	205	IF TOK EQC ')' THEN BEGIN
2150	2	206	THEN BEGIN
2160	2	207	CALL NEXTTOKEN;
2170	3	208	RETURN;
2180	3	209	END;
2190	2	210	END;
2200	1	211	WRITE ERR;
2210	1	212	END;
2220		213	
2230		214	
2240		215	
2250		216	PROCEDURE PROCWAIT;
2260	1	217	CALL NEXTTOKEN;
2270	1	218	IF TOK EQC 'FOREVER'
2280	1	219	THEN BEGIN CALL NEXTTOKEN; RETURN; END;
2290	1	220	ELSE BEGIN
2300	1	221	IF TOK EQC '('
2310	2	222	THEN BEGIN
2320	2	223	CALL NEXTTOKEN; CALL NEXTTOKEN;
2330	3	224	IF TOK EQC ')' THEN BEGIN
2340	3	225	THEN BEGIN
2350	3	226	CALL NEXTTOKEN;
2360	4	227	RETURN;
2370	4	228	END;
2380	3	229	END;
2390	2	230	END;
2400	1	231	WRITE ERR;
2410	1	232	END;
2420		233	
2430		234	
2440		235	
2450		236	PROCEDURE PROCFETCH;
2460	1	237	CALL NEXTTOKEN;
2470	1	238	IF TOK EQC 'STAYUR'
2480	1	239	THEN BEGIN
2490	1	240	CALL NEXTTOKEN;
2500	2	241	IF TOK EQC 'OF'
2510	2	242	THEN BEGIN
2520	2	243	CALL NEXTTOKEN;
2530	3	244	IF TOK EQC 'SLAVETASKS'
2540	3	245	THEN BEGIN
2550	3	246	CALL NEXTTOKEN;
2560	4	247	RETURN;
2570	4	248	END;
2580	3	249	RETURN;
2590	3	250	END;
2600	2	251	END;
2610	1	252	WRITE ERR;
2620	1	253	END;

2630		254	PROCEDURE PROCACTIVE;
2640	1	255	CALL NEXTTOKEN;
2650	1	256	IF TOK EQC 'SLAVETASK'
2660	1	257	THEN BEGIN
2670	1	258	CALL NEXTTOKEN;
2680	2	259	IF TOK EQC '('
2690	2	260	THEN BEGIN
2700	2	261	CALL NEXTTOKEN; CALL NEXTTOKEN;
2710	3	262	IF TOK EQC ')'
2720	3	263	THEN BEGIN CALL NEXTTOKEN; RETURN; END;
2730	3	264	END;
2740	2	265	END;
2750	1	266	WRITE ERR;
2760	1	267	END;
2770		268	
2780		269	
2790		270	PROCEDURE PROCDEDIC;
2800	1	271	CALL NEXTTOKEN;
2810	1	272	IF TOK EQC 'PROCESSOR'
2820	1	273	THEN BEGIN
2830	1	274	CALL NEXTTOKEN;
2840	2	275	IF TOK EQC '('
2850	2	276	THEN BEGIN
2860	2	277	CALL NEXTTOKEN; CALL NEXTTOKEN;
2870	3	278	IF TOK EQC ')'
2880	3	279	THEN BEGIN CALL NEXTTOKEN; RETURN; END;
2890	3	280	END;
2900	2	281	END;
2910	1	282	WRITE ERR;
2920	1	283	END;
2930		284	
2940		285	
2950		286	PROCEDURE PROCQUIT;
2960	1	287	CALL NEXTTOKEN;
2970	1	288	IF TOK EQC 'BY'
2980	1	289	THEN BEGIN
2990	1	290	CALL NEXTTOKEN;
3000	2	291	IF TOK EQC 'CHECKING'
3010	2	292	THEN BEGIN
3020	2	293	CALL NEXTTOKEN;
3030	3	294	IF TOK EQC '('
3040	3	295	THEN BEGIN
3050	3	296	CALL NEXTTOKEN; CALL NEXTTOKEN;
3060	4	297	IF TOK EQC ')'
3070	4	298	THEN BEGIN CALL NEXTTOKEN; RETURN; END;
3080	4	299	END;
3090	3	300	END;
3100	2	301	END;
3110	1	302	ELSE RETURN;
3120	1	303	WRITE ERR;
3130	1	304	END;



```

3140      305  PROCEDURE PROCSNOOPY;
3150      1   306  CALL NEXTTOKEN;
3160      1   307  IF ( TOK EQC 'ON' ) OR
3170      1   308  ( TOK EQC 'OFF' )
3180      1   309  THEN BEGIN CALL NEXTTOKEN; RETURN; END;
3190      1   310  WRITE ERR;
3200      1   311  END;
3210      312
3220      313
3230      314
3240      315  PROCEDURE PROCORIG;
3250      1   316  CALL NEXTTOKEN;
3260      1   317  IF TOK EQC 'MODE'
3270      1   318  THEN BEGIN CALL NEXTTOKEN; RETURN; END;
3280      1   319  WRITE ERR;
3290      1   320  END;
3300      321
3310      322
3320      323
3330      324  OPEN WRITE INTO 'SCRATCHS:SCRATCH1';
3340      325  CALL READLN;
3350      326  CALL NEXTTOKEN;
3360      327  WHILE TOK NEC S'EOF'
3370      328  DO BEGIN
3380      329  STRT:
3390      1   330  IF TOK EQC '*COMPILE'
3400      1   331  THEN BEGIN CALL PROCCPS;GO TO STRT; END;
3410      1   332  ELSE IF (TOK EQC 'STARTUP') OR (TOK EQC 'FORK')
3420      1   333  THEN BEGIN CALL PROCSTRUP; GO TO STRT; END;
3430      1   334  ELSE IF TOK EQC 'WAIT'
3440      1   335  THEN BEGIN CALL PROCWAIT; GO TO STRT; END;
3450      1   336  ELSE IF TOK EQC 'AWAIT'
3460      1   337  THEN BEGIN CALL PROCAWAIT;GO TO STRT; END;
3470      1   338  ELSE IF (TOK EQC 'ACTIVATE') OR
3480      1   339  (TOK EQC 'DEACTIVATE')
3490      1   340  THEN BEGIN CALL PROCACTIVE;GO TO STRT; END;
3500      1   341  ELSE IF TOK EQC 'DEDICATE'
3510      1   342  THEN BEGIN CALL PROCDERIC;GO TO STRT; END;
3520      1   343  ELSE IF TOK EQC 'ORIGINAL'
3530      1   344  THEN BEGIN CALL PROCORIG;GO TO STRT; END;
3540      1   345  ELSE IF TOK EQC 'SNOOPY'
3550      1   346  THEN BEGIN CALL PROCSNOOPY;GO TO STRT; END;
3560      1   347  ELSE IF TOK EQC 'QUIT'
3570      1   348  THEN BEGIN CALL PROCOUIT;GO TO STRT; END;
3580      1   349  ELSE IF TOK EQC 'FETCH'
3590      1   350  THEN BEGIN CALL PROCFETCH;GO TO STRT;END;
3600      1   351  ELSE IF (TOK EQC 'COBEGIN') OR
3610      1   352  (TOK EQC 'SIGNAL' ) OR
3620      1   353  (TOK EQC 'RELEASE') OR
3630      1   354  (TOK EQC 'LOCK' ) OR
3640      1   355  (TOK EQC 'UNLOCK' ) OR

```

```

3650 1 356
3660 1 357
3670 1 358
3680 1 359
3690 1 360
3700 2 361
3710 2 362
3720 1 363
3730 1 364
3740 1 365
3750 1 366
      END;
      CLOSE WRITE INTO 'SCRATCH$,SCRATCH1';
      STOP;
END MDC 0 ERRORS 0 WARNINGS 0 REMARKS

```

```

(TOK EOC 'SUSPEND') OR
(TOK EOC 'SNAPDUMP') OR
(TOK EOC 'CLEAR' )
THEN BEGIN
  CALL PROCOTHER;
  GO TO STRT;
END;
ELSE CALL NEXTTOKEN;

```



```

590      50      NT = 0;
500      51      IOINITLOCK = S'NUL';
610      52      IOTERMLOCK = S'NUL';
620      53      IOINITFLAG = S'NUL';
630      54      IOTERMFLAG = S'NUL';
640      55      OPEN WRITE INTO 'INFO$4';
650      56
660      57
670      58
680      59
690      1 60      MACRO CPSSLAVE TRIGGER PROTECT
700      1 61      < MCPS 'OPTIONS' '=' W1:<ID> ',' 'ELTNAME' '=' W2:<ID> ',' 'USER' '=' W3:<ID> ',' 'SIZE' '='
710      1 62      W4:<NUM> ',' 'PRIORITY' '=' W5:<NUM> ',' 'STARTTIME' '=' W6:<NUM>>;
720      1 63      NT = NT+1;
730      1 64      USER(NT)=TRIM(W3);
740      1 65      PROGSIZE(NT)=TRIM(W4);
750      1 66      PRIORITY(NT)=TRIM(W5);
760      1 67      STARTTIME(NT)=TRIM(W6);
770      1 68      ANSWER S'DELB'@CPSS,'@ W1 @ ' '@ W2 @ ','USER='@ USER(NT);
780      69      END;
790      70
800      71
810      72      MACRO CPSROUTINE TRIGGER PROTECT
820      1 73      < MCPS 'OPTIONS' '=' W1:<ID> ',' 'ELTNAME' '=' W2:<ID> >;
830      1 74      ANSWER S'DELB'@CPSS,'@ W1 @ ' '@ W2 @ ;
840      1 75      END;
850      76
860      77
870      78
880      79      MACRO IOINITLCK TRIGGER PROTECT
890      1 80      < W1:<ID> ':' 'IOINITIATION' 'LOCK' 'OF' 'QUEUE' ':'>;
900      1 81      IOINITLOCK = TRIM(W1);
910      1 82      ANSWER ' '@ W1 @ ':' @ 'QUEUE' @ ':' ;
920      1 83      END;
930      84
940      85
950      86
960      87
970      88      MACRO IOTERMLCK TRIGGER PROTECT
980      1 89      < W1:<ID> ':' 'IOTERMINATION' 'LOCK' 'OF' 'QUEUE' ':'>;
990      1 90      IOTERMLOCK = TRIM(W1);
1000     1 91      ANSWER ' '@ W1 @ ':' @ 'QUEUE' @ ':' ;
1010     1 92      END;
1030     93
1040     94
1050     95
1060     96      MACRO IOINITFLG TRIGGER PROTECT
1070     1 97      < W1:<ID> ':' 'IOINITIATION' 'FLAG' 'OF' 'QUEUE' ':'>;
1080     1 98      IOINITFLAG = TRIM(W1);
1090     1 99      ANSWER ' '@ W1 @ ':' @ 'QUEUE' @ ':' ;
1100     1 100     END;

```

```

1140      101      MACRO IOTERM TRIGGER PROTECT
1150      1      102      < W1:<ID> ':' ' IOTERMINATION' 'FLAG' 'OF' 'QUEUE' ':'>;
1160      1      103      IOTERMFLAG = TRIM(W1);
1170      1      104      ANSWER ' ' & W1 & ':' & 'QUEUE' & ':' ;
1180      1      105      END;
1190      106
1200      107
1210      108
1220      109      MACRO ENDOFTEXT TRIGGER PROTECT< EOTX >;
1230      1      110      A = NT&'&IOINITLOCK&'&IOTERMLOCK&'&IOINITFLAG&'&IOTERMFLAG;
1240      1      111      WRITE A INTO 'INFO$';
1250      1      112      I = 1;
1260      1      113      WHILE I LE NT DO BEGIN
1270      1      114          A = PROGSIZE(I) & ' ' & PRIORITY(I) & ' ' &
1280      2      115          ' ' & STARTTIME(I);
1290      2      116          WRITE A INTO 'INFO$';
1300      2      117          I = I + 1 ;
1310      2      118          END;
1320      1      119      CLOSE WRITE INTO 'INFO$.';
1330      1      120      END;
END MDC 0 ERRORS 0 WARNINGS 0 REMARKS

```



```

590      50      TOKEN EOTX<S'EOF'>;
600      51      TOKEN EQSIGN<'='>;
610      52
620      53
630      54
640      55      ERROR01 = ZZ&' ** ERROR ** UNDECLARED PROCEDURE IN LINE ' ;
650      56      ERROR02 = ZZ&' ** ERROR ** UNDECLARED QUEUE TYPE VARIABLE IN LINE ' ;
660      57      WARNING01 = ZZ&' ** WARNING ** CHECK THE ACTIVITY NAME(S) IN LINE';
670      58
680      59
690      60
700      61      PROG_ID ='MAIN';
710      62      PROC_FLAG = 0;
720      63      VAR_FLAG = 0;
730      64      BEGIN_FLAG=0;
740      65      COBEGIN_FLAG=S'NUL';
750      66      COBEGIN_COUNT=0;
760      67      MAINPROG_FLAG=1;
770      68      SACN=0;
780      69      CPS_COUNT =0;
790      70      SLVTASKNAME = S'NUL' ;
800      71      SLVTASKCNT = 0;
810      72      DECLARE R(*);
820      73
830      74
840      75
850      76      OPEN READ FROM 'INFOS&';
860      77      READ A FROM 'INFOS&';
870      78      TOKENIZE A INTO R;
880      79      NT=TRIM(R(1));
890      80      IOINITLOCK = TRIM(R(3));
900      81      IOTERMLOCK = TRIM(R(5));
910      82      IOINITFLAG = TRIM(R(7));
920      83      IOTERMFLAG = TRIM(R(9));
930      84
940      85
950      86
960      87      DECLARE QUEUES(**);
970      88      DECLARE VARIABLES(**);
980      89      DECLARE PROCS(**);
990      90      DECLARE ACTID(**);
1000     91      DECLARE ELTNAME(*);
1010     92
1020     93
1030     94
1040     95      PROCS('MAIN')='MAIN';
1050     96      ACN=0 ;
1060     97
1070     98
1080     99
1090    100

```

```

1100      101  MACRO AWAIT TRIG PROT<'AWAIT'(SL)OP(SL) W1:<<ID('')>>.(SL) W2:< CP(SL)('')>>;
1110      102  IF TRIM(W2)[*,,] EQC ' ; ' THEN WEND ='END'; ; ELSE WEND='END';
1120      103  ACNAMES=TRIM(W1);
1130      104  NM = 0;
1140      105  I = 2 ; NAME = ACNAMES[1,1];
1150      106  WHILE I LE LENGTH(ACNAMES)
1160      107  DO BEGIN
1170      108  IF (ACNAMES[I,1] EQC ' , ' ) OR
1180      109  ( I EQ LENGTH(ACNAMES) )
1190      110  THEN BEGIN
1200      111  IF( I EQ LENGTH(ACNAMES) ) AND
1210      112  ( ACNAMES[I,1] NEC ' , ' )
1220      113  THEN NAME=NAME&ACNAMES[I,1];
1230      114  IF VOID(ACTID,TRIM(NAME))
1240      115  THEN BEGIN
1250      116  WRITE ZZE WARNING01 & LINE ;
1260      117  ACN=ACN+1;
1270      118  ACTID(NAME)=ACN;
1280      119  END;
1290      120  M=ACTID(TRIM(NAME));
1300      121  J=1; N=1;
1310      122  WHILE J LE M
1320      123  DO BEGIN
1330      124  N=N*2;
1340      125  J=J+1;
1350      126  END;
1360      127  NM=NM+N;
1370      128  IF I LT LENGTH(ACNAMES)
1380      129  THEN BEGIN
1390      130  I=I+1;
1400      131  NAME= ACNAMES[I,1];
1410      132  END;
1420      133  ELSE IF ACNAMES[I,1] NEC ' ' THEN NAME=NAME&ACNAMES[I,1];
1430      134  I=I+1;
1440      135  END;
1450      136  END;
1460      137  ANS 000 ' BEGIN 'GZZ;
1470      138  ANS 000'SCODE L R15,A0 'GZZ;
1480      139  ANS 000'SCODE LA A0,('G NM 0')'GZZ;
1490      140  ANS 000'SCODE ER AWAITS 'GZZ;
1500      141  ANS 000'SCODE L A0,R15 'GZZ;
1510      142  ANS 000 WEND 'GZZ;
1520      143  END;
1530      144
1540      145
1550      146
1560      147  MACRO M1 TRIG <M:<'PROCEDURE'>>;
1570      148  PROC_FLAG=1;
1580      149  BEGIN_FLAG=0;
1590      150  RETURN M;
1600      151  END;

```



```

1610      152  MACRO M2 TRIG PROT <M:<OP>>;
1620      153  IF PROC_FLAG EQ 1
1630      154  THEN VAR_FLAG = 1;
1640      155  ELSE VAR_FLAG = 0;
1650      156  RETURN M;
1660      157  END;
1670      158
1680      159
1690      160
1700      161  MACRO M5 <M:<<('VAR') (SL) ID (SL) ':' (SL)ID (SL)(';')> .>>;
1710      162  ANS M;
1720      163  END;
1730      164
1740      165
1750      166
1760      167  MACRO PROC_DCL TRIG
1770      168  <PROCNAME:<ID>(SL)ARGS:<<OP(SL)M5(SL)CP(SL)':'> <'MONITOR'\ 'PROCESS'> ':'>;
1780      169  PROCS(TRIM(PROCNAME))=TRIM(PROCNAME);
1790      170  ANS PROCNAME & ARGS & 'EXTERNAL 1' ;
1800      171  END;
1810      172
1820      173
1830      174
1840      175  MACRO VAR_DCL1 TRIG PROT <('VAR')(SL) NAME:<<ID (';')> .>> (SL)':'(SL)TYPE:<ID> (SL)':'> ;
1850      176  IF VAR_FLAG EQ 1 THEN FAIL;
1860      177  IF MAINPROG_FLAG EQ 0 THEN FAIL;
1870      178  J = 1; W = ' ';NAME=TRIM(NAME)&' ';
1880      179  LOOP:
1890      180  IF TRIM(NAME)[J,1] NEC ' ';
1900      181  THEN W = W & TRIM(NAME)[J,1] ;
1910      182  ELSE BEGIN
1920      183  IF TRIM(TYPE) EOC 'QUEUE'
1930      184  THEN QUEUES(TRIM(W))='INTEGER';
1940      185  ELSE VARIABLES(TRIM(W))=TRIM(TYPE);
1950      186  W = ' ';
1960      187  END;
1970      188  J = J+1;
1980      189  IF TRIM(W) EOC 'VAR' THEN W = ' ';
1990      190  IF J LE LENGTH(TRIM(NAME)) THEN GO TO LOOP;
2000      191  END;
2010      192
2020      193
2030      194
2040      195  MACRO VAR_DCL2 TRIG PROT
2050      196  <('VAR')(SL)NAME:<<ID (';')> .>>(SL)':'(SL)TYPE:<<'ARRAY'(SL)'C'<<<NUM \ ID>(TP)<NUM \ ID>(';')>>.>>
2060      197  (SL) 'J' (SL)'OF'(SL) ID > (SL) ':'> ;
2070      198  IF VAR_FLAG EQ 1 THEN FAIL;
2080      199  IF MAINPROG_FLAG EQ 0 THEN FAIL;
2090      200  J = 1; W = ' ';NAME=TRIM(NAME)&' ';
2100      201  LOOP:
2110      202  IF TRIM(NAME)[J,1] NEC ' ';

```

```

2120 1 203 THEN W = W & TRIM(NAME)[J,1] ;
2130 1 204 ELSE BEGIN
2140 1 205 IF TRIM(TYPE) EQC 'QUEUE'
2150 2 206 THEN QUEUES(TRIM(W))='INTEGER';
2160 2 207 ELSE VARIABLES(TRIM(W))=TRIM(TYPE);
2170 2 208 W = ' ';
2180 2 209 END;
2190 1 210 J = J+1;
2200 1 211 IF TRIM(W) EQC 'VAR' THEN W = ' ';
2210 1 212 IF J LE LENGTH(TRIM(NAME)) THEN GO TO LOOP;
2220 1 213 END;
2230 214
2240 215
2250 216
2260 217 MACRO M3 TRIG PROT <M:<CP>>;
2270 1 218 IF PROC_FLAG EQ 1
2280 1 219 THEN BEGIN
2290 1 220 PROC_FLAG = 0 ;
2300 2 221 VAR_FLAG = 0 ;
2310 2 222 END;
2320 1 223 RETURN M;
2330 1 224 END;
2340 225
2350 226
2360 227
2370 228 MACRO M4 TRIG PROT <'VAR'>;
2380 1 229 IF (VAR_FLAG EQ 1) OR (MAINPROG_FLAG EQ 0) THEN RETURN 'VAR' ;
2390 1 230 END;
2400 231
2410 232
2420 233
2430 234 MACRO MACBEG TRIG PROT <'BEGIN'(SL) (';')>;
2440 1 235 BEGIN_FLAG = BEGIN_FLAG +1;
2450 1 236 IF BEGIN_FLAG NE 1 THEN FAIL;
2460 1 237 IF MAINPROG_FLAG EQ 1
2470 1 238 THEN BEGIN
2480 1 239 N=3*SIZE(PROCS)+SIZE(QUEUES)+4+900+900 ;
2490 2 240 NN = NT * 3+ N ;
2500 2 241 ANS 'VAR GLOBALS : ARRAY (,1 0, 'ENN 0.0') OF INTEGER ;'ZZ;
2510 2 242 IF SIZE(VARIABLES) NE 0
2520 2 243 THEN BEGIN
2530 2 244 NAME = FIRST(VARIABLES);
2540 3 245 LNAME = LAST(VARIABLES);
2550 3 246 WHILE NAME NEC LNAME
2560 3 247 DO BEGIN
2570 3 248 ANS ' ' & NAME & ' : ' & VARIABLES(NAME) & ' ;' & ZZ ;
2580 4 249 NAME = NEXT(VARIABLES,NAME);
2590 4 250 END;
2600 3 251 ANS ' ' & LNAME & ' : ' & VARIABLES(LNAME) & ' ;' & ZZ ;
2610 3 252 ANS QQ&'$CODE RSCCELL* T$CELL'&ZZ;
2620 3 253 ANS QQ&'$CODE CBGNST* T$CELL'&ZZ;

```

2530 3 254  
 2640 3 255  
 2650 3 256  
 2660 3 257  
 2670 3 258  
 2680 3 259  
 2690 4 260  
 2700 4 261  
 2710 4 262  
 2720 3 263  
 2730 3 264  
 2740 3 265  
 2750 3 266  
 2760 3 267  
 2770 4 268  
 2780 4 269  
 2790 4 270  
 2800 3 271  
 2810 3 272  
 2820 3 273  
 2830 3 274  
 2840 3 275  
 2850 4 276  
 2860 4 277  
 2870 3 278  
 2880 3 279  
 2890 3 280  
 2900 3 281  
 2910 3 282  
 2920 4 283  
 2930 4 284  
 2940 4 285  
 2950 4 286  
 2960 3 287  
 2970 3 288  
 2980 3 289  
 2990 3 290  
 3000 3 291  
 3010 4 292  
 3020 4 293  
 3030 3 294  
 3040 3 295  
 3050 3 296  
 3060 3 297  
 3070 3 298  
 3080 3 299  
 3090 3 300  
 3100 3 301  
 3110 3 302  
 3120 4 303  
 3130 4 304

```

ANS Q06'SCODE SYSTEMHOLD* T$CELL'6ZZ;
J=1;N=SIZE(QUEUES);
NAME= FIRST(QUEUES);
WHILE J LE N
DO BEGIN
  ANS Q06'SCODE '&NAMEC'*'6'                                'C1,13=LENGTH(NAME)]
                                                                C'T$CELL'6ZZ;
  J=J+1; NAME=NEXT(QUEUES,NAME);
END;
J=1;N=SIZE(PROCS);
NAME= FIRST(PROCS);
WHILE J LE N
DO BEGIN
  ANS Q06'SCODE T$'&NAMEC'*'6'                                'C1,13=LENGTH(NAME)]
                                                                C'T$CELL'6ZZ;
  J=J+1; NAME=NEXT(PROCS,NAME);
END;
J=1;N=SIZE(PROCS);
NAME= FIRST(PROCS);
WHILE J LE N
DO BEGIN
  ANS Q06'SCODE XSS'&NAMEC'*'6'                                'C1,13=LENGTH(NAME)]6'RES 1'6ZZ;
  J=J+1; NAME=NEXT(PROCS,NAME);
END;
ANS Q06'SCODE CBGNAME* RES 1'6ZZ;
K=1;
WHILE K LE NT
DO BEGIN
  ANS Q06' $CODE INTACTNAME'&K6'* RES 1'6ZZ ;
  ANS Q06' $CODE SAVEAD'&K6'* RES 1'6ZZ ;
  ANS Q06' $CODE RTADDR'&K6'* RES 1'6ZZ ;
  K=K+1 ;
END ;
J=1;N=SIZE(PROCS);
NAME= FIRST(PROCS);
WHILE J LE N
DO BEGIN
  ANS Q06'SCODE F$'&NAMEC'*'6'                                'C1,13=LENGTH(NAME)]6'RES 1'6ZZ;
  J=J+1; NAME=NEXT(PROCS,NAME);
END;
N=3*SIZE(PROCS) + SIZE(QUEUES) +4*900*900;
NN = NT * 3 + N;
ANS Q06'SCODE SVRGAD* RES 900 '6 ZZ;
ANS Q06'SCODE SVRGA1* RES 900 '6 ZZ;
ANS Q06'SCODE RES '6 'SIZES=136='&NNE ZZ&Q06'SCODE $(1)'6ZZ;
K=4;
WHILE K LE NT
DO BEGIN
  ANS Q06'SCODE CONTNG'&K6'* RES 2 '6ZZ ;
  ANS ' $CODE S AU,SAVEAD'6 K 6 ZZ ;
  ANS ' $CODE L,H2 AU,CONTNG'6 K 6 ZZ ;

```

```

3140 4 305      ANS %CODE A,U      AU,1      'E ZZ ;
3150 4 306      ANS %CODE S      AO,RTADDR'E K E ZZ ;
3160 4 307      ANS %CODE L      AU,SAVEAD'E K 'E ZZ ;
3170 4 308      ANS %CODE ER     DACTS 'E ZZ ;
3180 4 309      ANS %CODE ER     LEVELS 'E ZZ ;
3190 4 310      ANS %CODE J      *RTADDR'E K E ZZ ;
3200 4 311      K=K+1;
3210 4 312      END;
3220 3 313      ANS ZZE%CODE COBEGIN* 'EZZ;
3230 3 314      ANS %CODE L      AO,(''CBG'')'EZZ;
3240 3 315      ANS %CODE ER     NAMES 'EZZ;
3250 3 316      ANS %CODE S      AO,CBGNAM'EZZ;
3260 3 317      ANS ZZE%CODE TS     CBGNST 'EZZ;
3270 3 318      ANS ZZE%CODE C$TSA  CBGNST 'EZZ;
3280 3 319      ANS %CODE ER     DACTS 'EZZ;
3290 3 320      ANS %CODE ER     EXITS 'EZZ;
3300 3 321      ANS ' BEGIN'EZZ;
3310 3 322      ANS ' BEGIN'EZZ;
3320 3 323      ANS %CODE S      X10,XSS'EPROG_IDEZZ;
3330 3 324      N=SIZE(PROCS) + SIZE(QUEUES) +3;
3340 3 325      ANS QOE%CODE 'E' L,U  AO,057'EZZ;
3350 3 326      ANS QOE%CODE 'E'I DO 'ENG' , S2 RSCCELL=1+1'EZZ;
3360 3 327      ANS QOE%CODE 'E'I DO 'ENG' , S,S2 AO,RSCCELL=1+1'EZZ;
3370 3 328      ANS QOE%CODE 'E' ER   TSORGS'EZZ;
3380 3 329      ANS QOE%CODE 'E' L,U  R13,N 'EZZ;
3390 3 330      ANS QOE' END;'EZZ;
3400 3 331      MAINPROG_FLAG=0;
3410 3 332      END;
3420 2 333      END;
3430 1 334      ELSE IF SLVTASKNAME EOC S'NUL'
3440 1 335      THEN BEGIN
3450 1 336      ANS QOE'BEGIN'EZZ;
3460 2 337      ANS QOE' BEGIN'EZZ;
3470 2 338      ANS %CODE SZ     FS'EPROG_IDEZZ;
3480 2 339      ANS %CODE S      X10,XSS'EPROG_IDEZZ;
3490 2 340      ANS %CODE PROCST 'EZZ;
3500 2 341      ANS QOE' END;'EZZ;
3510 2 342      END;
3520 1 343      ELSE BEGIN
3530 1 344      ANS QOE'BEGIN'EZZ;
3540 2 345      ANS QOE' BEGIN'EZZ;
3550 2 346      ANS %CODE L      AO,(0200001,CONTNG'E$SLVTASKCNT'E')'EZZ;
3560 2 347      ANS %CODE ER     IALLS 'E ZZ;
3570 2 348      ANS %CODE L      AO,('' $SLVTASKNAME'E'')';
3580 2 349      ANS ZZE%CODE ER     NAMES 'E ZZ;
3590 2 350      ANS %CODE SZ     AO 'E ZZ;
3600 2 351      ANS %CODE ER     IDENT% 'E ZZ;
3610 2 352      ANS %CODE TZ     AO 'E ZZ;
3620 2 353      ANS %CODE J      $+4 'E ZZ;
3630 2 354      ANS %CODE L,U    A1,100 'E ZZ;
3640 2 355      ANS %CODE ER     TWAITE 'E ZZ;

```

3650 2 356  
 3660 2 357  
 3670 2 358  
 3680 2 359  
 3690 2 360  
 3700 2 361  
 3710 1 362  
 3720 363  
 3730 364  
 3740 365  
 3750 366  
 3760 367  
 3770 368  
 3780 369  
 3790 370  
 3800 1 371  
 3810 1 372  
 3820 1 373  
 3830 1 374  
 3840 1 375  
 3850 1 376  
 3860 1 377  
 3870 1 378  
 3880 2 379  
 3890 2 380  
 3900 1 381  
 3910 1 382  
 3920 1 383  
 3930 1 384  
 3940 1 385  
 3950 2 386  
 3960 2 387  
 3970 2 388  
 3980 3 389  
 3990 3 390  
 4000 2 391  
 4010 2 392  
 4020 2 393  
 4030 3 394  
 4040 3 395  
 4050 3 396  
 4060 3 397  
 4070 3 398  
 4080 4 399  
 4090 4 400  
 4100 3 401  
 4110 3 402  
 4120 3 403  
 4130 3 404  
 4140 3 405  
 4150 3 406

```

ANS %CODE J %5 %G ZZ;
ANS %CODE S AD,INTACTNAME'ESLVTASKCNTGZZ;
ANS %CODE ER DACTS % & ZZ;
ANS %CODE ER LEVELS % & ZZ;
ANS QQC' END;'&ZZ;
END;

END;

MACRO FORK1 TRIG
<'FORK'(SL)'WITH'(SL)'TASKNAME'(SL)OP(SL)W2:<ID>(SL)CP(SL)PROCNAME:<ID>W3:<(SL)W1:CP\CP\','\ID\':'>.>>;
DECLARE W(*);
IF SIZE(W1) EQ 1 THEN IF TRIM(W1(1)) EOC ';' THEN FAIL;
IF TRIM(W1(1)) EOC 'ELSE' THEN FAIL;
ANS QQC'%CODE FORK 'EPROCNAME%' %ZZ;
IF COBEGIN_FLAG NEC S'NUL' THEN COBEGIN_COUNT = COBEGIN_COUNT + 1;
IF VOID(ACTID,TRIM(W2)) THEN BEGIN
ACN=ACN+1;
ACTID(TRIM(W2))=ACN;
END;
IF VOID(PROCS,TRIM(PROCNAME)) THEN WRITE ERRORLINE;
J=1; K=0; L=0; NN=SIZE(W1);
WHILE J LE NN
DO BEGIN
IF (TRIM(W1(J)) EOC '(') OR
(TRIM(W1(J)) EOC ',')
THEN BEGIN
K=K+1;
W(K)=TRIM(W1(J));
END;
ELSE IF TRIM(W1(J)) EOC ')'
THEN BEGIN
K=K+1; W(K)=TRIM(W1(J));
N=SIZE(W);
K=1;
ANS PROCNAME;
WHILE K LE N DO BEGIN
ANS W(K);
K=K+1;
END;
OAID=ACTID(TRIM(W2));
AID=(OAID/R)*10+(OAID/(OAID/R)*8);
IF TRIM(W3)[C+,1] EOC ';' THEN ANS %G ZZ;
ANS %CODE'E' L,U R15,%& OAID & ZZ;
ANS %CODE'E' S R15,0,AU %G ZZ;
ANS %CODE'E' L R15,AU %G ZZ;
  
```

```

4160 3 407 IF AID LE 7
4170 3 408 THEN
4180 3 409 ANS '%CODE%' L AD,(00%'CAID%'01,' &TRIM(PROCNAME)&')%ZZ;
4190 3 410 ELSE
4200 3 411 ANS '%CODE%' L AD,(00%'CAID%'01,' &TRIM(PROCNAME)&')%ZZ;
4210 3 412 ANS '%CODE%' ER FORK;%ZZ;
4220 3 413 ANS '%CODE%' L AD,R15 '% ZZ;
4230 3 414 RETURN;
4240 3 415 END;
4250 2 416 ELSE
4260 2 417 IF (TRIM(W1(J)) EQC 'FORWARD') OR
4270 2 418 (TRIM(W1(J)) EQC 'FORW')
4280 2 419 THEN ANS QQC%'%CODE' DELETE ARGUMENT '&L%' '%ZZ;
4290 2 420 ELSE BEGIN
4300 2 421 L=L+1;
4310 3 422 K=K+1;
4320 3 423 W(K)=TRIM(W1(J));
4330 3 424 END;
4340 2 425 J=J+1;
4350 2 426 END;
4360 1 427 END;
4370 428
4380 429
4390 430
4400 431 MACRO FORK2 TRIG PROT
4410 1 432 <'FORK' (SL) 'WITH' (SL) 'TASKNAME'(SL)OP (SL) W2:<ID> (SL)CP (SL) W3:<PROCNAME:<ID> (';'') > >;
4420 1 433 ANS QQC%'%CODE' FORK '%PROCNAME%' '%ZZ;
4430 1 434 IF COBEGIN_FLAG NEC S'NUL' THEN COBEGETV_COUNT = COBEGIN_COUNT + 1;
4440 1 435 IF VOID(ACTID,TRIM(W2)) THEN BEGIN
4450 1 436 ACN=ACN+1;
4460 2 437 ACTID(TRIM(W2))=ACN;
4470 2 438 END;
4480 1 439 IF VOID(PROCS,TRIM(PROCNAME)) THEN WRITE ERRORLINE;
4490 1 440 ANS PROCNAME;
4500 1 441 OAID=ACTID(TRIM(W2));
4510 1 442 AID=(OAID/8)*10+(OAID-(OAID/8)*8);
4520 1 443 IF TRIM(W3)[*,1] EQC ';' THEN ANS '%;%ZZ;
4530 1 444 ANS '%CODE%' L,U R15,% OAID % ZZ ;
4540 1 445 ANS '%CODE%' S R15,0,AD '% ZZ ;
4550 1 446 ANS '%CODE%' L R15,AD '% ZZ ;
4560 1 447 IF AID LE 7
4570 1 448 THEN ANS '%CODE%' L AD,(00%'CAID%'01,' &TRIM(PROCNAME)&')%ZZ;
4580 1 449 ELSE ANS '%CODE%' L AD,(00%'CAID%'01,' &TRIM(PROCNAME)&')%ZZ;
4590 1 450 ANS '%CODE%' ER FORK;%ZZ;
4600 1 451 ANS '%CODE%' L AD,R15 '% ZZ;
4610 1 452 RETURN;
4620 1 453 END;
4630 454
4640 455
4650 456
4660 457

```

```

4670      458  MACRO COBEGIN TRIG PROT<'COBEGIN'(SL)OP(SL)W1:<ID>(SL) W2:<CP (SL)(';'>>;
4680      459  IF TRIM(W2)[*,,] EQC ';;' THEN WEND = 'END'; ;ELSE WEND = 'END' ;
4690      460  IF COBEGIN_COUNT EQ 0 THEN
4700      461      BEGIN
4710      462          ACN = ACN + 1;
4720      463          SACN = (ACN/8)+10*(ACN=(ACN/8)+8);
4730      464          COBEGIN_COUNT = COBEGIN_COUNT + 1;
4740      465      END;
4750      466  ACTID(TRIM(W1)) = ACN;
4760      467  ANS ZZ & ' BEGIN ' & ZZ;
4770      468  ANS ZZ&'$CODE L R15,AD ' &ZZ;
4780      469  IF SACN LE 7
4790      470      THEN ANS '$CODE L AD,(000'ESACN&'01,COBEGIN)'&ZZ;
4800      471      ELSE ANS '$CODE L AD,(00'ESACN&'01,COBEGIN)'&ZZ;
4810      472  ANS '$CODE ER FORKS ' &ZZ;
4820      473  ANS ZZ&'$CODE L AD,R15 ' &ZZ;
4830      474  ANS '$CODE TS CBNST ' &ZZ;
4840      475  ANS '$CODE C&TSQ CBNST ' &ZZ;
4850      476  ANS WEND & ZZ;
4860      477  END;
4870      478
4880      479
4890      480
4900      481  MACRO COEND TRIG PROT<W1:<'COEND' (SL) (';'>> >;
4910      482  IF TRIM(W1)[*,,] EOC ';;' THEN WEND = 'END'; ;ELSE WEND='END' ;
4920      483  COBEGIN_COUNT = 0;
4930      484  ANS ZZ & ' BEGIN ' & ZZ ;
4940      485  ANS ZZ&'$CODE L R15,AD ' &ZZ;
4950      486  ANS '$CODE L AD,CBGNAME'&ZZ;
4960      487  ANS '$CODE ER ACTS ' &ZZ;
4970      488  ANS '$CODE L AD,P15 ' &ZZ;
4980      489  ANS WEND & ZZ ;
4990      490  END;
5000      491
5010      492
5020      493
5030      494  MACRO STARTUP1 TRIG
5040      495  <'STARTUP' (SL) 'WITH' (SL) 'TASKNAME' (SL)OP (SL) W2:<ID> (SL)CP (SL) PROCNAME:<ID>
5050      496  W3:<(SL) W1:<OP\CP>','\ID>':'> & & > >;
5060      497  IF SIZE(W1) EQ 1 THEN IF TRIM(W1(1)) EOC ';;' THEN FAIL;
5070      498  IF TRIM(W1(1)) EOC 'ELSE' THEN FAIL;
5080      499  DECLARE W(*);
5090      500  ANS 00&'$CODE STARTUP ' &PROCNAME& ' &ZZ;
5100      501  IF COBEGIN_FLAG NEC S'NUL' THEN COBEGIN_COUNT = COBEGIN_COUNT + 1;
5110      502  IF VOID(ACTID,TRIM(W2)) THEN BEGIN
5120      503      ACN=ACN+1;
5130      504      ACTID(TRIM(W2))=ACN;
5140      505  END;
5150      506  IF VOID(PROCS,TRIM(PROCNAME)) THEN WRITE ERRORLINE&LINE;
5160      507  J=1; K=0; L=0; NN=SIZE(W1);
5170      508  WHILE J LE NN

```

```

5180 1 509
5190 1 510
5200 2 511
5210 2 512
5220 2 513
5230 3 514
5240 3 515
5250 2 516
5260 2 517
5270 2 518
5280 3 519
5290 3 520
5300 3 521
5310 3 522
5320 3 523
5330 3 524
5340 4 525
5350 4 526
5360 3 527
5370 3 528
5380 3 529
5390 3 530
5400 3 531
5410 3 532
5420 3 533
5430 3 534
5440 3 535
5450 3 536
5460 3 537
5470 3 538
5480 3 539
5490 3 540
5500 3 541
5510 2 542
5520 2 543
5530 2 544
5540 2 545
5550 2 546
5560 2 547
5570 3 548
5580 3 549
5590 3 550
5600 2 551
5610 2 552
5620 1 553
5630 554
5640 555
5650 556
5660 557
5670 558
5680 559

```

```

DO BEGIN
  IF (TRIM(W1(J)) EQC '(') OR
  (TRIM(W1(J)) EQC ',')
  THEN BEGIN
    K=K+1;
    W(K)=TRIM(W1(J));
  END;
  ELSE IF TRIM(W1(J)) EQC ')'
  THEN BEGIN
    K=K+1; W(K)=TRIM(W1(J));
    N=SIZE(W);
    K=1;
    ANS '%CODE%' L X10,CX105'&ZZ;
    ANS PROCNAME;
    WHILE K LE N DO BEGIN
      ANS W(K);
      K=K+1;
    END;
    OAID=ACTID(TRIM(W2));
    AID=(OAID/8)*10+(OAID-(OAID/8));
    IF TRIM(W3)[K,1] EQC '!' THEN ANS '!&ZZ;
    ANS '%CODE%' L,U R15,'& OAID &ZZ;
    ANS '%CODE%' S R15,0,A0 ' &ZZ;
    ANS '%CODE%' L R15,AU ' &ZZ;
    ANS '%CODE%' TS T'&TRIM(PROCNAME)&ZZ;
    IF AID LE 7
    THEN ANS '%CODE%' L AN,(000'&AID&'01,' &TRIM(PROCNAME)&')&ZZ;
    ELSE ANS '%CODE%' L AN,(00'&AID&'01,' &TRIM(PROCNAME)&')&ZZ;
    ANS '%CODE%' ER FORK&&ZZ;
    ANS '%CODE%' C&TSQ TS'&TRIM(PROCNAME)&ZZ;
    ANS '%CODE%' L AU,R15 ' &ZZ;
    RETURN;
  END;
  ELSE
  IF (TRIM(W1(J)) EQC 'FORWARD') OR
  (TRIM(W1(J)) EQC 'FORW')
  THEN ANS Q0C'%CODE' DELETE ARGUMENT 'CL6' ' &ZZ;
  ELSE BEGIN
    L=L+1;
    K=K+1;
    W(K)=TRIM(W1(J));
  END;
  J=J+1;
END;
END;

```



```

5690      560  MACRO STARTUP2 TRIG PROT
5700      561  <'STARTUP'(SL)'WITH'(SL)'TASKNAME'(SL)OP(SL)W2:<ID> (SL)CP (SL) W3:<PROCNAME:<ID> (';' ) > >;
5710      562  ANS Q06'%CODE' STARTUP '&PROCNAME&' %ZZ;
5720      563  IF COBEGIN_FLAG NEC S'MUL' THEN COREGIN_COUNT = COREGIN_COUNT + 1;
5730      564  IF VOID(ACTID,TRIM(W2)) THEN BEGIN
5740      565          ACN=ACN+1;
5750      566          ACTID(TRIM(W2))=ACN;
5760      567          END;
5770      568  IF VOID(PROCS,TRIM(PROCNAME)) THEN WRITE ERROR01&LINE;
5780      569  ANS '%CODE'&' L X10,CX10%'&ZZ;
5790      570  ANS PROCNAME;
5800      571  IF TRIM(W3)[*,I] EOC ';' THEN ANS ';' & ZZ;
5810      572  OAID=ACTID(TRIM(W2));
5820      573  AID=(OAID/8)*10+(OAID-(OAID/8)*8);
5830      574  ANS '%CODE'&' L,U R15,%& OAID %ZZ;
5840      575  ANS '%CODE'&' S R15,0,AU %ZZ;
5850      576  ANS '%CODE'&' L R15,AU % & ZZ ;
5860      577  ANS '%CODE'&' TS TS'&TRIM(PROCNAME)'&ZZ;
5870      578  IF AID LE 7 THEN
5880      579      ANS '%CODE'&' L AD,(000'&AID&'01,' &TRIM(PROCNAME)'&')'&ZZ;
5890      580      ELSE
5900      581      ANS '%CODE'&' L AD,(00'&AID&'01,' &TRIM(PROCNAME)'&')'&ZZ;
5910      582  ANS '%CODE'&' ER FORKS'&ZZ;
5920      583  ANS '%CODE'&' C&TSQ TS'&TRIM(PROCNAME)'&ZZ;
5930      584  ANS '%CODE'&' L AD,R15 %ZZ;
5940      585  RETURN;
5950      586  END;
5960      587
5970      588
5980      589
5990      590  MACRO WAIT TRIG PROT<'WAIT'(SL)OP(SL) W1:<ID> (SL) W2:<CP(SL)(';' )>>;
6000      591  IF TRIM(W2)[*,I] EOC ';' THEN WEND='END'; ELSE WEND='FND' ;
6010      592  IF VOID(QUEUES,TRIM(W1)) THEN WRITE ERROR02&LINE;
6020      593  ANS ZZ&'BEGIN'&ZZ;
6030      594  ANS '%CODE C&TSQ %& W1 &ZZ;
6040      595  ANS WEND %ZZ;
6050      596  END;
6060      597
6070      598
6080      599
6090      600  MACRO SIGNAL TRIG PROT<'SIGNAL'(SL)OP(SL) #1:<ID>(SL)W2:<CP(SL)(';' )>>;
6100      601  IF TRIM(W2)[*,I] EOC ';' THEN WEND='END'; ELSE WEND='END' ;
6110      602  IF VOID(QUEUES,TRIM(W1)) THEN WRITE ERROR02&LINE;
6120      603  ANS ZZ&'BEGIN'&ZZ;
6130      604  ANS '%CODE C&TSA %& W1 &ZZ;
6140      605  ANS WEND %ZZ;
6150      606  END;
6160      607
6170      608
6180      609
6190      610

```

```

6200      611  MACRO DEDICATE TRIGGER PROTECT <'DEDICATE'(SL)*PROCESSOR'(SL)OP(SL)W1:<NUM>(SL)W2:<CP(SL)(';')>>;
6210      1   612  IF TRIM(W2)[*,] EQC ';;' THEN WEND='END;' ; ELSE WEND='END';
6220      1   613  IF W1 EQ 0 THEN W2 = 1 ;
6230      1   614  IF W1 EQ 1 THEN W2 = 2 ;
6240      1   615  IF W1 EQ 2 THEN W2 = 4 ;
6250      1   616  IF W1 EQ 3 THEN W2 = 8 ;
6260      1   617  ANS ZZ & ' BEGIN ' & ZZ ;
6270      1   618  ANS QO&' $CODE L R15,AD 'E5'EOL';
6280      1   619  ANS QO&' $CODE L,U AD,'GWZE S'EOL';
6290      1   620  ANS QO&' $CODE ER ADED$ 'E S'EOL';
6300      1   621  ANS QO&' $CODE L AD,R15 'E S'EOL';
6310      1   622  ANS ZZ & WEND & ZZ;
6320      1   623  END;
6330      1   624
6340      1   625
6350      1   626
6360      1   627
6370      1   628  MACRO NORMALMODE TRIGGER PROTECT <'ORIGINAL'(SL) W1:<'MODE'(SL)(';')>>;
6380      1   629  IF TRIM(W1)[*,] EQC ';;' THEN WEND='END;' ; ELSE WEND='END';
6390      1   630  ANS ZZ & ' BEGIN ' & ZZ ;
6400      1   631  ANS QO&' $CODE L R15,AD 'E5'EOL';
6410      1   632  ANS QO&' $CODE L,U AD,'O' & S'EOL';
6420      1   633  ANS QO&' $CODE ER ADED$ 'E S'EOL';
6430      1   634  ANS QO&' $CODE L AD,R15 'E S'EOL';
6440      1   635  ANS ZZ & WEND & ZZ;
6450      1   636  END;
6460      1   637
6470      1   638
6480      1   639
6490      1   640  MACRO PROGRAM TRIG PROT
6500      1   641  <'PROGRAM'(SL) PROCNAME:<ID>(SL)W1:<OP(SL)<ID / ', '>(SL)CP(SL)(';')>>;
6510      1   642  IF TRIM(PROCNAME) EQC 'DEFINE' THEN FAIL;
6520      1   643  BEGIN_FLAG=0;
6530      1   644  IF SLVTASKNAME NEC S'NUL'
6540      1   645  THEN BEGIN
6550      1   646  ANS QO&' PROGRAM DEFINE(SLAVETASK'&SLVTASKCNT&');'EZZ;
6560      2   647  ANS QO&' PROCEDURE SLAVETASK'&SLVTASKCNT&';'EZZ;
6570      2   648  ANS QO&' $CODE SLAVETASK ENTRY 'EZZ;
6580      2   649  END;
6590      1   650  ELSE BEGIN
6600      1   651  IF VOID(PROCS,TRIM(PROCNAME)) THEN WRITE ERROR01&LINE;
6610      2   652  PROG_ID=TRIM(PROCNAME);
6620      2   653  RETURN 'PROGRAM'&PROCNAME & W;
6630      2   654  END;
6640      1   655  END;
6650      1   656
6660      1   657
6670      1   658
6680      1   659
6690      1   660
6700      1   661

```

```

6710      662  MACRO PROGDEF TRIG PROT<'PROGRAM'(SL)'DEFINE'(SL)W:<(SL)OP(SL)PROCNAME:<ID>(SL)CP(SL)';>>;
6720      663  IF VOID(PROCS,TRIM(PROCNAME)) THEN WRITE ERROR01&LINE;
6730      664  BEGIN_FLAG=0;
6740      665  PROG_ID=TRIM(PROCNAME);
6750      666  RETURN 'PROGRAM'&' DEFINE'& W;
6760      667  END;
6770      668
6780      669
6790      670
6800      671  MACRO RELEASE TRIG PROT<'RELEASE'(SL)OP(SL)W1:<ID>(SL)W3:<CP(SL)(';')>>;
6810      672  IF TRIM(W3)[*,1] EQC ';' THEN WEND='END'; ELSE WEND='END';
6820      673  IF VOID(PROCS,TRIM(W1)) THEN WRITE ERROR01&LINE;
6830      674  ANS ZZ&'BEGIN'&ZZ;
6840      675  ANS '%CODE      TS      TS'&TRIM(W1)&ZZ;
6850      676  ANS '%CODE      C&TSA  TS'&TRIM(W1)&ZZ;
6860      677  ANS WEND &ZZ;
6870      678  END;
6880      679
6890      680
6900      681
6910      682  MACRO LOCK TRIG PROT<'LOCK'(SL) OP(SL) W1:<ID>(SL)W3:<CP(SL)(';')>>;
6920      683  IF TRIM(W3)[*,1] EOC ';' THEN WEND='END'; ELSE WEND='END';
6930      684  ANS ZZ & 'BEGIN' & ZZ;
6940      685  ANS '%CODE      TS      ' & W1 & ZZ;
6950      686  ANS WEND      & ZZ;
6960      687  END;
6970      688
6980      689
6990      690
7000      691  MACRO UNLOCK TRIG PROT<'UNLOCK'(SL)OP(SL) W1:<ID>(SL)W3:<CP(SL)(';')>>;
7010      692  IF TRIM(W3)[*,1] EOC ';' THEN WEND='END'; ELSE WEND='END';
7020      693  ANS ZZ & 'BEGIN' & ZZ;
7030      694  ANS '%CODE      C&TS      ' & W1 & ZZ;
7040      695  ANS WEND      & ZZ;
7050      696  END;
7060      697
7070      698
7080      699
7090      700  MACRO SUSPEND TRIG PROT<'SUSPEND'(SL)OP(SL)W1:<NUM>(SL)W3:<CP(SL)(';')>>;
7100      701  IF TRIM(W3)[*,1] EQC ';' THEN WEND='END'; ELSE WEND='END';
7110      702  ANS ZZ&' BEGIN' &ZZ;
7120      703  ANS ZZ&'%CODE      L      R15,A1      ' &ZZ;
7130      704  ANS ZZ&'%CODE      L,U     A1,'&TRIM(W1)&ZZ;
7140      705  ANS ZZ&'%CODE      ER      TWAITS'&ZZ;
7150      706  ANS ZZ&'%CODE      L      A1,R15      ' &ZZ;
7160      707  ANS WEND & ZZ;
7170      708  END;
7180      709
7190      710
7200      711
7210      712

```

```

7220      713      MACRO SUSPEND? TRIG PROT<'SUSPEND'(SL)OP(SL)W1:<ID>(SL)W3:<CP(SL)(':'))>;
7230      1       714      IF TRIM(W3)[*,]J EOC '*;' THEN WEND='END'; ELSE WEND='END';
7240      1       715      ANS ZZE'%CODE   SUSPEND  KEYWORD      'EZZ;
7250      1       716      ANS ZZE  W1 &' := 0 ;      ' & ZZ;
7260      1       717      ANS ZZE'  BFGIN' EZZ;
7270      1       718      ANS ZZE'%CODE   ER          TWAITS'EZZ;
7280      1       719      ANS ZZE'%CODE   L           AL,R15      'EZZ;
7290      1       720      ANS WEND  & ZZ;
7300      1       721      END;
7310      722
7320      723
7330      724
7340      725      MACRO CPS2 TRIG PROT <MCPS W1:<ID> W2:<ID> ',' 'USER' '=' W3:< ID > > ;
7350      1       726      SLVTASKNAME = TRIM(W3);
7360      1       727      SLVTASKCNT = SLVTASKCNT + 1 ;
7370      1       728      CPS_COUNT =CPS_COUNT +1;
7380      1       729      ELTNAME(CPS_COUNT)=TRIM(W2);
7390      1       730      IF CPS_COUNT NE 1
7400      1       731      THEN BEGIN
7410      1       732          M=CPS_COUNT - 1 ;
7420      2       733          W3=ELTNAME(M);
7430      2       734          ANS QOE'@CTS,N          'EZZ;
7440      2       735          ANS QOE'OLD   TPF%'PASCAL CODE  'EZZ;
7450      2       736          ANS QOE'ASSUME ASCII OFF  'EZZ;
7460      2       737          ANS QOE'CALL   UNIVAC*CPS'CLEAR'EZZ;
7470      2       738          ANS QOE'SAVE   '&TRIM(W3) EZZ;
7480      2       739          ANS QOE'XCTS          'EZZ;
7490      2       740      END;
7500      1       741      ANS QOE'@BU*PASCAL'S,'&TRIM(W1)EZZ;
7510      1       742      END;
7520      743
7530      744
7540      745
7550      746
7560      747      MACRO CPS TRIG PROT <MCPS W1:<ID> W2:<ID> >;
7570      1       748      CPS_COUNT =CPS_COUNT +1;
7580      1       749      ELTNAME(CPS_COUNT)=TRIM(W2);
7590      1       750      IF CPS_COUNT NE 1
7600      1       751      THEN BEGIN
7610      1       752          M=CPS_COUNT - 1 ;
7620      2       753          W3=ELTNAME(M);
7630      2       754          ANS QOE'@CTS,N          'EZZ;
7640      2       755          ANS QOE'OLD   TPF%'PASCAL CODE  'EZZ;
7650      2       756          ANS QOE'ASSUME ASCII OFF  'EZZ;
7660      2       757          ANS QOE'CALL   UNIVAC*CPS'CLEAR'EZZ;
7670      2       758          ANS QOE'SAVE   '&TRIM(W3) EZZ;
7680      2       759          ANS QOE'XCTS          'EZZ;
7690      2       760      END;
7700      1       761      ANS QOE'@BU*PASCAL'S,'&TRIM(W1)EZZ;
7710      1       762      END;
7720      763

```

```

7730      764
7740      765
7750      766      MACRO END_OF_TEXT TRIG PROT <EOTX>;
7760      767          N=SIZE(ELTNAME);
7770      768          W3=ELTNAME(N);
7780      769          ANS QQE'ACTS,N          'EZZ;
7790      770          ANS QQE'OLD          TPF$PASCOD 'EZZ;
7800      771          ANS QQE'ASSUME ASCII OFF 'EZZ;
7810      772          ANS QQE'CALL UNIVAC+CPS,CLEAR 'EZZ;
7820      773          ANS QQE'SAVE          'G TRIM(W3) 'EZZ;
7830      774          ANS QQE'XCTS          'EZZ;
7840      775          J=I;
7850      776          WHILE J LE N
7860      777              DO BEGIN
7870      778                  ANS QQE'BCPS$S,N          'ELTNAME(J)E',CPS$S/3,'G 'TPF$PASCOD'EZZ;
7880      779                  ANS QQE'RELT,IN          'ELTNAME(J)EZZ;
7890      780                  ANS QQE'ADD          TPF$PASCOD'EZZ;
7900      781                  ANS QQE'ASAM,US          'G ELTNAME(J)EZZ;
7910      782                  J=J+1;
7920      783              END;
7930      784          M=FIRST(ELTNAME);
7940      785          ANS QQE'SMAP,I          'ELTNAME(M)EZZ;
7950      786          ANS QQE'IN          'ELTNAME(M)EZZ;
7960      787          ANS QQE'LIB          BU*PASCAL'EZZ;
7970      788          ANS QQE'END          'EZZ;
7980      789          ANS QQE'BUSE          IMPOTABLE,INPOS'EZZ;
7990      790          ANS QQE'BUSE          PASQOQOQO,INPOS'EZZ;
8000      791          ANS QQE'BSETC,D          'EZZ;
8010      792          ANS QQE'XQT          'ELTNAME(M)EZZ;
8020      793      END;
8030      794
8040      795
8050      796
8060      797      MACRO SNAPDUMP TRIG PROT <'SNAPDUMP'(SL)OP(SL)WT:<ID>(SL)W2:<CP(SL)(';')>>;
8070      798          IF TRIM(W2)[C*,J] EQC ';' THEN WEND='END'; ELSE WEND='END';
8080      799          ANS ZZE'BEGIN'EZZ;
8090      800          ANS ZZE'SCODE L$SNAP ';&TRIM(W1)&'&'&,D,&NTSCELL&,&RCELL';
8100      801          ANS ZZE WEND 'EZZ;
8110      802      END;
8120      803
8130      804
8140      805
8150      806      MACRO SNOOPYON TRIG PROT <'SNOOPY'(SL)WT:<'ON'(SL)(';')>>;
8160      807          IF TRIM(W1)[C*,J] EQC ';' THEN WEND='END'; ELSE WEND='END';
8170      808          ANS ZZE'BEGIN'EZZ;
8180      809          ANS ZZE'SCODE SLJ TONS';
8190      810          ANS ZZE WEND 'EZZ;
8200      811      END;
8210      812
8220      813
8230      814

```

```

8240      815
8250      816
8260      817
8270      818
8280      819
8290      1  820
8300      1  821
8310      1  822
8320      1  823
8330      824
8340      825
8350      826
8360      827
8370      1  828
8380      1  829
8390      1  830
8400      1  831
8410      1  832
8420      1  833
8430      834
8440      835
8450      836
8460      837
8470      1  838
8480      1  839
8490      1  840
8500      1  841
8510      1  842
8520      1  843
8530      1  844
8540      845
8550      846
8560      847
8570      848
8580      849
8590      1  850
8600      1  851
8610      1  852
8620      1  853
8630      1  854
8640      855
8650      856
8660      857
8670      858
8680      1  859
8690      1  860
8700      1  861
8710      1  862
8720      1  863
8730      1  864
8740      1  865

MACRO SNOOPYOFF TRIG PROT <'SNOOPY'(SL) W1:<'OFF'(SL)(';')>>;
  IF TRIM(W1)[*,1] EQC ';;' THEN WEND='END;'; ELSE WEND='END;';
  ANS ZZE 'BEGIN' & ZZ;
  ANS ZZE 'CODE SLJ TOFF';
  ANS ZZE WEND & ZZ;
END;

MACRO CLEAR TRIG PROT <'CLEAR'(SL) OP(SL)W1:<ID>(SL)W2:<CP(SL)(';')>>;
  IF TRIM(W2)[*,1] EQC ';;' THEN WEND='END;'; ELSE WEND='END;';
  IF VOID(QUEUES,TRIM(W1)) THEN WRITE ERROR02&LINE;
  ANS ZZE 'BEGIN' & ZZ;
  ANS 'CODE SZ ' & W1 & ZZ;
  ANS WEND & ZZ;
END;

MACRO QUITBY TRIG PROT<'QUIT'(SL)'BY'(SL)'CHECKING'(SL)OP(SL)W1:<ID> W2:<(SL)CP (';')>>;
  IF TRIM(W2)[*,1] EQC ';;' THEN WEND='END;'; ELSE WEND='END;';
  IF VOID(QUEUES,TRIM(W1)) THEN WRITE ERROR02&LINE;
  ANS ZZE 'BEGIN' & ZZ;
  ANS 'CODE TNZ ' & W1 & ZZ;
  ANS 'CODE ER EXIT' & ZZ;
  ANS WEND & ZZ;
END;

MACRO QUIT TRIG <W1:<'QUIT' (SL)(';')>>;
  IF TRIM(W1)[*,1] EQC ';;' THEN WEND='END;'; ELSE WEND='END;';
  ANS ZZE ' BEGIN ' & ZZ;
  ANS ZZE Q&'CODE ER EXIT' & ZZ;
  ANS WEND & ZZ;
END;

MACRO SLVSTARTUP TRIG PROT <'STARTUP'(SL)'SLAVETASK'(SL)OP(SL)W1:< ID >(SL)W2:< CP(SL)(';')>>;
  IF TRIM(W2)[*,1] EQC ';;' THEN WEND='END;'; ELSE WEND='END;';
  ANS QO ' ' BEGIN ' & ZZ;
  ANS QO ' ' BEGIN ' & ZZ;
  ANS QOE 'CODE L X10,OX10' & ZZ;
  ANS QOE 'CODE L R15,AB ' & ZZ;
  ANS QOE 'CODE L,H2 AB,S,X10' & ZZ;
  ANS QOE 'CODE L,U X9,START' & ZZ;

```

```

8750 1 866 ANS QOE ' END: ' & Z7 ;
8760 1 867 ANS QOE ' CASE ' & W1 & ' OF ' & Z7 ;
8770 1 868 K = 1 ;
8780 1 869 WHILE K LE NT
8790 1 870 DO BEGIN
8800 1 871 ANS QOE ' ' & K & ': BEGIN ' & Z7 ;
8810 2 872 ANS QOE ' %CODE L AD,(000001,SLAVETASK'&K&')'&Z7;
8820 2 873 ANS QOE ' %CODE SZ INTACTNAME'&K&Z7 ;
8830 2 874 ANS QOE ' %CODE L,U X7,INTACTNAME'&K&Z7 ;
8840 2 875 ANS QOE ' END: ' & Z7 ;
8850 2 876 K = K+1 ;
8860 2 877 END;
8870 1 878 ANS QOE ' END: ' & Z7 ;
8880 1 879 ANS QOE ' BEGIN ' & Z7 ;
8890 1 880 ANS QOE ' %CODE ER FORK% ' & Z7 ;
8900 1 881 ANS QOE ' %CODE TZ ,*X7 ' & Z7 ;
8910 1 882 ANS QOE ' %CODE J %+4 ' & Z7 ;
8920 1 883 ANS QOE ' %CODE L,U A1,1U ' & Z7 ;
8930 1 884 ANS QOE ' %CODE ER TWATT% ' & Z7 ;
8940 1 885 ANS QOE ' %CODE J %74 ' & Z7 ;
8950 1 886 ANS QOE ' %CODE L X1U,CX10% ' & Z7 ;
8960 1 887 ANS QOE ' %CODE L A9,R15 ' & Z7 ;
8970 1 888 ANS QOE ' END: ' & Z7 ;
8980 1 889 ANS QOE WEND &Z7;
8990 1 890 END;
9000 1 891
9010 1 892
9020 1 893
9030 1 894
9040 1 895
9050 1 896 MACRO ACTIVATE TRIG PROT<'ACTIVATE'(SL)'SLAVETASK'(SL)OP(SL) W1:< ID >(SL) W2:< CP(SL)((:'))>;
9060 1 897 IF TRIM(W2)[*,,] EQC ' ;' THEN WEND='END;' ; ELSE WEND='END;' ;
9070 1 898 ANS QOE ' BEGIN ' & Z7 ;
9080 1 899 ANS QOE ' CASE ' & W1 & ' OF ' & Z7 ;
9090 1 900 K = 1 ;
9100 1 901 WHILE K LE NT
9110 1 902 DO BEGIN
9120 2 903 ANS QOE ' %K&': BEGIN ' & Z7 ;
9130 2 904 ANS QOE ' %CODE L AD,INTACTNAME'&K&Z7;
9140 2 905 ANS QOE ' % ' & Z7 ;
9150 2 906 K = K+1 ;
9160 1 907 END;
9170 1 908 ANS QOE ' %CODE ER ACT% ' & Z7 ;
9180 1 909 ANS QOE WEND &Z7 ;
9190 1 910 END;
9200 1 911
9210 1 912
9220 1 913
9230 1 914
9240 1 915 MACRO DEACTIVATE TRIG PROT<'DEACTIVATE'(SL)'SLAVETASK'(SL)OP(SL)W1:< ID >(SL)W2:< CP(SL)((:'))>;
9250 1 916 IF TRIM(W2)[*,,] EQC ' ;' THEN WEND='END;' ; ELSE WEND='END;' ;

```

```

9260 1 917 ANS QQ F ' BEGIN ' & ZZ ;
9270 1 918 ANS QQ F ' CASE ' & W1 & ' OF ' & ZZ ;
9280 1 919 K = 1 ;
9290 1 920 WHILE ' Y LE NT
9300 1 921 DO BEGTH
9310 1 922 ANS QOE ' ' & K ' : BEGIN ' & ZZ ;
9320 2 923 ANS QOE ' %CODE L AD,INTACTNAME' & K & ZZ ;
9330 2 924 ANS QOE ' ' & ' END; ' & ZZ ;
9340 2 925 K = K + 1 ;
9350 2 926 END;
9360 1 927 ANS QOE ' END; ' & ZZ ;
9370 1 928 ANS QOE ' %CODE ER TNT% ' & ZZ ;
9380 1 929 ANS QOE WEND ' & ZZ ;
9390 1 930 END;
9400 931
9410 932
9420 933
9430 934 MACRO TERMINATE TRIG PROT < W1 : < 'TERMINATE'(SL)(' ; ') >> ;
9440 1 935 IF TRIM(W1) EQ ' ' THEN WEND = 'END ;' ; ELSE WEND = 'END ;' ;
9450 1 936 ANS QOE ' BEGIN ' & ZZ ;
9460 1 937 ANS QOE ' %CODE ER APORT% ' & ZZ ;
9470 1 938 ANS QOE WEND ' & ZZ ;
9480 1 939 END;
9490 940
9500 941
9510 942
9520 943 MACRO INITTERMIO TRIG PROT < W1 : < 'WRITE' ' ' 'WRITEFLN' >> (SL) W2 : < (OP << (SL) TO (SL) (' ; ') >> , < > (SL) CP) (SL) ' ; ' >> ;
9530 1 944 IF SLVTASKNAME EOC S'NUL' THEN FAIL ;
9540 1 945 ANS QQ & ' BEGIN ' & ZZ ;
9550 1 946 ANS QQ & ' %CODE TS ' & IOINITLOCK & ZZ ;
9560 1 947 ANS QQ & ' %CODE TS ' & IOINITFLAG & ZZ ;
9570 1 948 ANS QQ & ' %CODE C&TSA ' & IOINITFLAG & ZZ ;
9580 1 949 ANS QQ & ' %CODE ER DACT% ' & ZZ ;
9590 1 950 ANS QQ & ' %CODE TS RSCCELL ' & ZZ ;
9600 1 951 ANS QQ & ' END ; ' & ZZ ;
9610 1 952 ANS QQ & W1 & W2 & ZZ ;
9620 1 953 ANS QQ & ' BEGIN ' & ZZ ;
9630 1 954 ANS QQ & ' %CODE C&TSA RSCCELL ' & ZZ ;
9640 1 955 ANS QQ & ' %CODE TS ' & IOTERMLOCK & ZZ ;
9650 1 956 ANS QQ & ' %CODE TS ' & IOTERMFLAG & ZZ ;
9660 1 957 ANS QQ & ' %CODE C&TSA ' & IOTERMFLAG & ZZ ;
9670 1 958 ANS QQ & ' %CODE ER DACT% ' & ZZ ;
9680 1 959 ANS QQ & ' END ; ' & ZZ ;
9690 1 960 END;
9700 961
9710 962
9720 963
9730 964 MACRO LNDPOINT TRIG PROT < 'END.' > ;
9740 1 965 IF SLVTASKNAME EOC S'NUL' THEN FAIL ;
9750 1 966 ANS QOE ' %CODE SZ INTACTNAME' & SLVTASKCNT & ZZ ;
9760 1 967 ANS QOE ' %CODE TS SYSTEMHOLD' & ZZ ;

```



```

9770 1 963 ANS QOE '%CODE' C%TSO SYSTEMHOLD' & Z7 ;
9780 1 969 ANS QOE ' END; ' & Z7 ;
9790 1 971 END;
9800 971 MACRO FLTCHING TRIG PROT<'FETCH'(SL)'STATUS'(SL)'OF'(SL) W1:<'SLAVETASKS'(SL)(';>> >;
9810 1 972 'IF TRIM(W1)[*,,] EQC ';> THEN WEND='END';> ELSE WEND='END';>
9820 1 973 ANS ZZ & ' REGIM ' & ZZ ;
9830 1 974 I = 1;
9840 1 975 WHILE I LE NT
9850 1 976 DO BEGIN
9860 1 977 ANS QOE 'STATUS' & I & 'J' := FALSE ;'&ZZ;
9870 2 978 ANS QOE '%CODE' TNZ IN.TACTNAME'& I &ZZ;
9880 2 979 ANS QOE '%CODE' J '+3 ' &ZZ;
9890 2 980 ANS QOE 'STATUS' & I & 'J' := TRUE ;'&ZZ;
9900 2 981 I = I + 1 ;
9910 2 982 END;
9920 1 983 ANS WEND & ZZ ;
9930 1 984 END;
9940 985
9950 986
9960 987
9970 988 MACRO FORKEY TRIGGER PROTECT <'FOR'(SL) W1:<ID>' '='> >;
9980 1 989 IF SLVTASKNAME NEC S'NUL' THEN FAIL;
9990 1 990 ANS QO & '%CODE' FOR KEYWORD ' & ZZ ;
10000 1 991 ANS ' FOR ' & W1 & ' I = ' ;
10010 1 992 END;
10020 993
10030 994
10040 995
10050 996 MACRO WAITFOREVER TRIGGER PROTECT <'WAIT' W1:<'FOREVER'(';>>>;
10060 1 997 IF TRIM(W1)[*,,] EQC ';> THEN WEND='END';> ELSE WEND='END';>
10070 1 998 ANS ZZ & ' REGIM ' & ZZ;
10080 1 999 ANS QO & '%CODE' TS SYSTEMHOLD' & Z7 ;
10090 1 1000 ANS QO & '%CODE' C%TSO SYSTEMHOLD' & Z7 ;
10100 1 1001 ANS ZZ & WEND ' & ZZ;
10110 1 1002 END;
10120 1003
10130 1004
10140 1005
10150 1006
10160 1007
10170 1008 MACRO CREGION TRIG PROT <'CREGION' ';> >;
10180 1 1009 ANS ZZ & ' BFGIN ' & ZZ ;
10190 1 1010 ANS QO & '%CODE' TS RSCCELL ' & ZZ ;
10200 1 1011 ANS QO & '%CODE' L R15,R13' & ZZ ;
10210 1 1012 ANS QO & '%CODE' L,U R13,1 ' & ZZ ;
10220 1 1013 ANS QO & '%CODE' CREGION ON ' & ZZ ;
10230 1 1014 ANS QO & ' END; ' & ZZ ;
10240 1 1015 END;
10250 1016
10260 1017
10270 1018

```

```

10280      1019
10290      1020
10300      1021
10310      1022      MACRO CEND TRIG PROT <'CEND' ';' > ;
10320      1      1023      ANS QQ E ' BEGIN ' E ZZ ;
10330      1      1024      ANS QQ E '%CODE' C&TS RSCCELL ' E ZZ ;
10340      1      1025      ANS QQ E '%CODE' L W13,R15' E ZZ ;
10350      1      1026      ANS QQ E '%CODE' CRESION OFF ' E ZZ ;
10360      1      1027      ANS QQ E ' END; ' E ZZ ;
10370      1      1028      END;
END MDC 0 ERRORS 0 WARNINGS 0 REMARKS

```



```

590      50      SUSFLAG = 0 ;
600      51      CREGION_FLAG = 0 ;
610      52
620      53
630      54      DECLARE   FORSAV(**);
640      55
650      56
660      57
670      58      MACRO M0 TRIGGER PROTECT < W1:<%(1)>>;
680      1  59          PN_FLAG = 1;
690      1  60          ANSWER S'DELB'&TRIM(W1);
700      1  61      END;
710      62
720      63
730      64
740      65      MACRO M1 TRIGGER PROTECT < '** 'RES' 'SIZE$'>;
750      1  66          PROCTYPE = 'MAINPROG' ;
760      1  67          ANSWER '* RES 136' ;
770      1  68      END;
780      69
790      70
800      71
910      72      MACRO M2 TRIGGER PROTECT < 'LMJ' 'A1,PENTS'>;
820      1  73          IF TRIM(PROCTYPE) NEC 'MAINPROG'
830      1  74              THEN IF SLAVEFLAG EQ 1
840      1  75                  THEN BEGIN
850      1  76                      ANSWER S'EOL'&'          L      AD,AR' & S'EOL';
860      2  77                      SLAVEFLAG = 0 ;
870      2  78                      END;
880      1  79                  ELSE BEGIN
890      1  80                      ANSWER S'EOL'&'          L      AD,RIS' & S'EOL' ;
900      2  81                      ANSWER S'EOL'&'          L      R13,,*AD'& S'EOL' ;
910      2  82                      ANSWER '          T2      FS'&PROCNAME&S'EOL';
920      2  83                      ANSWER '          J      ENTER ' & S'EOL' ;
930      2  84                      ANSWER '          L      X10,XSS'&PROCNAME&S'EOL';
940      2  85                      ANSWER '          J      PPROCST' & S'EOL';
950      2  86                      ANSWER 'ENTER          ' & S'EOL';
960      2  87                  END;
970      1  88          ANSWER '          LMJ      A1,PENTS' ;
980      1  89      END;
990      90
1000     91
1010     92
1020     93      MACRO M3 TRIGGER PROTECT < W1:<ID > '** > ;
1030     1  94          IF PN_FLAG EQ 0 THEN FAIL;
1040     1  95          PROCNAME=TRIM(W1);
1050     1  96          PN_FLAG = 0;
1060     1  97          RETURN S'DELB'& W1&'**';
1070     1  98      END;
1080     99
1090    100

```

```

1100      101  MACRO M4 TRIGGER PROTECT <'STARTUP' W1:<ID>>;
1110      1  102      CALLEDPROC = TRIM(W1);
1120      1  103      DEL_FLAG = 1;
1130      1  104      STARTUP_FLAG=1;
1140      1  105      J=1;
1150      1  106      WHILE J LE 50 DO BEGIN
1160      1  107          DEL(J)=0;
1170      2  108          J=J+1;
1180      2  109      END;
1190      1  110      RETURN ;
1200      1  111  END;
1210      1  112
1220      1  113
1230      1  114
1240      1  115  MACRO M5 TRIGGER PROTECT <'FORK' W1:<ID>>;
1250      1  116      CALLEDPROC = TRIM(W1);
1260      1  117      DEL_FLAG = 1;
1270      1  118      J=1;
1280      1  119      WHILE J LE 50 DO BEGIN
1290      1  120          DEL(J)=0;
1300      2  121          J=J+1;
1310      2  122      END;
1320      1  123      RETURN ;
1330      1  124  END;
1340      1  125
1350      1  126
1360      1  127
1370      1  128
1380      1  129  MACRO M6 TRIGGER PROTECT <'LMJ' ID > ;
1390      1  130      IF DEL_FLAG EQ 0 THEN FAIL;
1400      1  131      ELSE BEGIN
1410      2  132          J=1;
1420      2  133          WHILE J LE 50 DO BEGIN
1430      3  134              DEL(J)=0;
1440      3  135              J=J+1;
1450      2  136          END;
1460      2  137          DEL_FLAG = 0;
1470      2  138          DEL_FLAG2 = 0;
1480      1  139      END;
1490      1  140
1500      1  141
1510      1  142
1520      1  143  MACRO M7 TRIGGER PROTECT <W1:<'L'\L,U'> W2:<ID>>;
1530      1  144      IF DEL_FLAG2 EQ 0 THEN FAIL;
1540      1  145      DELJ=DELJ+1;
1550      1  146      IF DEL(DELJ) EQ 0 THEN RETURN W1C' *W2;
1560      1  147  END;
1570      1  148
1580      1  149
1590      1  150
1600      1  151

```

```

1610      152      MACRO M8 TRIGGER PROTECT <W1:<'S'> W2:<ID>>;
1620      153      IF DEL_FLAG2 EQ 0 THEN FAIL;
1630      154      IF DELJ      EQ 0 THEN FAIL;
1640      155      IF DEL(DELJ) EQ 0 THEN RETURN W1& 'C&W2;
1650      156      END;
1660      157
1670      158
1680      159
1690      160      MACRO M9 TRIGGER PROTECT<'L,H2' 'A0,3,X10'>;
1700      161      IF DEL_FLAG EQ 0 THEN FAIL;
1710      162      IF ( TRIM(PROCTYPE) EQC 'MAINPROG') AND
1720      163      ( STARTUP_FLAG EQ 1 )
1730      164      THEN BEGIN
1740      165          ANSWER '      L,H2      A0,3,X10';
1750      166          STARTUP_FLAG = 0;
1760      167      END;
1770      168      ELSE ANSWER '      L      A0,XSS'& CALLEDPROC ;
1780      169      DEL_FLAG2=1;
1790      170      DELJ=0;
1800      171      END;
1810      172
1820      173
1830      174
1840      175      MACRO M11 TRIGGER PROTECT <'DELETE' 'ARGUMENT' W1:<NUM>>;
1850      176      J=TRIM(W1);
1860      177      DEL(J)=1;
1870      178      END;
1880      179
1890      180
1900      181
1910      182
1920      183      MACRO M12 TRIGGER PROTECT< 'SLAVETASK' 'ENTRY' >;
1930      184      SLAVEFLAG= 1;
1940      185      END;
1950      186
1960      187
1970      188
1980      189      MACRO FORMAC TRIGGER PROTECT< 'FOR' 'KEYWORD' >;
1990      190      FORFLAG = 1 ;      FORNUM = FORNUM + 36 ;
2000      191      IF FORNUM GE 900
2010      192      THEN ANSWER S'EOL'&S'EOL'&' MAXIMUM NUMBER OF FOR STATEMENT ' &
2020      193      'EXCEEDED ' & S'EOL' & S'EOL' ;
2030      194      END;
2040      195
2050      196
2060      197
2070      198
2080      199      MACRO DSHAC TRIGGER PROTECT <'DS' W1:< ID > >;
2090      200      IF FORFLAG NE 1 THEN FAIL;
2100      201      IF TRIM(W1)[1,3] NEC 'A0,' THEN FAIL;
2110      202      FORFLAG = 0 ;

```

```

2120 1 203 FORSAV(TRIM(W1)) = FORNUM ;
2130 1 204 ANSWER S'EOL' & ' L R15,X11 ' &
2140 1 205 S'EOL' & ' L X11,R13 ' &
2150 1 206 S'EOL' & ' S A0,SVRGA0+' & FORNUM & ',X11' &
2160 1 207 S'EOL' & ' S A1,SVRGA1+' & FORNUM & ',X11' &
2170 1 208 S'EOL' & ' L X11,R15' & S'EOL' ;
2180 1 209 END;
2190 210
2200 211
2210 212
2220 213 MACRO DLHAC TRIGGER PROTECT <'DL' W1:< ID > >;
2230 1 214 IF VOID(FORSAV,TRIM(W1)) THEN FAIL ;
2240 1 215 ANSWER S'EOL' & ' L R15,X11 ' &
2250 1 216 S'EOL' & ' L X11,R13 ' &
2260 1 217 S'EOL' & ' L A0,SVRGA0+' & FORSAV(TRIM(W1)) & ',X11' &
2270 1 218 S'EOL' & ' L A1,SVRGA1+' & FORSAV(TRIM(W1)) & ',X11' &
2280 1 219 S'EOL' ;
2290 1 220 END;
2300 221
2310 222
2320 223
2330 224 MACRO STORER13 TRIGGER PROTECT <'S' W1:< ID > >;
2340 1 225 A = TRIM(W1)[1,3];
2350 1 226 B = TRIM(W1)[5,LENGTH(W1)];
2360 1 227 IF A NEC 'R13' THEN FAIL ;
2370 1 228 ANSWER S'EOL' & ' L,U R15,1 ' &
2380 1 229 S'EOL' & ' S R15,' & B & S'EOL';
2390 1 230 END;
2400 231
2410 232
2420 233
2430 234 MACRO SUSKEY TRIGGER <'SUSPEND' 'KEYWORD'>;
2440 1 235 SUSFLAG=1;
2450 1 236 END;
2460 237
2470 238
2480 239
2490 240
2500 241 MACRO SUSKEY2 TRIGGER <'SZ' W1:<ID>>;
2510 1 242 IF SUSFLAG EQ 0 THEN FAIL;
2520 1 243 SUSFLAG = 0;
2530 1 244 ANSWER S'EOL' & ' L R15,A1' & S'EOL';
2540 1 245 ANSWER S'EOL' & ' L A1,' & TRIM(W1) & S'EOL';
2550 1 246 END;
2570 247
2580 248
2590 249
2600 250 MACRO RISIGNORE TRIGGER PROTECT <'L,U' W1:<ID> >;
2610 1 251 IF CREGION_FLAG EQ 0 THEN FAIL;
2620 1 252 IF TRIM(W1)[1,4] NEC 'R15,' THEN FAIL;
2630 1 253 END;

```

```
2680      254  MACRO CREGIONON TRIGGER PROTECT <'CREGION' 'ON' >;
2690      1  255      CREGION_FLAG = 1;
2700      1  256  END;
2710      257
2720      258
2730      259
2740      260
2750      261
2760      262  MACRO CREGIONOFF TRIGGER PROTECT <'CREGION' 'OFF' >;
2770      1  263      CREGION_FLAG = 0;
2780      1  264  END;
END MOC 0 ERRORS 0 WARNINGS 0 REMARKS
```



## APPENDIX E - PROGRAM LISTINGS OF CASE STUDIES

- E.1 - Case Study One: Producer - Consumer Problem
- E.2 - Case Study Two: Dining Philosophers
- E.3 - Case Study Three: Quicksort
  - E.3.1 - Without Processor Dedication
  - E.3.2 - With Processor Dedication
- E.4 - Case Study Four: Gaussian Elimination
  - E.4.1 - Without Processor Dedication
  - E.4.2 - With Processor Dedication
- E.5 - Case Study Five: A Model Operating System

# E.1 - CASE STUDY ONE: Producer - Consumer Problem

```

#CPS 25,S ,CPS S/O,
MACRO INTERPRETER LEVEL 7RT 02/03/84 11:27:36
T *COMPILE OPTIONS=IRSLM,ELTNAME=TPFS,MAIN
T PROGRAM MAIN(INPUT,OUTPUT);
T TYPE LABELS=(RECEIVE,SEND,INIT);
T PROCEDURE BUFFER(VAR ENTRY:LABELS;VAR INP:INTEGER;VAR OUTP:INTEGER);MONITOR;
T PROCEDURE CARDPROC;PROCESS;
T PROCEDURE PRINTPROC;PROCESS;
T VAR RECEIVER : QUEUE ;
T SENDER : QUEUE ;
T LOCKV : QUEUE ;
T ENTRY : LABELS ;
T INP : INTEGER ;
T OUTP : INTEGER ;
T HOLDPERIOD : INTEGER ;
T BEGIN
T HOLDPERIOD := 500;
T ENTRY := INIT;
T STARTUP WITH TASKNAME(T1) BUFFER(ENTRY,INP FORWARD ,OUTP FORWARD);
T COBEGIN(STARTER);
T STARTUP WITH TASKNAME(T2) CARDPROC;
T STARTUP WITH TASKNAME(T3) PRINTPROC;
T COEND;
T SUSPEND(HOLDPERIOD);
T TERMINATE;
T END;
T *COMPILE OPTIONS=IRSLMV,ELTNAME=TPFS,CARD
T PROGRAM DEFINE(CARDPROC);
T TYPE LABELS=(RECEIVE,SEND,INIT);
T PROCEDURE BUFFER(VAR ENTRY:LABELS;VAR INP:INTEGER;VAR OUTP:INTEGER);MONITOR;
T PROCEDURE CARDPROC;
T VAR ENTRY : LABELS ;
T I : INTEGER;
T INP : INTEGER;
T OUTP : INTEGER;
T BEGIN
T RELEASE(CARDPROC);
T AWAIT(STARTER);
T I:=1;
T WHILE TRUE DO
T BEGIN
T INP:=I+1;
T LOCK(LOCKV);
T ENTRY:= SEND;
T FORK WITH TASKNAME(T1) BUFFER(ENTRY,INP,OUTP FORWARD);
T AWAIT(C1);
T I:=I+1;
T END;
T END;
T *COMPILE OPTIONS=IRSLMV,ELTNAME=TPFS,PRINT

```

```

1 PROGRAM DEFINE(PRINTPROC);
1 TYPE LABELS=(RECEIVE,SEND,INIT);
1 PROCEDURE BUFFER(VAR ENTRY:LABELS;VAR INP:INTEGER;VAR OUTP:INTEGER);MONITOR;
1 PROCEDURE PRINTPROC;
1 VAR ENTRY : LABELS ;
1 INP : INTEGER;
1 OUTP : INTEGER;
1 BEGIN
1 RELEASE(PRINTPROC);
1 AWAIT(STARTER);
1 WHILE TRUE DO
1 BEGIN
1 LOCK(LOCKV);
1 ENTRY:= RECEIVE;
1 FORK WITH TASKNAME(P1) BUFFER(ENTRY,INP FORWARD ,OUTP);
1 AWAIT(P1);
1 CREGION; WRITELN(OUTP); CEND;
1 END;
1 END;
1 *COMPILE OPTIONS=IRSLMV,ELTNAME=TPF$,BUFFER
1 PROGRAM DEFINE(BUFFER);
1 TYPE LABELS =(RECEIVE,SEND,INIT);
1 PROCEDURE BUFFER(VAR ENTRY:LABELS;VAR INP:INTEGER;VAR OUTP:INTEGER);
1 VAR FULL : BOOLEAN ;
1 CONTENTS: INTEGER ;
1 BEGIN
1 CASE ENTRY OF
1 RECEIVE:
1 BEGIN
1 UNLOCK(LOCKV);
1 LOCK(RECEIVER);
1 IF NOT FULL THEN WAIT(RECEIVER)
1 ELSE UNLOCK(RECEIVER);
1 OUTP := CONTENTS ;
1 LOCK(SENDER);
1 FULL := FALSE ;
1 SIGNAL(SENDER);
1 QUIT;
1 END;
1 SEND:
1 BEGIN
1 UNLOCK(LOCKV);
1 LOCK(SENDER);
1 IF FULL THEN WAIT(SENDER)
1 ELSE UNLOCK(SENDER);
1 CONTENTS:= INP;
1 LOCK(RECEIVER);
1 FULL := TRUE;
1 SIGNAL(RECEIVER);
1 QUIT;
1 END;
1 END;

```



	35	%CODE	%{(1)		1	23	23
	36						
	37	%CODE	COBEGIN*		1	24	24
	38	%CODE	L AD,('CBG')		1	25	25
	39	%CODE	ER NAMES		1	26	26
	40	%CODE	S AD,CBGNAME		1	27	27
	41						
	42	%CODE	TS CBGNST		1	28	28
	43						
	44	%CODE	C&TSA CBGNST		1	29	29
	45	%CODE	ER DACTS		1	30	30
	46	%CODE	ER EXITS		1	31	31
	47		BEGIN		1		
1	48		BEGIN		1		
2	49	%CODE	S X10,XS\$MAIN		1	32	36
	50	%CODE	L,U AD,057		1	37	37
	51	%CODE	I DO 11, SZ RSCCELL 1+I		1	38	38
	52	%CODE	I DO 11, S,S2 AD,RSCCELL 1+I		1	39	39
	53	%CODE	ER TSQRG\$		1	40	40
	54	%CODE	L,U R13,0		1	41	41
	55		END;		1	42	42
	56		HOLDPERIOD := 500;		1		
	57		ENTRY := INIT;		1	43	45
	58	%CODE	STARTUP BUFFER		1	46	48
	59	%CODE	DELETE ARGUMENT 2		1	49	49
	60	%CODE	DELETE ARGUMENT 3		1	50	50
	61	%CODE	L X10,CX10\$		1	51	51
	62		BUFFER(ENTRY,INP,OUTP);		1	52	52
	63	%CODE	L,U R15,1		1	53	62
	64	%CODE	S R15,0,AD		1	63	63
	65	%CODE	L R15,AD		1	64	64
	66	%CODE	TS TSBUFFER		1	65	65
	67	%CODE	L AD,(000101,BUFFER)		1	66	66
	68	%CODE	ER FORKS		1	67	67
	69	%CODE	C&TSQ TSBUFFER		1	68	68
	70	%CODE	L AD,R15		1	69	69
	71				1	70	70
	72						
2	73		BEGIN		1		
	74						
	75	%CODE	L R15,AD		1	71	71
	76	%CODE	L AD,(000201,COBEGIN)		1	72	72
	77	%CODE	ER FORKS		1	73	73
	78						
	79	%CODE	L AD,R15		1	74	74
	80	%CODE	TS CBGNST		1	75	75
	81	%CODE	C&TSO CBGNST		1	76	76
	82		END;		1		
	83						
	84	%CODE	STARTUP CARDPROC		1	77	77
2	85	%CODE	L X10,CX10\$		1	78	78

	86	CARDPROC;			1	79	82
	87	%CODE	L,U	R15,3	1	83	83
	88	%CODE	S	R15,0,AD	1	84	84
	89	%CODE	L	R15,AD	1	85	85
	90	%CODE	TS	T%CARDPROC	1	86	86
	91	%CODE	L	AD,(000301,CARDPROC)	1	87	87
	92	%CODE	ER	FORKS	1	88	88
	93	%CODE	C%TSQ	T%CARDPROC	1	89	89
	94	%CODE	L	AD,R15	1	90	90
	95						
	96	%CODE		STARTUP PRINTPROC	1	91	91
	97	%CODE	L	X10,CX10S	1	92	92
	98	PRINTPROC;			1	93	96
	99	%CODE	L,U	R15,4	1	97	97
	100	%CODE	S	R15,0,AD	1	98	98
	101	%CODE	L	R15,AD	1	99	99
	102	%CODE	TS	T%PRINTPROC	1	100	100
	103	%CODE	L	AD,(000401,PRINTPROC)	1	101	101
	104	%CODE	ER	FORKS	1	102	102
	105	%CODE	C%TSQ	T%PRINTPROC	1	103	103
	106	%CODE	L	AD,R15	1	104	104
	107						
	108						
2	109	BEGIN			1		
	110						
	111	%CODE	L	R15,AD	1	105	105
	112	%CODE	L	AD,CBGNAME	1	106	106
	113	%CODE	ER	ACTS	1	107	107
	114	%CODE	L	AD,R15	1	108	108
2	115	END;			1		
	116						
	117						
	118	%CODE		SUSPEND KEYWORD	1	109	109
	119						
	120	HOLDPERIOD :=	0 ;		1	110	111
	121						
2	122	BEGIN			1		
	123						
	124	%CODE	ER	TWAITS	1	112	112
	125						
	126	%CODE	L	A1,R15	1	113	113
2	127	END;			1		
	128						
2	129	BEGIN			1		
	130	%CODE	ER	ABORTS	1	114	114
2	131	END;			1		
	132						
1	133	END,			1	115	115

PROGRAM MAIN CONTAINS 133 LINES AND 3 PROCEDURES/3 EXTERNALS

```

INPJT AND OUTPUT ARE ASCII FILES
STORAGE: 2080/5179 TIME USED: 6 SECONDS
END PASCAL 8P1A NO ERRORS NO WARNINGS 116 I-BANK 3072 O-BANK
#CTS,N
CTS 7R1B 28 JUL 83 AT 17:46:43
THE ASSUMED MODE IS FIELDATA
OLD TPF$,PASCAL
COMPILER TYPE: ASM,S
ASSUME ASCII OFF
CALL UNIVAC*CPS.CLEAR
SAVE TPF$,MAIN
YCT$
IN EXEC MODE
3BU*PASCAL'S,IRSLMV
PASCAL 8P1A RL13 THURSDAY, 1983 JULY 28, 17:46:49
1

```

	2	PROGRAM DEFINE(CARDPROC);	:	1				
	3	TYPE LABELS=(RECEIVE,SEND,INIT);	:	1				
	4	PROCEDURE BUFFER(VAR ENTRY:LABELS;VAR INP:INTEGER;VAR OUTP:INTEGER);EXTERNAL ;	:	2				
	5	PROCEDURE CAPDPROC;	:	2				
	6	VAR ENTRY : LABELS ;	:	2				
	7	I : INTEGER;	:	2				
	8	INP : INTEGER;	:	2				
	9	OUTP : INTEGER;	:	2				
1	10	BEGIN	:	2				
2	11	BEGIN	:	2				
	12	%CODE SZ F%CARDPROC	:	2	0		4	
	13	%CODE S X10,XS%CARDPROC	:	2	5		5	
	14	%CODE PROCST	:	2	6		6	
2	15	END;	:	2	7		7	
	16		:	2				
2	17	BEGIN	:	2				
	18	%CODE TS T%CARDPROC	:	2	8		8	
2	19	%CODE C* TSA T%CARDPROC	:	2	9		9	
	20	END;	:	2				
	21		:	2				
2	22	BEGIN	:	2				
	23	%CODE L R15,AN	:	2	10		10	
	24	%CODE LA AN,(4)	:	2	11		11	
	25	%CODE EP AWAIT\$	:	2	12		12	
	26	%CODE L AN,P15	:	2	13		13	
2	27	END;	:	2				
	28		:	2				
	29	I:=1;	:	2	14		15	
	30	WHILE TRUE DO	:	2	16		18	
2	31	BEGIN	:	2				
	32	INP:=I*1;	:	2	19		23	
	33		:	2				
3	34	BEGIN	:	2				

	35	%CODE	TS	LOCKV	2	24	24
	36	END;			2		
	37						
	38		ENTRY:= SEND;		2	25	26
	39	%CODE	FORK	BUFFER	2	27	27
	40	%CODE	DELETE	ARGUMENT	2	28	28
	41		BUFFER(ENTRY,INP,OUTP);		2	29	38
	42	%CODE	L,U	R15,5	2	39	39
	43	%CODE	S	P15,0,AU	2	40	40
	44	%CODE	L	P15,A0	2	41	41
	45	%CODE	L	A0,(000501,BUFFER)	2	42	42
	46	%CODE	ER	FORK	2	43	43
	47	%CODE	L	AU,R15	2	44	44
	48						
3	49	BEGIN			2		
	50	%CODE	L	R15,A0	2	45	45
	51	%CODE	LA	A0,(32)	2	46	46
	52	%CODE	EF	AWAIT	2	47	47
	53	%CODE	L	A0,R15	2	48	48
3	54	END;			2		
	55						
	56		I:=I+1;		2	49	52
2	57	END;			2	53	53
1	58	END;			2	54	54

PROGRAM DEFINE CONTAINS 58 LINES AND 2 PROCEDURES/2 EXTERNALS  
NO MAIN PROGRAM INPUT AND OUTPUT ARE ASCII FILES  
STORAGE: 2153/5179 TIME USED: 3015 MILLISECONDS  
END PASCAL BPIA NO ERRORS NO WARNINGS 55 I-BANK  
ACTS,4  
OLD TPF%PA%CODE  
ASSUME ASCII OFF  
CALL UNIVAC%CP%:CLEAR  
SAVE TPF%PCARD  
XCTS  
IN EXEC MODE  
@BU@PASCAL%S,IRSLMV  
PASCAL BPIA PLTB THURSDAY, 1983 JULY 28, 17:46:54  
1

	2	PROGRAM	OFFINE	(PRINTPROC);	1		135
	3	TYPE	LABELS=(RECEIVE,SEND,INIT);		1		
	4	PROCEDURE	PUFFER(VAR ENTRY:LABELS;VAR INP:INTEGER;VAR OUTP:INTEGER);EXTERNAL ;		2	0	7
	5	PROCEDURE	PRINTPROC;		2	0	4
	6	VAR	ENTRY : LABELS ;		2	5	5
	7	INP	: INTEGER;		2	6	6
	8	OUTP	: INTEGER;		2	7	7
1	9	BEGIN			2		
2	10	BEGIN			2		



	11	%CODE	SZ	F%PRINTPROC	2	5	5
	12	%CODE	S	X10,XS%PRINTPROC	2	6	6
	13	%CODE	PROCST		2	7	7
2	14	END;			2		
	15						
2	16	BEGIN			2		
	17	%CODE	TS	T%PRINTPROC	2	8	8
	18	%CODE	C%TSA	T%PRINTPROC	2	9	9
2	19	END;			2		
	20						
2	21	BEGIN			2		
	22	%CODE	L	R15,AD	2	10	10
	23	%CODE	LA	AD,(4)	2	11	11
	24	%CODE	EP	AWAIT%	2	12	12
	25	%CODE	L	AD,R15	2	13	13
2	26	END;			2		
	27						
2	28	WHILE TRUE DO			2	14	16
	29	BEGIN			2		
	30						
3	31	BEGIN			2		
	32	%CODE	TS	LOCKV	2	17	17
3	33	END;			2		
	34						
	35	ENTRY:= RECEIVE;			2	18	19
	36	%CODE	FORK	BUFFER	2	20	20
	37	%CODE	DELETE	ARGUMENT 2	2	21	21
	38	BUFFER(ENTRY,INP,OUTP);			2	22	31
	39	%CODE	L,U	R15,6	2	32	32
	40	%CODE	S	R15,0,AD	2	33	33
	41	%CODE	L	R15,AD	2	34	34
	42	%CODE	L	AD,(000601,BUFFER)	2	35	35
	43	%CODE	ER	FORK%	2	36	36
	44	%CODE	L	AD,R15	2	37	37
	45						
3	46	BEGIN			2		
	47	%CODE	L	R15,AD	2	38	38
	48	%CODE	LA	AD,(64)	2	39	39
	49	%CODE	EP	AWAIT%	2	40	40
	50	%CODE	L	AD,R15	2	41	41
3	51	END;			2		
	52						
	53						
3	54	BEGIN			2		
	55	%CODE	TS	RSCCELL	2	42	42
	56	%CODE	L	R15,R13	2	43	43
	57	%CODE	L,U	R13,1	2	44	44
	58	%CODE	C%REGION	ON	2	45	45
3	59	END;			2		
	60	WRITELN(OUTP); BEGIN			2	46	50
3	61	%CODE	C%TS	RSCCELL	2	51	51

3	62	%CODE	L	R13,R15	2	52	52
	63	%CODE		CREGION OFF	2	53	53
	64	END;			2		
	65						
2	66	END;			2	54	54
1	67	END;			2	55	55

```

PROGRAM DEFINE CONTAINS 67 LINES AND 2 PROCEDURES/2 EXTERNALS
NO MAIN PROGRAM INPUT AND OUTPUT ARE ASCII FILES
STORAGE: 2147/5179 TIME USED: 3283 MILLISECONDS
END PASCAL BR1A NO ERRORS NO WARNINGS 56 T BANK
ACTS,N
OLD TPF%:PASCAL
ASSUME ASCII OFF
CALL UNIVAC*CPS*CLEAR
SAVE TPF%:PRINT
XCTS
IN EXEC MODE
@BU*PASCAL'S,IRSLMV
PASCAL BR1A PL18 THURSDAY, 1983 JULY 28, 17:46:58
1

```

1	2	PROGRAM DEFINE(BUFFER);			1		
2	3	TYPE LABELS =(RECEIVE,SEND,INIT);			1		
	4	PROCEDURE BUFFER(VAR ENTRY:LABELS;VAR INP:INTEGER;VAR OUTP:INTEGER);			2		
	5	VAR FULL : BOOLEAN ;			2		
	6	CONTENTS: INTEGER ;			2		
1	7	BEGIN			2		
2	8	BEGIN			2		
	9	%CODE SZ F%BUFFER			2	0	4
	10	%CODE S X10,XS%BUFFER			2	5	5
	11	%CODE PROCST			2	6	6
2	12	END;			2	7	7
	13	CASE ENTRY OF			2	8	11
3	14	RECEIVE:			2		
	15	BEGIN			2		
	16						
4	17	BEGIN			2		
	18	%CODE C%TS LOCKV			2	12	12
4	19	END;			2		
	20						
	21						
4	22	BEGIN			2		
	23	%CODE TS RECEIVER			2	13	13
4	24	END;			2		
	25						
	26	IF NOT FULL THEN			2	14	16
4	27	BEGIN			2		
	28	%CODE C%TSQ RECEIVER			2	17	17

```

0 135
0 7
0 8
0 9

```

4	29	END			2		
	30		ELSE		2		
4	31	BEGIN			2	18	18
	32	%CODE	C*TS	RCFIVER	2	19	19
4	33	END;			2		
	34						
	35		OUTP := CONTENTS ;		2	20	23
	36						
4	37	BEGIN			2		
	38	%CODE	TS	SENDER	2	24	24
4	39	END;			2		
	40						
	41		FULL := FALSE ;		2	25	26
	42						
4	43	BEGIN			2		
	44	%CODE	C*TSA	SENDER	2	27	27
4	45	END;			2		
	46						
	47						
4	48	BEGIN			2		
	49						
	50	%CODE	EP	EXIT*	2	28	28
4	51	END;			2		
	52						
3	53	END;			2	29	29
	54	SEND:			2		
3	55	REGIN			2		
	56						
4	57	BEGIN			2		
	58	%CODE	C*TS	LOCKV	2	30	30
4	59	END;			2		
	60						
	61						
4	62	BEGIN			2		
	63	%CODE	TS	SENDER	2	31	31
4	64	END;			2		
	65						
	66		IF FULL THEN		2	32	34
4	67	BEGIN			2		
	68	%CODE	C*TSQ	SENDER	2	35	35
4	69	END			2		
	70		ELSE		2		
4	71	BEGIN			2	36	36
	72	%CODE	C*TS	SENDER	2	37	37
4	73	END;			2		
	74						
	75		CONTENTS := INP;		2	38	41
	76						
4	77	BEGIN			2		
	78	%CODE	TS	RCFIVER	2	42	42
4	79	END;			2		

	87							
	87	FULL := TRUE;			2	43	44	
	88							
4	83	BEGIN			2			
	84	CODE	CATSA	RECEIVER	2	45	45	
4	85	END;			2			
	86							
	87							
4	88	BEGIN			2			
	89							
	90	CODE	ER	EXIT	2	46	46	
4	91	END;			2			
	92							
3	93	END;			2	47	47	
	94	INIT:			2			
3	95	BEGIN			2			
	96	FULL := FALSE;			2	48	49	
	97							
4	98	BEGIN			2			
	99	CODE	TS	TSBUFFER	2	50	50	
	100	CODE	CATSA	TSBUFFER	2	51	51	
4	101	END;			2			
	102							
	103							
4	104	BEGIN			2			
	105	CODE	TS	SYSTEMHOLD	2	52	52	
	106	CODE	CATSA	SYSTEMHOLD	2	53	53	
	107							
4	108	END;			2			
	109							
3	110	END;			2	54	54	
2	111	END;			2	55	62	
1	112	END;			2	63	63	

PROGRAM DEFINE CONTAINS 112 LINES AND 1 PROCEDURE/1 EXTERNAL  
NO MAIN PROGRAM INPUT AND OUTPUT ARE ASCII FILES  
STORAGE: 2142/5179 TIME USED: 4365 MILLISECONDS  
END PASCAL BR1A NO ERRORS NO WARNINGS 64 T\_BANK  
ACTS,N  
OLD TPF%PA%CODE  
ASSUME ASCII OFF  
CALL UNIVAC+CPS,CLEAR  
SAVE TPF%,BUFFER  
XCTS  
IN EXEC MODE  
@CPS%S,N TPF%.MAIN,CPS.S/3,TPF%.PA%CODE  
MACRO INTERPPETER LEVEL 7R1 07/28/83 17:47:06  
@ELT,IN TPF%.MAIN  
ELT BR1 S7401C 07/28/83 17:47:16 (70)

END FLT. ERRORS: NONE. TIME: 2.057 SEC. IMAGE COUNT: 131  
 \*ASM,US TPF1,MAIN  
 ASM 1SR1 573010 07/28/93 17:47:18 (0,1)

1.						AXR* R00715	
2. U						PCONV* LOU ASCPOAS0*	
3.						ASCIO\$* EQU 1	
4.						PSL\$* FQU 77	
5.						\$(30),K\$11*	
6.						SIZE** EQU 3072	
7.						START\$* RES 176	
8.	36	000000				R\$CELL* T\$CELL	
9.		000210	00	57	00	CB\$NST* T\$CELL	
10.		000211	00	57	00	SYSTEMHOLD* T\$CELL	
11.		000212	00	57	00	LOCKV* T\$CELL	
12.		000213	00	57	00	RECEIVER* T\$CELL	
13.		000214	00	57	00	SENDER* T\$CELL	
14.		000215	00	57	00	STOPPER* T\$CELL	
15.		000216	00	57	00	T\$BUFFER* T\$CELL	
16.		000217	00	57	00	T\$CARDPROC* T\$CELL	
17.		000220	00	57	00	T\$MAIN* T\$CELL	
18.		000221	00	57	00	T\$PRINTPROC* T\$CELL	
19.		000222	00	57	00	X\$BUFFER* RES 1	
20.		000223				X\$CARDPROC* RES 1	
21.		000224				X\$MAIN* RES 1	
22.		000225				X\$PRINTPROC* RES 1	
23.		000226				CB\$NAME* RES 1	
24.		000227				F\$BUFFER* RES 1	
25.		000230				F\$CARDPROC* RES 1	
26.		000231				F\$MAIN* RES 1	
27.		000232				F\$PRINTPROC* RES 1	
28.		000233				SVRG00* RES 900	
29.		000234				SVRG01* RES 900	
30.		002040				RES SIZE* 136,1820	
31.		003644				\$(1)	
32.						CO\$BEGIN*	
33.		000000	27	00	14	L AD,(*CB\$*)	
34. U	01	000001	72	11	00	EP NAME\$	
35.		000002	06	00	14	S AD,CB\$NAME	
36.		000003	73	17	00	TS CB\$NST	
37.		000004	05	15	00	C\$TSA CB\$NST	
		000005	50	01	00		
		000006	72	11	00		
38. U		000007	72	11	00	EP DACT\$	
39. U		000010	72	11	00	EP EXIT\$	
40.							
41.						\$(1)	
42.						MAIN**	
43. U		000011	74	13	15	L\$J A1,PENT\$	
44.		000012	003650	000210		*MAS,136	

0000000000000001  
\*MAIN

S X10,XS,MATN  
L,U AD,057  
I NO 11, SZ RSCCELL,1+1

I NO 11, S,S2 AD,RSCCELL,1+1

000013 0000000000000001  
000014 220616230505  
000015 050505050505  
000016 06 00 12 00 0 000225  
000017 27 16 14 00 000057

000020 05 00 00 00 0 000210  
000021 05 00 00 00 0 000211  
000022 05 00 00 00 0 000212  
000023 05 00 00 00 0 000213  
000024 05 00 00 00 0 000214  
000025 05 00 00 00 0 000215  
000026 05 00 00 00 0 000216  
000027 05 00 00 00 0 000217  
000030 05 00 00 00 0 000220  
000031 05 00 00 00 0 000221  
000032 05 00 00 00 0 000222

000033 06 14 14 00 0 000210  
000034 06 14 14 00 0 000211  
000035 06 14 14 00 0 000212  
000036 06 14 14 00 0 000213  
000037 06 14 14 00 0 000214  
000040 06 14 14 00 0 000215  
000041 06 14 14 00 0 000216  
000042 06 14 14 00 0 000217  
000043 06 14 14 00 0 000220  
000044 06 14 14 00 0 000221  
000045 06 14 14 00 0 000222  
000046 72 11 00 00 0 000000  
000047 23 16 15 00 000000  
000050 23 16 17 00 000070  
000051 27 16 14 00 000764  
000052 06 00 14 00 0 003645  
000053 23 16 17 00 000071  
000054 27 16 14 00 000002  
000055 06 00 14 00 0 003644

000056 27 00 12 00 0 000000  
000057 23 16 17 00 000076  
000060 27 01 14 12 0 000003  
000061 27 16 01 00 003644  
000062 06 00 01 14 0 000005

000063 27 16 11 00 000000

45.  
46.

47.  
48.  
49.

50.

51. U

52.

53.

54.

55.

56.

57.

58.

59.

60.

61. U

62.

63.

64.

65.

66.

67.

68.

69.

70.

71.

72.

EP TSOR3\$  
L,U R13,0  
L,U R15,56  
L,U AD,500  
S AU,STARTS+1957  
L,U R15,57  
L,U AU,2  
S AU,STARTS+1956

L X10,CX10F  
L,U R15,62  
L,U L,HZ AD,3,X10  
L,U X1,STARTS+1956  
S X1,5,AD

L,U X9,STARTS

```

73.      U      000064 23 16 17 00 000001
74.      U      000065 04 00 14 00 000000
75.      U      000065 23 00 17 00 000014
76.      U      000067 73 17 00 00 000217
77.      U      000070 27 00 14 00 000001
78.      U      000071 72 11 00 00 000000
79.      U      000072 74 06 00 00 000217
80.      U      000073 72 11 00 00 000000
81.      U      000074 27 00 14 00 000117
82.      U      000075 23 00 17 00 000014
83.      U      000076 27 00 14 00 000002
84.      U      000077 72 11 00 00 000000
85.      U      000100 27 00 14 00 000117
86.      U      000101 73 17 00 00 000211
87.      U      000102 74 06 00 00 000211
88.      U      000103 72 11 00 00 000000
89.      U      000104 27 00 12 00 000000
90.      U      000105 23 16 17 00 000126
91.      U      000106 27 01 14 12 000003
92.      U      000107 27 16 11 00 000000
93.      U      000110 23 16 17 00 000003
94.      U      000111 04 00 17 14 00 000000
95.      U      000112 23 00 17 00 000014
96.      U      000113 73 17 00 00 000220
97.      U      000114 27 00 14 00 000003
98.      U      000115 72 11 00 00 000000
99.      U      000116 74 06 00 00 000220
100.      U      000117 72 11 00 00 000000
101.      U      000120 27 00 14 00 000117
102.      U      000121 27 00 12 00 000000
103.      U      000122 23 16 17 00 000142
104.      U      000123 27 01 14 12 000003
105.      U      000124 27 16 11 00 000000
106.      U      000125 23 16 17 00 000004
107.      U      000126 04 00 17 14 00 000000
108.      U      000127 23 00 17 00 000014
109.      U      000130 73 17 00 00 000222
110.      U      000131 27 00 14 00 000004
111.      U      000132 72 11 00 00 000000
112.      U      000133 74 06 00 00 000222
113.      U      000134 72 11 00 00 000000
114.      U      000135 27 00 14 00 000117
115.      U      000136 23 00 17 00 000014
116.      U      000137 27 00 14 00 000227
117.      U      000140 72 11 00 00 000000
118.      U      000141 27 00 14 00 000117
119.      U      000141 27 00 14 00 000117

L,U      R15,1
S      R15,0,AD
      R15,AD
TS      T$BUFFER
L      AN,(U000101,BUFFER)
FR      FORK$
CSTSQ  T$BUFFER

L      AU,R15
L      R15,AD
L      AU,(U000201,C08BEGIN)
FR      FORK$
L      AD,R15
TS      CBNST
CSTSQ  CBNST

L      X10,CX10$
L,U      R15,R5
L,HZ      AD,3,X10
L,U      X9,START$

L,U      R15,3
S      R15,0,AD
L      R15,AD
TS      T$CARDPROC
L      AN,(U000301,CARDPROC)
FR      FORK$
CSTSQ  T$CARDPROC

L      AU,R15

L      X10,CX10$
L,U      R15,98
L,HZ      AD,3,X10
L,U      X9,START$

L,U      R15,4
S      R15,0,AD
L      R15,AD
TS      T$PRINTPROC
L      AN,(U000401,PRINTPROC)
FR      FORK$
CSTSQ  T$PRINTPROC

L      AU,R15
L      R15,AD
L      AO,C3$NAME
FR      ACT$
L      AO,R15

```

```

120.      000142  23 16 17 00 000170      L,U      R15,120
121.
122.      000143  23 00 17 00 0 000015      L      R15,A1
123.
124.      000144  27 00 15 00 0 003645      L      A1,START*+1957
125.
126. U      000145  72 11 00 00 0 000000      ER      TWAITS
127.      000146  27 00 15 00 0 000117      L      A1,R15
128. U      000147  72 11 00 00 0 000000      EP      AROPTS
129. U      000150  74 04 00 00 0 000000      J      PEND*
130.      000000003650      MAT EQU 1950
131.      00      000000      0000000000714      END
          000001      000101      000000
          000002      000201      000000
          000003      000301      000000
          000004      000401      000000

```

UNDEFINED SYMBOLS:

PEND*	ABORT*	TWAITS	ACT*	PRINTPROC	CARDPROC	CTS*	FORK*	BUFFER	CX10*	TSOR6*
PENT*	FXTT*	DACT*	NAME*	CTSA*	CTSS*	ASCFDASC*				

```

END ASM, ERRORS : NONE      ITEM COUNT: 131
BCPS*,S,N      TPF*,CARD,CPS,S/3,TPF*,PA*CODE
MACRO INTERPRETER LEVEL 7R1 07/28/83 17:47:24
@ELT,IN      TPF*,CARD
FLT BR1 S7401C 07/28/83 17:47:30 (->0)

```

```

END ELT, ERRORS: NONE; TIME: 10.696 SEC; IMAGE COUNT: 78
BAS*,US      TPF*,CARD
ASM 15R1 S7301D 07/28/83 17:47:31 (U,1)

```

```

1,
2, U      000000000000      AXRS 800715
3,
4,
5,
6,
7,      01 000000      27 00 14 00 0 000117      L      A0,R15
8,
9,      000001      23 00 15 14 2 000000      L      R13,*,A0
10. U      000002      50 00 00 00 0 000000      TZ      F*CARDPROC
11,      000003      74 04 00 00 0 000006      J      ENTER
12, U      000004      27 00 17 00 0 000000      L      X10,XS*CAPDPROC
13,      000005      74 04 00 00 0 000015      J      PROCST
14,
15. U      000006      74 13 15 00 0 000000      LMJ      A1,PENT*
16,      000007      000011      000000      +D1*,5
17,      000010      000000000000      000000000000      0
18,      000011      100627112527      *CARDPROC

```



19.	U	000012	24	00	05	05	05	05	05	00	U	000000	SZ	F%CARDPROC
20.	U	000013	05	00	00	00	00	00	00	00	U	000000	S	X10,XS%CARDPROC
21.		000014	06	00	12	00	00	00	00	00	U	000000	PROCST	
22.	U	000015	73	17	00	00	00	00	00	00	U	000000	TS	T%CARDPROC
23.	U	000016	05	15	00	00	00	00	00	00	U	000000	C%TSA	T%CARDPROC
		000017	50	01	00	00	00	00	00	00	U	000000		
		000017	72	11	00	00	00	00	00	00	U	000000		
		000021	23	00	17	00	00	00	00	00	U	000014	L	R15,A0
24.		000022	10	00	00	00	00	00	00	00	U	000000	LA	A0,(4)
25.		000023	72	11	00	00	00	00	00	00	U	000000	EP	AWAIT%
26.	U	000024	27	00	14	00	00	00	00	00	U	000017	L	A0,P15
27.		000025	23	16	17	00	00	00	00	00	U	000035	L,U	R15,29
28.														
29.														
30.		000026	23	16	17	00	00	00	00	00	U	000001	L,U	R15,1
31.		000027	04	00	17	12	00	00	00	00	U	000006	S	R15,6,X10
32.														
33.													ZUS	
34.		000030	23	16	17	00	00	00	00	00	U	000036	L,U	R15,30
35.		000031	74	04	00	00	00	00	00	00	U	000033	J	%+2
36.		000032	74	04	00	00	00	00	00	00	U	000072	J	ZIF
37.		000033	23	16	17	00	00	00	00	00	U	000040	L,U	R15,32
38.		000034	27	00	14	12	00	00	00	00	U	000006	L	A0,6,X10
39.		000035	31	16	00	00	00	00	00	00	U	000001	MST,U	A0,1
40.		000036	24	16	14	00	00	00	00	00	U	000000	A,U	A0,0
41.		000037	06	00	14	12	00	00	00	00	U	000007	S	A0,7,X10
42.	U	000040	73	17	00	00	00	00	00	00	U	000000	TS	LOCKV
43.		000041	23	16	17	00	00	00	00	00	U	000046	L,U	R15,38
44.														
45.		000042	23	16	17	00	00	00	00	00	U	000001	L,U	R15,1
46.		000043	04	00	17	12	00	00	00	00	U	000005	S	R15,5,X10
47.														
48.														
49.														
50.		000044	23	16	17	00	00	00	00	00	U	000051	L,U	R15,41
51.	U	000045	27	00	14	00	00	00	00	00	U	000000	L	A0,XS%BUFFER
52.		000046	27	16	01	12	00	00	00	00	U	000005	L,U	X1,5,X10
53.		000047	06	00	01	14	00	00	00	00	U	000005	S	X1,5,A0
54.		000050	27	16	01	12	00	00	00	00	U	000007	L,U	X1,7,X10
55.		000051	06	00	01	14	00	00	00	00	U	000006	S	X1,6,A0
56.														
57.														
58.	U	000052	27	16	11	00	00	00	00	00	U	000000	L,U	X9,START%
59.														
60.		000053	23	16	17	00	00	00	00	00	U	000005	L,U	R15,5
61.		000054	04	00	17	14	00	00	00	00	U	000000	S	R15,0,A0
62.		000055	23	00	17	00	00	00	00	00	U	000014	L	R15,A0
63.	U	000056	27	00	14	00	00	00	00	00	U	000001	L	A0,(0000501,BUFFER)
64.	U	000057	72	11	00	00	00	00	00	00	U	000000	ER	FOPK%
65.		000060	27	00	14	00	00	00	00	00	U	000017	L	A0,R15
66.		000061	23	00	17	00	00	00	00	00	U	000014	L	R15,A0

67.	000062	10 00 00 00 0 000002	LA	AD,(32)
58. U	000063	72 11 00 00 0 000000	LP	AWAIT\$
59.	000064	27 00 14 00 0 000017	L	AD,R15
70.	000065	23 16 17 00 000070	L,U	R15,56
71.	000065	27 00 14 12 0 000006	L	AD,6,X10
72.	000067	24 16 14 00 000001	A,U	AD,1
73.	000070	06 00 14 12 0 000006	S	AD,6,X10
74.	000071	74 04 00 00 0 000030	J	ZU\$
75.			Z1\$	
76. U	000072	74 04 00 00 0 000000	J	PRET\$
77.		0000000000011	01\$ EQU 0	
78.			END	

00	000000	000000000004
	000001	000001 000000
	000002	000000000040

UNDEFINED SYMBOLS:

PRET\$	FORK\$	RUFFER	START\$	XS\$BUFFER	LOCKV	AWAIT\$	T\$CARDPROC	PENT\$	XS\$CARDPROC	F\$CARDPROC
CTSA\$	CT\$	ASCFDASC\$								

END ASM, ERRORS : NONE ITEM COUNT: 78  
@CPS\$.N TPF\$.PRINT,CPS\$.S/3,TPF\$.PA\$CODE  
MACRO INTERPRETER LEVEL 7R1 07/28/83 17:47:35  
@ELT,IN TPF\$.PRINT  
ELT BR1 S74010 07/28/83 17:47:42 (END)

END ELT, ERRORS: NONE TIME: 1.707 SEC\$ IMAGE COUNT: 73  
@ASM,US TPF\$.PRINT  
ASM 15R1 S73010 07/28/83 17:47:43 (0,1)

1.					AXR\$ 800715
2. U		000000000000			PCONV\$ EQU ASCFDASC\$
3.					
4.					\$ (1)
5.					PRINTPROC*
6.					
7.	01	000000	27 00 14 00 0 000017	L	AD,R15
8.					
9.		000001	23 00 15 14 2 000000	L	R13,;*AD
10. U		000002	50 00 00 00 0 000000	TZ	F\$PRINTPROC
11.		000003	74 04 00 00 0 000006	J	ENTER
12. U		000004	27 00 12 00 0 000000	L	X10,XS\$PRINTPROC
13.		000005	74 04 00 00 0 000015	J	PROCST
14.				ENTER	
15. U		000006	74 13 15 00 0 000000	LMJ	A1,PENT\$
16.		000007	000010 000005	+DIS,5	
17.		000010	000000000000		U
18.		000011	252716233125	*PRINTPROC	
		000012	272410050505		
19. U		000013	05 00 00 00 0 000000	SZ	F\$PRINTPROC

```

20. U      J00014  06  00  17  00  0  000000
21. U      J00015  73  17  00  00  0  000000
22. U      J00016  05  15  00  00  0  000000
23. U      J00017  50  01  00  00  0  000000
24. U      J00020  72  11  00  00  0  000000
25. U      J00021  23  00  17  00  0  000000
26. U      J00022  10  00  00  00  0  000000
27. U      J00023  72  11  00  00  0  000000
28. U      J00024  27  00  14  00  0  000117
29. U      J00025  23  16  17  00  00  000134
30. U      J00026  74  04  00  00  0  000030
31. U      J00027  74  04  00  00  0  000070
32. U      J00030  73  17  00  00  0  000000
33. U      J00031  23  16  17  00  00  000043
34. U      J00032  05  00  00  12  0  000005
35. U
36. U
37. U      J00033  23  16  17  00  00  000046
38. U      J00034  27  00  14  00  0  000000
39. U      J00035  27  16  01  12  0  000005
40. U      J00036  06  00  01  14  0  000005
41. U
42. U
43. U      J00037  27  16  01  12  0  000007
44. U      J00040  06  00  01  14  0  000007
45. U      J00041  27  16  11  00  000000
46. U
47. U      J00042  23  16  17  00  000006
48. U      J00043  04  00  17  14  0  000000
49. U      J00044  23  00  17  00  0  000014
50. U      J00045  27  00  14  00  0  000000
51. U      J00046  72  11  00  00  0  000000
52. U      J00047  27  00  14  00  0  000117
53. U      J00050  23  00  17  00  0  000014
54. U      J00051  10  00  00  00  0  000002
55. U      J00052  72  11  00  00  0  000000
56. U      J00053  27  00  14  00  0  000117
57. U      J00054  73  17  00  00  0  000000
58. U      J00055  23  00  17  00  0  000115
59. U      J00056  23  16  15  00  000001
60. U
61. U      J00057  27  16  14  00  000013
62. U
63. U      J00060  27  00  15  12  0  000007
64. U      J00061  74  13  13  00  0  000000
65. U      J00062  74  13  13  00  0  000000
66. U      J00063  05  15  00  00  0  000000
67. U      J00064  50  01  00  00  0  000000
68. U      J00065  72  11  00  00  0  000000

```

X10,XS\$PRINTPROC  
PROCESS S  
TS TS  
C\$TSA TS\$PRINTPROC  
TS\$PRINTPROC  
L R15,A0  
LA AN,(4)  
EP AWAIT\$  
L AN,R15  
Z01 L  
L,U R15,29  
J 8+2  
J Z1\$  
TS LOCKV  
L,U R15,35  
S2 5,Y40  
L,U R15,38  
L AN,XS\$BUFFER  
L,U X1,5,X10  
S X1,5,AU  
L,U X1,7,X10  
S X1,7,A0  
L,U X9,START\$  
L,U R15,6  
S R15,0,A0  
L R15,A0  
L AN,(UNDS01,BUFFER)  
FR FORK\$  
L AU,R15  
L R15,A0  
LA AN,(54)  
EP AWAIT\$  
L AN,R15  
TS R\$CELL  
L R15,R13  
L,U R13,1  
L,U AU,START\$+7\$  
L R1,7,X10  
LMJ X11,PWR\$  
LMJ X11,PLN\$  
C\$TS R\$CELL

```

57.      000065 23 00 15 00 0 0001,7      L      R13,R15
58.
59.      000067 74 04 00 00 0 000025      J      Z01
70.
71. U    000077 74 04 00 00 0 000000      J      PRET1
72.      00000000000000000000000000000000  D11 EQU A
73.
      00  000000 0000000000000000
      000001 000601 000000
      000002 000000000000

```

```

UNDEFINED SYMBOLS:
PRET1  PWN11  PWR1  RSCCELL  FORK1  BUFFER  START1  XS1BUFFER  LOCKV  AWAITS  TSPRINTPROC
PENT1  YS1PRINTPROC1PRINTPROC CISA1  CTS1  ASCFDASC1

```

```

END ASM, ERRORS : NONE ITEM COUNT: 73
BCPS:5,N TPF1,BUFFER,CPS:5/3,TPF1,PASCODE
MACRO INTERPRETER LEVEL 7R1 07/28/83 17:47:47
BELT,IN TPF1,BUFFER
FLT 8R1 S7401C 07/28/83 17:47:52 (F>0)

```

```

END ELT, ERRORS: NONE TIME: 1.825 SEC. IMAGE COUNT: 92
@ASM,US TPF1,BUFFER
ASM 1SR1 S73010 07/28/83 17:47:54 (0,1)

```

```

1.
2. U    0000000000000000      AXR1 R00715
3.
4.
5.
6.
7. 01  000000 27 00 14 00 0 000117      L      A0,R15
8.
9.      000001 23 00 15 14 2 000000      L      R13,,*A0
10. U   000002 50 00 00 00 0 000000      IZ     F1BUFFER
11.      000003 74 04 00 00 0 000006      J      ENTER
12. U   000004 27 00 12 00 0 000000      L      X10,XS1BUFFER
13.      000005 74 04 00 00 0 000015      J      PROCST
14.
15. U   000006 74 13 15 00 0 000000      ENTER  LMJ    A1,PENT1
16.      000007 000012 000010      +D01,8
17.      000010 00000000000000000000000000  0
18.      000011 073213131227      *BUFFER
      000012 050505050505
19. U   000013 05 00 00 00 0 000000      S7     F1BUFFER
20. U   000014 06 00 12 00 0 000000      S      X10,XS1BUFFER
21.
22.      000015 23 16 17 00 000015      PROCST L,U     R15,13
23.      000016 27 01 13 12 0 000005      L,H2   X11,5,X10
24.      000017 27 00 14 13 0 000000      L      A0,0,X11

```

25, 000020 74 04 00 00 0 000126  
 26, 000021 05 15 00 00 0 000000  
 27, U 000022 50 01 00 00 0 000000  
 000023 72 11 00 00 0 000000  
 28, U 000024 73 17 00 00 0 000000  
 29, 000025 23 16 17 00 000032  
 30, 000026 50 00 00 12 0 000010  
 31, 000027 74 04 00 00 0 000033  
 32, U 000030 74 06 00 00 0 000000  
 U 000031 72 11 00 00 0 000000  
 33, 000032 74 04 00 00 0 000036  
 34,  
 35, U 000033 05 15 00 00 0 000000  
 000034 50 01 00 00 0 000000  
 000035 72 11 00 00 0 000000  
 36,  
 37, 000036 23 16 17 00 000043  
 38, 000037 27 00 14 12 0 000011  
 39, 000040 27 01 13 12 0 000007  
 40, 000041 06 00 14 13 0 000000  
 41, U 000042 73 17 00 00 0 000000  
 42, 000043 23 16 17 00 000051  
 43, 000044 05 00 00 12 0 000010  
 44, U 000045 05 15 00 00 0 000000  
 000046 50 01 00 00 0 000000  
 000047 72 11 00 00 0 000000  
 45, U 000050 72 11 00 00 0 000000  
 46, 000051 74 04 00 00 0 000126  
 47,  
 48, U 000052 05 15 00 00 0 000000  
 000053 50 01 00 00 0 000000  
 000054 72 11 00 00 0 000000  
 49, U 000055 73 17 00 00 0 000000  
 50, 000056 23 16 17 00 000010  
 51, 000057 51 00 00 12 0 000010  
 52, 000060 74 04 00 00 0 000064  
 53, U 000061 74 06 00 00 0 000000  
 U 000062 72 11 00 00 0 000000  
 54, 000063 74 04 00 00 0 000067  
 55,  
 56, U 000064 05 15 00 00 0 000000  
 000065 50 01 00 00 0 000000  
 000066 72 11 00 00 0 000000  
 57,  
 58, 000067 23 16 17 00 000013  
 59, 000070 27 01 13 12 0 000006  
 60, 000071 27 00 14 13 0 000000  
 61, 000072 06 00 14 12 0 000011  
 62, U 000073 73 17 00 00 0 000000  
 63, 000074 23 16 17 00 000021

J 70%  
 C010 C\*TS LOCKV  
 TS RECEIVER  
 L,U R15,26  
 TZ 9,X10  
 J 23%  
 C\*TSO RECEIVER  
 J 24%  
 23% C\*TS RECEIVER  
 24%  
 L,U R15,35  
 L AU,9,X10  
 L,H2 X11,7,X10  
 S AU,0,X11  
 TS SENDER  
 L,U R15,41  
 SZ 9,X10  
 C\*TSA SENDER  
 ER EXIT%  
 J 22%  
 C051 C\*TS LOCKV  
 TS SENDER  
 L,U R15,66  
 TNZ 9,X10  
 J 25%  
 C\*TSO SENDER  
 J 26%  
 25% C\*TS SENDER  
 26%  
 L,U R15,75  
 L,H2 X11,5,X10  
 L AU,0,X11  
 S AU,9,X10  
 TS RECEIVER  
 L,U R15,41

```

54.
55.      000075 23 16 17 00 000001      L,U      R15,1
56.      000076 04 00 17 12 0 000010      S        R15,0,X10
57.
58. U      000077 05 15 00 00 0 000000      C* TSA   RECEIVER
      000100 50 01 00 00 0 000000
      000101 72 11 00 00 0 000000
59. U      000102 72 11 00 00 0 000000      EP       EXIT$
70.      000103 74 04 00 00 0 000126      J        Z2$
71.
72.      000104 23 16 17 00 000140      C* T2    L,U      R15,96
73.      000105 05 00 00 12 0 000010      S        9,X10
74. U      000106 73 17 00 00 0 000000      TS       T$BUFFER
75. U      000107 05 15 00 00 0 000000      C* TSA   T$BUFFER
      000110 50 01 00 00 0 000000
      000111 72 11 00 00 0 000000
76. U      000112 73 17 00 00 0 000000      TS       SYSTEMHOLD
77. U      000113 74 06 00 00 0 000000      C* TSQ   SYSTEMHOLD
      000114 72 11 00 00 0 000000
      000115 74 04 00 00 0 000126      J        Z2$
78.
79.
80.      000116 55 16 00 00 000000      70$     TG,U    A0,0
81.      000117 55 16 00 00 000003      TG,U    A0,3
82.      000120 74 04 00 00 0 000125      J        Z1$
83.      000121 74 04 00 14 1 000122      J        *$+1,A0
84.      000122 000000000021      +C0$0
85.      000123 000000000052      +C0$1
86.      000124 000000000104      +C0$2
87.
88. U      000125 74 13 13 00 0 000000      Z1$     LM,J   X11,PXCS$
89.
90. U      000126 74 04 00 00 0 000000      Z2$     J        PRET$
91.      000000000012      DU$ FQU 10
92.      END

```

```

UNDEFINED SYMBOLS:
PRET$    PXCS$    SYSTEMHOLD  T$BUFFER  EXIT$    SENDER    CTSQ$    RECEIVER  LOCKV    PENT$    XS$BUFFER
F$R$JFFER  CTS$    CTS$      ASCFDASC$

```

```

END ASM. ERRORS : NONE    ITEM COUNT:    92
MAP, I      , TPF$ MAIN
MAP 30R1 S74T11 07/28/93 17:47:57

```

```

ADDRESS LIMITS    001000 011420    4369 TBANK WORDS DECIMAL
                  040000 046424    3349 DBANK WORDS DECIMAL
STARTING ADDRESS  005660

```

SEGMENT	MAIN%	001000 011420	040000 046424			
FDASC%/SYS74	%(1)	001000 001174				07 AUG 78 09:04:51
EDIT%/SYS74	%(1)	001175 001570				07 AUG 78 09:04:26
FRU%/SYS74P1						20 DEC 78 17:30:56
TARLE%/SYS74	%(1)	001571 001770				07 AUG 78 09:11:06
P%LEVEL/PASBP1	%(1)	001771 002004				23 OCT 78 11:52:20
P%PVF/PASBP1						17 JUL 78 13:55:38
P%CASE/PASBP1	%(1)	002005 002024	%(2)	040000 040000		18 MAR 80 14:27:44
	%(3)	002025 002031				
P%PAR/PASBP1			%(2)	040001 040034		30 AUG 79 10:44:44
P%WRITE/PASBP1	%(1)	002032 003232	%(2)	040037 040152		11 DEC 79 16:52:28
	%(3)	003233 003257				
P%TRACE/PASBP1	%(1)	003260 003433				23 OCT 78 11:52:21
	%(3)	003434 003466				
P%DJMP/PASBP1	%(1)	003467 003653	%(2)	040153 040164		23 OCT 78 11:52:18
	%(3)	003654 003670				
P%NJM/PASBP1	%(1)	003671 004044	%(2)	040155 040174		17 JUL 78 13:55:46
P%READ/PASBP1	%(1)	004045 004622	%(2)	040175 040204		23 APR 82 10:40:57
	%(3)	004623 004660				
P%INTRPT/PASBP1	%(1)	004661 005450	%(2)	040205 040305		23 APR 82 10:37:37
	%(3)	005451 005545				
P%STDERR/PASBP1	%(1)	005546 005615				10 DEC 79 17:45:17
	%(3)	005616 005651				
P%RS/PASBP1	%(1)	005652 007010	%(2)	040306 040411		23 JUL 80 13:49:59
	%(3)	007011 007220				
P%IO/PASBP1	%(1)	007221 010531				23 APR 82 10:40:32
	%(3)	010532 010734				
BUFFER	%(1)	010735 011063				28 JUL 83 17:47:56
PRINT	%(1)	011064 011154	%(0)	040412 040414		28 JUL 83 17:47:45
CARD	%(1)	011155 011247	%(0)	040415 040417		28 JUL 83 17:47:33
MAIN	%(1)	011250 011420	%(0)	040420 040424		28 JUL 83 17:47:21
			%(036)	040425 046424		

SYSS\*RLIR% LEVEL  
 END MAP, ERRORS: 0 TIME: 13,652 STORAGE: 17792/4/040777/073777

0USE INFDTABLE,INFO%.  
 0USE PAS000000,INFO%.  
 0SETC,D  
 0XQT TPF%.MAIN

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12

13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42

ER ABORT# ABORT ADR: 011420 BDI:000004  
ER ABORT# ABORT ADR: 011052 BDI:000004  
ID: 000001  
ER ABORT# ABORT ADR: 011133 BDI:000004  
ID: 000004  
ER ABORT# ABORT ADR: 011241 BDI:000004  
ID: 000003  
ER ABORT# ABORT ADR: 010735 BDI:000004  
ID: 000006  
RUNSTREAM ANALYSIS TERMINATED



## E.2 - CASE STUDY TWO: Dining Philosophers

```

@CPS,S,S ,CPS,S/O,
MACRO INTERPRETER LEVEL 7R1 07/27/83 07:51:28
1 *COMPILE OPTIONS=ISRLM,ELTNAME=TPFS,MAIN
1 PROGRAM MAIN(INPUT,OUTPUT);
1 TYPE LABELS=(PICKUP,PUTDOWN,INIT);
1 TYPE PHILENT=(PHILO,PHIL1,PHIL2,PHIL3,PHIL4,PINIT);
1 PROCEDURE FORKS(VAR FENTRY:LABELS;VAR PHILID:INTEGER);MONITOR;
1 PROCEDURE PHILS(VAR PENTRY:PHILENT);PROCESS;
1
1 VAP FENTRY : LABELS ;
1 PENTRY : PHILENT ;
1 PHILID : INTEGER ;
1 PHILO0 : QUEUE ;
1 PHILO1 : QUEUE ;
1 PHILO2 : QUEUE ;
1 PHILO3 : QUEUE ;
1 PHILO4 : QUEUE ;
1 ENTRYLOCK: QUEUE ;
1 WAITWD : QUEUE ;
1 PHILST : QUEUE ;
1
1 BEGIN
1 FENTRY := INIT ;
1 STARTUP WITH TASKNAME(FOO) FORKS(FENTRY,PHILID FORWARD);
1 PENTRY := PINIT ;
1 STARTUP WITH TASKNAME(POO) PHILS(PENTRY);
1 COREGIN(STARTER);
1 LOCK(PHILST);
1 PENTRY := PHILO ;
1 FORK WITH TASKNAME(P0) PHILS(PENTRY);
1 WAIT(PHILST);
1 LOCK(PHILST);
1 PENTRY := PHIL1 ;
1 FORK WITH TASKNAME(P1) PHILS(PENTRY);
1 WAIT(PHILST);
1 LOCK(PHILST);
1 PENTRY := PHIL2 ;
1 FORK WITH TASKNAME(P2) PHILS(PENTRY);
1 WAIT(PHILST);
1 LOCK(PHILST);
1 PENTRY := PHIL3 ;
1 FORK WITH TASKNAME(P3) PHILS(PENTRY);
1 WAIT(PHILST);
1 LOCK(PHILST);
1 PENTRY := PHIL4 ;
1 FORK WITH TASKNAME(P4) PHILS(PENTRY);
1 WAIT(PHILST);
1 COEND;
1 WAIT(WAITWD);
1 CLEAR(WAITWD);
1 SHUSPEND(1000);
1 TERMINATE;
1
1 END

```

```

1 *COMPILE    OPTIONS=ISRLMV,ELTNAMF=TPF&PHILS
1 PROGRAM DEFINE(PHILS);
1 TYPE LABELS=(PICKUP,PUTDOWN,INIT);
1 TYPE PHILENT=(PHILO,PHIL1,PHIL2,PHIL3,PHIL4,PINIT);
1 PROCEDURE PHILS(VAR PENTRY:PHILENT;VAR I:INTEGER);
1 PROCEDURE FORKS(VAR FENTRY:LABELS;VAR I:INTEGER);MONITOR;
1 VAP PHID0 : INTEGER ;
1 PHID1 : INTEGER ;
1 PHID2 : INTEGER ;
1 PHID3 : INTEGER ;
1 PHID4 : INTEGER ;
1 FENTRY : LABELS ;
1 BEGIN
1 CASE PENTRY OF
1 PHILO:
1 BEGIN
1 LOCK(PHILST);
1 SIGNAL(PHILST);
1 AWAIT(STARTER);
1 PHID0 := 0;
1 WHILE TRUE DO
1 BEGIN
1 SUSPEND(3);
1 LOCK(ENTRYLOCK);
1 LOCK(PHIL00);
1 FENTRY := PICKUP ;
1 FORK WITH TASKNAME(F0) FORKS(FENTRY,PHID0);
1 WAIT(PHIL00);
1 AWAIT(F0);
1 SUSPEND(8);
1 LOCK(ENTRYLOCK);
1 FENTRY := PUTDOWN ;
1 FORK WITH TASKNAME(F0) FORKS(FENTRY,PHID0);
1 AWAIT(F0);
1 QUIT BY CHECKING(WAITWD);
1 END;
1 END;
1 PHIL1:
1 BEGIN
1 LOCK(PHILST);
1 SIGNAL(PHILST);
1 AWAIT(STARTER);
1 PHID1 := 1;
1 WHILE TRUE DO
1 BEGIN
1 SUSPEND(2);
1 LOCK(ENTRYLOCK);
1 LOCK(PHIL01);
1 FENTRY := PICKUP ;
1 FORK WITH TASKNAME(F1) FORKS(FENTRY,PHID1);
1 WAIT(PHIL01);

```

```

1      AWAIT(F1);
1      SUSPEND(1);
1      LOCK(ENTRYLOCK);
1      FENTRY := PUTDOWN ;
1      FORK WITH TASKNAME(F1) FORKS(FENTRY,PHID1);
1      AWAIT(F1);
1      QUIT BY CHECKING(WAITWD);
1      END;
1      END;
1      PHIL2:
1      BEGIN
1      LOCK(PHILST);
1      SIGNAL(PHILST);
1      AWAIT(STARTER);
1      PHID2 := 2;
1      WHILE TRUE DO
1      BEGIN
1      SUSPEND(5);
1      LOCK(ENTRYLOCK);
1      LOCK(PHILO2);
1      FENTRY := PICKUP ;
1      FORK WITH TASKNAME(F2) FORKS(FENTRY,PHID2);
1      WAIT(PHILO2);
1      AWAIT(F2);
1
1      SUSPEND(2);
1      LOCK(ENTRYLOCK);
1      FENTRY := PUTDOWN ;
1      FORK WITH TASKNAME(F2) FORKS(FENTRY,PHID2);
1      AWAIT(F2);
1      QUIT BY CHECKING(WAITWD);
1      END;
1      END;
1      PHIL3:
1      BEGIN
1      LOCK(PHILST);
1      SIGNAL(PHILST);
1      AWAIT(STARTER);
1      PHID3 := 3;
1      WHILE TRUE DO
1      BEGIN
1      SUSPEND(6);
1      LOCK(ENTRYLOCK);
1      LOCK(PHILO3);
1      FENTRY := PICKUP ;
1      FORK WITH TASKNAME(F3) FORKS(FENTRY,PHID3);
1      WAIT(PHILO3);
1      AWAIT(F3);
1      SUSPEND(3);
1      LOCK(ENTRYLOCK);
1      FENTRY := PUTDOWN ;

```

```

1         FORK WITH TASKNAME(F3) FORKS(FENTRY,PHID3);
1         AWAIT(F3);
1         QUIT BY CHECKING(WAITWD);
1     END;
1     END;
1     PHIL4:
1     BEGIN
1         LOCK(PHILST);
1         SIGNAL(PHILST);
1         AWAIT(STARTER);
1         PHID4 := 4;
1         WHILE TRUE DO
1             BEGIN
1                 SUSPEND(4);
1                 LOCK(ENTRYLOCK);
1                 LOCK(PHILO4);
1                 FENTRY := PICKUP ;
1                 FORK WITH TASKNAME(F4) FORKS(FENTRY,PHID4);
1                 WAIT(PHILO4);
1                 AWAIT(F4);
1                 SUSPEND(9);
1                 LOCK(ENTRYLOCK);
1                 FENTRY := PUTDOWN ;
1                 FORK WITH TASKNAME(F4) FORKS(FENTRY,PHID4);
1                 AWAIT(F4);
1                 QUIT BY CHECKING(WAITWD);
1             END;
1         END;
1     END;
1     PINIT:
1     BEGIN
1         RELEASE(PHILS);
1         WAIT FOREVER;
1         QUIT;
1     END;
1     END;
1     END;
1     *COMPILE     OPTIONS=ISRLMV,ELTNAME=TPF5,FORKS
1     PROGRAM DEFINE(FORKS);
1     TYPE LABELS=(PICKUP,PUTDOWN,INIT);
1     PROCEDURE FORKS(VAR FENTRY:LABELS;VAR I:INTEGER);
1     VAR FRK      : ARRAY [ 0..4 ] OF 0..2 ;
1     VAR WAITING  : ARRAY [ 0..4 ] OF BOOLEAN ;
1     BEGIN
1     CASE FENTRY OF
1     PICKUP:
1     BEGIN
1     CASE I OF
1     0: LOCK(PHILO0);
1     1: LOCK(PHILO1);
1     2: LOCK(PHILO2);
1     3: LOCK(PHILO3);

```

```

1      4: LOCK(PHILO4);
1      END;
1
1      IF FRK[I] <> 2 THEN BEGIN
1          WAITING[I] := TRUE;
1          CASE I OF
1              0: UNLOCK(PHILO0);
1              1: UNLOCK(PHILO1);
1              2: UNLOCK(PHILO2);
1              3: UNLOCK(PHILO3);
1              4: UNLOCK(PHILO4);
1          END;
1      CREGION;
1      WRITELN;
1      WRITELN(' OPERATION      : PICKUP / REZERV FORKS ');
1      WRITE(' PHILOSOPHER    : ',I,' FORKS ON THE TABLE : ');
1      WRITE(FRK[0],' ',FRK[1],' ',FRK[2],' ',FRK[3],' ',FRK[4]);WRITELN;
1      WRITE(' WAITING ARRAY : ');
1      WRITE(WAITING[0],' ',WAITING[1],' ',WAITING[2],' ',WAITING[3],' ',WAITING[4]);
1      WRITELN;
1      CFND;
1
1          UNLOCK(ENTRYLOCK);
1          QUIT;
1          END
1
1      ELSE BEGIN
1          WAITING[I] := FALSE ;
1          FRK (I+4) MOD 5 ]:= FRK (I+4) MOD 5 ] + 1;
1          FRK (I+1) MOD 5 ]:= FRK (I+1) MOD 5 ] + 1;
1          CASE I OF
1              0: SIGNAL(PHILO0);
1              1: SIGNAL(PHILO1);
1              2: SIGNAL(PHILO2);
1              3: SIGNAL(PHILO3);
1              4: SIGNAL(PHILO4);
1          END;
1      CREGION;
1      WRITELN;
1      WRITELN(' OPERATION      : PICKUP / GET FORKS      ');
1      WRITE(' PHILOSOPHER    : ',I,' FORKS ON THE TABLE : ');
1      WRITE(FRK[0],' ',FRK[1],' ',FRK[2],' ',FRK[3],' ',FRK[4]);WRITELN;
1      WRITE(' WAITING ARRAY : ');
1      WRITE(WAITING[0],' ',WAITING[1],' ',WAITING[2],' ',WAITING[3],' ',WAITING[4]);
1      WRITELN;
1      CFND;
1      UNLOCK(ENTRYLOCK);
1      QUIT;
1      END;
1      END;
1      PUTDOWN;
1      BEGIN
1      FRK (I+4) MOD 5 ]:= FRK (I+4) MOD 5 ] + 1;

```

```

1 FRKC (I+1) MOD 5 J := FRKC (I+1) MOD 5 J + 1;
1 IF ( FRKC (I+4) MOD 5 J = 2 ) AND
1 ( WAITINGC (I+4) MOD 5 J ) THEN
1 BEGIN
1 WAITINGC (I+4) MOD 5 J := FALSE ;
1 FRKC (I+3) MOD 5 J := FRKC (I+3) MOD 5 J + 1;
1 FRKC I J := FRKC I J - 1;
1 CASE (I+4) MOD 5 OF
1 0: SIGNAL(PHILQ0);
1 1: SIGNAL(PHILQ1);
1 2: SIGNAL(PHILQ2);
1 3: SIGNAL(PHILQ3);
1 4: SIGNAL(PHILQ4);
1 END;
1 END;
1 IF ( FRKC (I+1) MOD 5 J = 2 ) AND
1 ( WAITINGC (I+1) MOD 5 J ) THEN
1 BEGIN
1 WAITINGC (I+1) MOD 5 J := FALSE ;
1 FRKC (I+2) MOD 5 J := FRKC (I+2) MOD 5 J + 1;
1 FRKC I J := FRKC I J - 1;
1 CASE (I+1) MOD 5 OF
1 0: SIGNAL(PHILQ0);
1 1: SIGNAL(PHILQ1);
1 2: SIGNAL(PHILQ2);
1 3: SIGNAL(PHILQ3);
1 4: SIGNAL(PHILQ4);
1 END;
1 END;
1 CREGION;
1 WRITELN;
1 WRITELN(' OPERATION      | PUTDOWN      ');
1 WRITE(' PHILOSOPHER   | ',1,' FORKS ON THE TABLE : ');
1 WRITE(FRKC0,' ',FRKC1,' ',FRKC2,' ',FRKC3,' ',FRKC4);WRITELN;
1 WRITE(' WAITING ARRAY : ');
1 WRITE(WAITINGC0,' ',WAITINGC1,' ',WAITINGC2,' ',WAITINGC3,' ',WAITINGC4);
1 WRITELN;
1 END;
1 UNLOCK(ENTRYLOCK);
1 QUIT;
1 END;
1 INIT:
1 BEGIN
1 FOR I:= 0 TO 4 DO BEGIN
1 FRKC I J := 0;
1 WAITINGC I J := FALSE;
1 END;
1 RELEASE(FORKS);
1 WAIT FOREVER;
1 SIGNAL(WAITWD);
1 QUIT;

```

```

1      END;
1      END;
1      END;
1
@CPS,S,N  SCRATCH$,SCRATCH1,CPS,S/1,SCRATCH$,SCRATCH2
MACRO INTERPRETER LEVEL 7R1 07/27/83 07:56:06
@CPS,S,N  SCRATCH$,SCRATCH2,CPS,S/2,
MACRO INTERPRETER LEVEL 7R1 07/27/83 07:56:44
@CPS,X
@BU*PASCAL$,ISRLM
PASCAL ARIA PLTB WEDNESDAY, 1983 JULY 27, 7:58:13
1

```

```

2  PROGRAM MATN(INPUT,OUTPUT);
3  TYPE LABELS=(PTCKUP,PUTDOWN,INIT);
4  TYPE PHILENT=(PHILO,PHIL1,PHIL2,PHIL3,PHIL4,PINIT);
5  PROCEDURE FORKS(VAR FENTRY:LABELS;VAR PHILID:INTEGER);EXTERNAL ;
6  PROCEDURE PHILS(VAR PENTRY:PHILENT);EXTERNAL ;
7  VAR GLOBALS : ARRAY (1..1821) OF INTEGER ;
8      FENTRY : LABELS ;
9      PENTRY : PHILENT ;
10     PHILID : INTEGER ;
11 $CODE RSCCELL*      T$CELL
12 $CODE CBGNST*      T$CELL
13 $CODE SYSTEMHOLD*  T$CELL
14 $CODE ENTRYLOCK*  T$CELL
15 $CODE PHILO0*      T$CELL
16 $CODE PHILO1*      T$CELL
17 $CODE PHILQ2*      T$CELL
18 $CODE PHILQ3*      T$CELL
19 $CODE PHILO4*      T$CELL
20 $CODE PHILST*      T$CELL
21 $CODE WAITWD*      T$CELL
22 $CODE T$FORKS*     T$CELL
23 $CODE T$MAIN*      T$CELL
24 $CODE T$PHILS*     T$CELL
25 $CODE X$FORKS*     RFS 1
26 $CODE X$MAIN*      RES 1
27 $CODE X$PHILS*     RFS 1
28 $CODE CBGNAME*     RES 1
29 $CODE F$FORKS*     RES 1
30 $CODE F$MAIN*      RES 1
31 $CODE F$PHILS*     RES 1
32 $CODE SVRGA0*      RES 900
33 $CODE SVRGA1*      RES 900
34 $CODE             RES SIZE*_136_1821
35 $CODE             $(1)
36
37 $CODE COBEGIN*
38 $CODE             L          AP,('CBG')

```

```

1      0 135
1
1
2      0 6
2      0 5
1      136 1956
1      1957 1957
1      1958 1958
1      1959 1959
1
1 0
1 1
1 2
1 3
1 4
1 5
1 6
1 7
1 8
1 9
1 10
1 11
1 12
1 13
1 14
1 15
1 16
1 17
1 18
1 19
1 20
1 21
1 22
1 23
1 24
1
1 25
1 26

```

```

P$CHECK/PAS8P1      $(1)  010735 01044      $(2)  040412 040417      18 MAR 80  14:27:55
                   $(3)  011045 011070
FORKS               $(1)  011071 012516      27 JUL 83  08:01:06
                   $(3)  012517 012604
PHILS               $(1)  012605 013407      $(0)   040420 040432      27 JUL 83  08:00:11
MAIN                $(1)  013410 013701      $(0)   040433 040443      27 JUL 83  07:59:28
                   $(036) 040444 046444

```

```

SYS$RLIBS: LEVEL
END MAP, ERRORS: 0 TIME: 14,671 STORAGE: 17792/4/040777/U73777
@USE  INFOTABLE,INFO$,
@USE  PAS000000,INFO$,
@SETC,0
@XOT  TPF$,MAIN

```

```

OPERATION      : PICKUP / GET FORKS
PHILOSOPHER   : 4   FORKS ON THE TABLE : 1 2 2 1 2
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

```

```

OPERATION      : PICKUP / GET FORKS
PHILOSOPHER   : 1   FORKS ON THE TABLE : 0 2 1 1 2
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

```

```

OPERATION      : PUTDOWN
PHILOSOPHER   : 4   FORKS ON THE TABLE : 1 2 1 2 2
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

```

```

OPERATION      : PUTDOWN
PHILOSOPHER   : 1   FORKS ON THE TABLE : 2 2 2 2 2
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

```

```

OPERATION      : PICKUP / GET FORKS
PHILOSOPHER   : 4   FORKS ON THE TABLE : 1 2 2 1 2
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

```

```

OPERATION      : PICKUP / GET FORKS
PHILOSOPHER   : 1   FORKS ON THE TABLE : 0 2 1 1 2
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

```

```

OPERATION      : PUTDOWN
PHILOSOPHER   : 4   FORKS ON THE TABLE : 1 2 1 2 2
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

```

```

OPERATION      : PUTDOWN
PHILOSOPHER   : 1   FORKS ON THE TABLE : 2 2 2 2 2
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

```

```

OPERATION      : PICKUP / GET FORKS
PHILOSOPHER   : 4   FORKS ON THE TABLE : 1 2 2 1 2
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

```

```

OPERATION      : PICKUP / GET FORKS

```



PHILOSOPHER : 1 FORKS ON THE TABLE : 0 2 1 1 2  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 4 FORKS ON THE TABLE : 1 2 1 2 2  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 1 FORKS ON THE TABLE : 2 2 2 2 2  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 4 FORKS ON THE TABLE : 1 2 2 1 2  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 1 FORKS ON THE TABLE : 0 2 1 1 2  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 4 FORKS ON THE TABLE : 1 2 1 2 2  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 1 FORKS ON THE TABLE : 2 2 2 2 2  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 4 FORKS ON THE TABLE : 1 2 2 1 2  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / REZERV FORKS  
PHILOSOPHER : 0 FORKS ON THE TABLE : 1 2 2 1 2  
WAITING ARRAY : TRUE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 4 FORKS ON THE TABLE : 2 1 2 2 1  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 3 FORKS ON THE TABLE : 2 1 1 2 0  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 0 FORKS ON THE TABLE : 2 2 1 2 1  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 3 FORKS ON THE TABLE : 2 2 2 2 2  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 0 FORKS ON THE TABLE : 2 1 2 2 1  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 3 FORKS ON THE TABLE : 2 1 1 2 0  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 0 FORKS ON THE TABLE : 2 2 1 2 1  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 3 FORKS ON THE TABLE : 2 2 2 2 2  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 0 FORKS ON THE TABLE : 2 1 2 2 1  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 3 FORKS ON THE TABLE : 2 1 1 2 0  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 0 FORKS ON THE TABLE : 2 2 1 2 1  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 3 FORKS ON THE TABLE : 2 2 2 2 2  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 0 FORKS ON THE TABLE : 2 1 2 2 1  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 3 FORKS ON THE TABLE : 2 1 1 2 0  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 0 FORKS ON THE TABLE : 2 2 1 2 1  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 3 FORKS ON THE TABLE : 2 2 2 2 2  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 0 FORKS ON THE TABLE : 2 1 2 2 1  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 3 FORKS ON THE TABLE : 2 1 1 2 0  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 0 FORKS ON THE TABLE : 2 2 1 2 1  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 3 FORKS ON THE TABLE : 2 2 2 2 2  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 0 FORKS ON THE TABLE : 2 1 2 2 1  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 3 FORKS ON THE TABLE : 2 1 1 2 0  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 0 FORKS ON THE TABLE : 2 2 1 2 1  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 3 FORKS ON THE TABLE : 2 2 2 2 2  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 0 FORKS ON THE TABLE : 2 1 2 2 1  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 3 FORKS ON THE TABLE : 2 1 1 2 0  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 0 FORKS ON THE TABLE : 2 2 1 2 1  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PUTDOWN  
PHILOSOPHER : 3 FORKS ON THE TABLE : 2 2 2 2 2  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 0 FORKS ON THE TABLE : 2 1 2 2 1  
WAITING ARRAY : FALSE FALSE FALSE FALSE FALSE

OPERATION : PICKUP / GET FORKS  
PHILOSOPHER : 3 FORKS ON THE TABLE : 2 1 1 2 0

### E.3.1 - CASE STUDY THREE: Quicksort (Without Processor Dedication)

```

@CPS,S ,CPS,S/O,
MACRO INTERPRETER LEVEL TRI 07/27/83 08:03:12
I *COMPILE OPTIONS=IRSLMG,ELTNAME=TPFS,MAIN
I PROGRAM MAIN(INPUT,OUTPUT);
I CONST N=1000;
I TYPE Q1ENTRIES = ( Q1INIT , Q1START );
I Q2ENTRIES = ( Q2INIT , Q2START );
I ARRAYTYPE = ARRAY [ 1 , N ] OF INTEGER ;
I PROCEDURE DIVIDETWO(VAR X : ARRAYTYPE; N : INTEGER); PROCESS;
I PROCEDURE QUICKONE(VAR Q1ENTRY : Q1ENTRIES;VAR X : ARRAYTYPE; LOWER1 : INTEGER; UPPER1 : INTEGER); PROCESS;
I PROCEDURE QUICKTWO(VAR Q2ENTRY : Q2ENTRIES;VAR X : ARRAYTYPE; LOWER2 : INTEGER; UPPER2 : INTEGER); PROCESS;
I VAR FINISH : QUEUE ;
I QLOCK1 : QUEUE ;
I QLOCK2 : QUEUE ;
I FORKLOCK : QUEUE ;
I Q1ENTRY : Q1ENTRIES ;
I Q2ENTRY : Q2ENTRIES ;
I DMINT : INTEGER ;
I I,CP : INTEGER ;
I X : ARRAYTYPE ;
I BEGIN
I CREGION;
I READLN;
I PAGE; WRITELN; WRITELN;
I WRITELN(' ** UNSORTED ARRAY (1000 ELEMENTS) ** ');
I WRITELN; WRITELN; WRITELN;
I CEND;
I FOR I := 1 TO N
I DO BEGIN
I CREGION; READ(X(I)); WRITE(X(I):5); CEND;
I IF ( I MOD 20 ) = 0 THEN BEGIN CREGION; WRITELN; CEND; END;
I END;
I LOCK(FINISH); LOCK(QLOCK1); LOCK(QLOCK2);
I CREGION; CP := CPTIME(3); CEND;
I Q1ENTRY := Q1INIT;
I STARTUP WITH TASKNAME(TO1) QUICKONE(Q1ENTRY,X,DMINT FORW,DMINT FORW);
I Q2ENTRY := Q2INIT;
I STARTUP WITH TASKNAME(TO2) QUICKTWO(Q2ENTRY,X,DMINT FORW,DMINT FORW);
I STARTUP WITH TASKNAME(TO0) DIVIDETWO(X,N);
I WAIT(FINISH);
I CREGION;
I CP := CPTIME(3)-CP;
I PAGE; WRITELN; WRITELN; WRITELN(' TIME CONSUMED : ',CP,' MSEC');
I WRITELN; WRITELN;
I WRITELN; WRITELN; WRITELN(' ** SORTED ARRAY ** ');
I WRITELN; WRITELN;
I CEND;
I FOR I := 1 TO N
I DO BEGIN
I CREGION; WRITE(X(I):5); CEND;
I IF ( I MOD 20 ) = 0 THEN BEGIN CREGION; WRITELN; CEND; END;
I

```

```

I      END;
I      LOCK(QLOCK1);
I      SIGNAL(QLOCK1);
I      LOCK(QLOCK2);
I      SIGNAL(QLOCK2);
I      QUIT;
I
I      END;
I      *COMPILE      OPTIONS=IRSLMVG,ELTNAME=TPF%,DIVIDE2
I      PROGRAM DEFINE(DIVIDETWO);
I      CONST N=1000;
I      TYPE Q1ENTRIES =( Q1INIT , Q1START );
I      Q2ENTRIES =( Q2INIT , Q2START );
I      ARRAYTYPE = ARRAY [ 1 .. N ] OF INTEGER ;
I      PROCEDURE DIVIDETWO(VAR X : ARRAYTYPE; N : INTEGER);
I      PROCEDURE QUICKONE(VAR Q1ENTRY : Q1ENTRIES;VAR X : ARRAYTYPE; LOWER1 : INTEGER; UPPER1 : INTEGER); PROCESS;
I      PROCEDURE QUICKTWO(VAR Q2ENTRY : Q2ENTRIES;VAR X : ARRAYTYPE; LOWER2 : INTEGER; UPPER2 : INTEGER); PROCESS;
I      VAR UP,DOWN,A : INTEGER ;
I          LB,UB,J : INTEGER ;
I          Q1ENTRY : Q1ENTRIES ;
I          Q2ENTRY : Q2ENTRIES ;
I          JPI,JMI : INTEGER ;
I          TEMP : INTEGER ;
I      BEGIN
I      RELEASE(DIVIDETWO);
I      LB := 1 ;   UB := N ;   A := X [ LB ] ;
I      UP := UB ;   DOWN := LB ;
I      REPEAT
I          WHILE ( UP > DOWN ) AND ( X [ UP ] > A ) DO UP := UP - 1;
I          J := UP ;
I          IF UP <> DOWN THEN BEGIN
I              TEMP := X [ DOWN ] ;
I              X [ DOWN ] := X [ UP ] ;
I              X [ UP ] := TEMP ;
I              WHILE ( DOWN < UP ) AND ( X [ DOWN ] <= A ) DO DOWN := DOWN + 1;
I              J := DOWN;
I              IF DOWN <> UP THEN BEGIN
I                  TEMP := X [ UP ] ;
I                  X [ UP ] := X [ DOWN ] ;
I                  X [ DOWN ] := TEMP ;
I              END;
I          END;
I      UNTIL DOWN = UP ;
I      X [ J ] := A ;
I      JPI := J + 1;
I      JMI := J - 1;
I      IF UB > JPI THEN BEGIN
I          LOCK(FORKLOCK);
I          Q1ENTRY := Q1START ;
I          FORK WITH TASKNAME(TD6) QUICKONE(Q1ENTRY,X,JPI,UB);
I          WAIT(FORKLOCK);
I      END;

```

```

I      IF JM1 > LB THEN BEGIN
I          LOCK(FORKLOCK);
I          Q2ENTRY := Q2START ;
I          FORK WITH TASKNAME(T07) QUICKTWO(Q2ENTRY,X,LB,JM1);
I          WAIT(FORKLOCK);
I      END;
I      AWAIT(T06,T07);
I      LOCK(FINISH);
I      SIGNAL(FINISH);
I      QUIT;
I      END;
I      *COMPILE      OPTIONS=IRSLMVG,ELTNAME=TPFS,QUICK1
I      PROGRAM DEFINE(QUICKONE);
I      CONST      N=1000;
I      TYPE      Q1ENTRIES =( Q1INIT , Q1START );
I      Q2ENTRIES =( Q2INIT , Q2START );
I      ARRAYTYPE = ARRAY [ 1 .. N ] OF INTEGER ;
I      PROCEDURE QUICKONE(VAR ENTRY : Q1ENTRIES; VAR X:ARRAYTYPE; LOWER : INTEGER; UPPER : INTEGER);
I      TYPE      STACKITEM = RECORD
I          LB : INTEGER ;
I          UB : INTEGER ;
I      END;
I      STACK = RECORD
I          TOP : INTEGER ;
I          ITEM : ARRAY [ 1 .. N ] OF STACKITEM ;
I      END;
I      VAR      %      I      STACK      I
I          NEWBND : STACKITEM ;
I          I,J,A : INTEGER ;
I          UP,DOWN : INTEGER ;
I          TEMP : INTEGER ;
I      BEGIN
I      CASE ENTRY OF
I      Q1START:
I      BEGIN
I      LOCK(FORKLOCK);
I      SIGNAL(FORKLOCK);
I      WITH NEWBND
I      DO BEGIN
I      LB := LOWER;  UB := UPPER;
I      S1TOP := S1TOP + 1;
I      S1ITEM[S1TOP]LB := LB;
I      S1ITEM[S1TOP]UB := UB;
I      WHILE NOT(S1TOP = 0)
I      DO BEGIN
I      LB := S1ITEM[S1TOP]LB;
I      UB := S1ITEM[S1TOP]UB;
I      S1TOP := S1TOP - 1;
I      WHILE UB > LB DO BEGIN
I          A := X [ LB ]; J:= LB; UP := UB; DOWN:= LR;
I          REPEAT

```



```

1      ARRAYTYPE = ARRAY [ 1 : N ] OF INTEGER ;
1      PROCEDURE QUICKTWO (VAR ENTRY : QZENTRIES; VAR X:ARRAYTYPE; LOWER : INTEGER; UPPER : INTEGER);
1      TYPE STACKITEM = RECORD
1          LB : INTEGER ;
1          UB : INTEGER ;
1          END;
1      STACK = RECORD
1          TOP : INTEGER ;
1          ITEM : ARRAY [ 1 : N ] OF STACKITEM ;
1          END;
1      VAR S : STACK ;
1          NEWBND : STACKITEM ;
1          I,J,A : INTEGER ;
1          UP,DOWN : INTEGER ;
1          TEMP : INTEGER ;
1      BEGIN
1      CASE ENTRY OF
1      QZSTART:
1      BEGIN
1      LOCK(FORKLOCK);
1      SIGNAL(FORKLOCK);
1      WITH NEWBND
1      DO BEGIN
1      LB := LOWER; UB := UPPER;
1      S.TOP := S.TOP + 1;
1      S.ITEM[S.TOP].LB := LB;
1      S.ITEM[S.TOP].UB := UB;
1      WHILE NOT(S.TOP = 0)
1      DO BEGIN
1      LB := S.ITEM[S.TOP].LB;
1      UB := S.ITEM[S.TOP].UB;
1      S.TOP := S.TOP - 1;
1      WHILE UB > LB DO BEGIN
1      A := X [ LB ]; J := LB; UP := UB; DOWN := LB;
1      REPEAT
1      WHILE (UP > DOWN) AND (X [ UP ] >= A) DO UP := UP - 1;
1      J := UP ;
1      IF UP <> DOWN THEN BEGIN
1      TEMP := X [ UP ];
1      X [ UP ] := X [ DOWN ];
1      X [ DOWN ] := TEMP ;
1      END;
1      WHILE (DOWN < UP) AND (X[DOWN] <= A)
1      DO DOWN := DOWN + 1;
1      J := DOWN;
1      IF DOWN <> UP THEN BEGIN
1      TEMP := X [ UP ];
1      X [ UP ] := X [ DOWN ];
1      X [ DOWN ] := TEMP ;
1      END;
1      UNTIL DOWN = UP ;

```





P\$TRACE/PASBR1	\$(1)	003516 003671			23 OCT 78	11:52:21
	\$(3)	003672 003724				
P\$WRITE/PASBR1	\$(1)	003725 005125	\$(2)	040077 040212	11 DEC 79	16:52:28
	\$(3)	005126 005152				
P\$INTRPT/PASBR1	\$(1)	005153 005742	\$(2)	040213 040313	23 APR 82	10:37:37
	\$(3)	005743 006037				
P\$RS/PASBR1	\$(1)	006040 007176	\$(2)	040314 040417	23 JUL 80	13:49:59
	\$(3)	007177 007406				
P\$IO/PASBR1	\$(1)	007407 010717			23 APR 82	10:40:32
	\$(3)	010720 011122				
QUICK2	\$(1)	011123 011710	\$(0)	040420 040421	27 JUL 83	08:21:23
QUICK1	\$(1)	011711 012476	\$(0)	040422 040423	27 JUL 83	08:20:13
DIVIDE2	\$(1)	012477 013105	\$(0)	040424 040427	27 JUL 83	08:19:11
MAIN	\$(1)	013106 013555	\$(0)	040430 040433	27 JUL 83	08:18:23
	\$(3)	013556 013601	\$(036)	040434 140435		

SYSS\*RLIB% LEVEL

END MAP, ERRORS: 0 TIME: 14.353 STORAGE: 17792/4/040777/073777

#USE INFOTABLE,INFO%

#USE PAS00000,INFO%

#SETC,D

#XQT TPF%,MAIN

\*\* UNSORTED ARRAY (1000 ELEMENTS) \*\*

674	137	901	506	283	761	416	940	896	976	311	875	367	723	967	842	993	460	703	475
835	338	788	825	743	503	814	956	746	197	193	901	36	25	881	516	889	977	262	754
436	511	486	335	328	245	388	766	885	485	474	129	462	63	269	31	313	678	717	563
366	137	117	110	890	617	459	288	144	292	837	286	679	396	1	343	70	979	899	52
440	377	465	676	240	601	687	597	328	574	402	527	506	391	47	31	243	456	979	434
427	592	807	104	572	945	511	604	996	693	0	167	91	330	63	400	894	469	548	125
756	870	909	695	12	579	163	131	142	820	559	849	998	463	203	775	288	642	130	43
579	299	241	512	685	143	803	740	406	761	97	84	161	292	115	7	171	337	75	684
420	469	836	455	748	941	171	613	975	892	181	551	110	593	979	175	230	974	114	40
763	895	69	571	461	361	478	241	464	63	573	529	450	396	720	952	949	265	724	431
505	768	948	733	447	14	121	679	646	87	73	297	710	366	442	52	671	140	790	770
316	670	370	812	887	574	142	573	4	623	489	291	225	763	420	666	769	386	879	733
516	605	153	912	494	495	179	120	657	990	929	307	845	334	860	740	427	711	804	826
195	907	458	250	706	543	47	639	321	674	515	982	835	257	973	443	622	180	988	239
883	317	605	660	613	244	865	332	552	997	931	253	439	458	738	116	493	689	490	911
168	731	95	754	760	130	50	349	605	643	66	495	163	319	113	474	373	355	383	168
281	854	17	171	28	819	304	193	859	456	366	457	942	48	239	816	358	419	546	746
709	185	848	348	413	742	83	990	824	133	340	136	921	686	581	429	391	484	741	35
547	517	2	717	595	506	867	119	363	176	19	151	707	932	762	382	395	625	226	787
358	310	341	185	814	629	911	155	833	648	5	192	527	715	184	349	144	122	955	548
915	59	177	239	628	585	974	54	473	52	90	540	809	629	417	78	91	304	816	504
292	957	163	964	744	815	907	980	429	806	279	676	322	86	620	588	506	173	143	442
360	904	805	763	205	599	778	135	420	561	379	413	87	260	149	147	867	248	125	950
855	103	169	510	800	380	597	428	132	799	166	999	868	605	837	561	518	242	562	986
754	236	701	348	668	354	513	788	40	868	502	68	147	948	257	244	228	924	752	397
802	396	930	300	660	873	734	384	412	843	583	326	87	893	657	52	365	739	167	928
630	699	159	359	97	415	229	6	898	305	878	284	67	468	65	794	997	474	878	771
997	701	85	655	196	584	644	300	139	367	143	360	981	682	427	817	317	466	573	610
263	472	663	785	486	850	687	552	491	966	237	105	572	880	487	38	398	587	329	645
332	390	334	372	159	432	596	149	986	701	329	306	691	476	440	249	372	449	157	19
853	454	111	358	384	955	139	848	296	168	401	170	418	910	251	503	707	981	666	531
601	112	840	530	207	231	940	429	546	2	821	763	643	337	885	847	371	767	67	519
530	323	319	236	749	734	768	286	392	531	93	356	730	825	504	651	253	253	42	233
82	820	771	223	22	58	777	978	914	171	723	297	527	530	66	234	311	116	876	6
955	205	545	984	756	858	518	727	431	420	518	472	62	540	584	465	498	856	590	0
588	782	740	986	251	501	896	335	470	513	431	445	862	796	649	490	633	866	635	250
254	715	789	614	324	877	81	924	683	516	462	755	911	733	677	703	955	928	83	239
720	333	904	631	52	517	738	418	632	894	909	897	899	979	672	528	209	369	70	160
34	116	612	424	96	419	771	624	927	238	672	924	441	686	130	592	692	749	606	527
97	174	446	280	984	660	130	240	118	133	103	607	611	779	94	283	787	247	687	736
242	339	234	399	639	132	62	105	921	796	257	543	172	629	157	700	171	564	24	679
254	611	219	320	821	425	549	447	863	391	267	587	513	372	605	452	705	433	515	442
603	629	653	911	598	118	505	888	210	604	858	428	379	322	699	728	173	516	947	625
85	536	280	35	641	459	648	398	157	300	921	125	77	640	463	540	414	691	254	766
399	550	537	475	559	611	813	383	24	811	347	865	950	632	176	998	562	457	994	835

396	149	78	674	907	393	819	557	734	585	223	189	472	782	108	962	916	397	572	710
331	114	638	311	384	641	830	207	1	703	797	388	455	845	897	357	631	368	58	56
489	927	32	697	330	33	519	936	580	928	478	757	400	22	294	945	510	292	295	950
210	769	483	461	744	453	157	940	511	718	479	847	179	482	855	267	782	148	361	96
432	528	272	946	639	679	125	342	394	823	553	828	763	206	498	989	808	991	249	437

TIME CONSUMED : 1010 MSFC

\*\* SORTED ARRAY \*\*

0	0	1	1	2	2	4	5	6	6	7	12	14	17	19	19	22	22	24	24
25	28	31	31	32	33	34	35	35	36	38	40	40	42	43	47	47	48	50	52
52	52	52	52	54	56	58	58	59	62	62	63	63	63	65	66	66	67	67	68
69	70	70	73	75	77	78	78	81	82	83	83	84	85	85	86	87	87	87	90
91	91	92	93	94	95	96	96	97	97	103	103	104	105	105	108	110	110	111	112
113	114	114	115	116	116	116	117	118	119	120	121	122	124	125	125	125	125	129	130
130	130	130	131	132	132	133	133	133	135	136	137	137	139	139	140	142	142	143	143
144	144	147	147	148	149	149	149	151	153	155	157	157	157	157	159	159	160	161	163
163	163	164	167	167	168	168	168	169	170	171	171	171	171	171	172	173	173	175	176
176	177	179	179	180	181	184	185	185	189	192	193	193	195	196	197	203	205	205	206
207	207	209	210	218	219	223	223	225	226	228	229	230	231	233	234	234	236	236	237
238	239	239	239	239	240	240	241	241	242	242	243	244	244	245	247	248	249	249	250
250	251	251	253	253	253	254	254	254	257	257	257	260	262	263	265	267	267	269	272
279	280	280	281	283	283	284	286	286	288	288	291	292	292	292	294	295	296	296	297
297	299	300	300	300	304	304	305	306	307	310	311	311	311	313	316	317	317	318	319
319	320	321	322	322	323	324	326	328	328	329	329	330	330	331	332	332	333	334	334
335	335	337	337	338	339	340	341	342	343	347	348	348	349	349	354	355	356	357	358
358	358	359	360	360	361	361	363	365	366	366	366	366	367	367	368	369	370	371	372
372	373	377	379	379	380	382	383	383	384	384	384	386	388	388	390	391	391	391	392
393	394	395	396	396	396	396	397	397	398	398	399	399	400	400	401	402	406	412	413
413	414	415	416	417	418	418	419	419	420	420	420	420	424	425	427	427	427	428	428
429	429	429	431	431	431	432	432	432	433	434	436	437	439	440	440	441	442	442	443
445	446	447	447	449	450	452	453	454	455	455	456	456	457	457	458	458	459	459	460
461	461	462	462	463	463	464	465	465	466	468	469	469	470	472	472	472	473	474	474
474	475	475	476	478	478	479	482	483	484	485	486	486	487	489	489	490	490	491	493
494	495	495	496	498	501	502	503	503	504	504	505	505	506	506	506	506	510	510	511
511	511	512	513	513	513	515	515	516	516	516	516	517	517	518	518	518	519	519	527
527	527	527	528	528	529	530	530	530	531	531	536	537	540	540	540	543	543	545	546
546	547	548	548	549	550	551	552	552	553	557	559	559	561	561	562	562	563	564	571
572	572	572	573	573	573	574	574	579	579	580	581	583	584	584	585	585	587	587	588
588	590	592	592	593	595	596	597	597	598	599	601	601	603	604	604	605	605	605	605
605	606	607	610	611	611	611	612	613	613	614	617	620	622	623	624	625	625	628	629
629	629	629	630	631	631	632	632	633	635	638	639	639	639	640	641	641	642	643	643
644	645	646	648	648	649	651	653	655	657	657	660	660	660	663	666	666	668	670	671
672	672	674	674	674	676	676	677	678	679	679	679	679	682	683	684	685	686	686	687
687	687	689	691	691	692	693	695	697	699	699	700	701	701	701	703	703	703	705	706
707	707	709	710	710	711	715	715	717	717	718	720	720	723	723	724	727	728	730	731
733	733	733	734	734	734	736	738	739	740	740	740	741	742	743	744	744	746	746	746
748	749	749	752	754	754	754	755	756	756	757	760	761	761	762	763	763	763	763	763
766	766	767	768	768	769	769	770	771	771	771	775	777	778	779	782	782	782	785	787
788	788	789	790	794	796	796	797	797	797	799	800	802	803	804	805	806	807	808	811

812	813	814	814	815	816	816	817	819	819	820	820	821	821	823	824	825	825	826	828
830	833	835	835	835	836	837	837	840	842	843	845	845	847	847	848	848	849	850	853
854	855	855	856	858	858	859	860	862	863	865	865	866	867	867	868	868	870	873	875
876	877	878	878	879	880	881	883	885	885	887	888	889	890	892	893	894	894	895	896
896	897	897	898	899	899	901	901	904	904	907	907	907	909	909	910	911	911	911	911
912	914	915	916	921	921	921	924	924	924	927	927	928	928	928	929	930	931	932	936
940	940	940	941	942	945	945	946	947	948	948	949	950	950	950	952	954	955	955	955
955	956	957	962	964	966	966	967	973	974	974	975	976	977	978	979	979	979	980	981
981	982	984	986	986	986	988	989	990	990	991	993	994	996	997	997	997	998	998	999

### E.3.2 - CASE STUDY THREE: Quicksort (With Processor Dedication)

```

1  IF JM1 > LB THEN BEGIN
1      LOCK(FORKLOCK);
1      QZENTRY := QZSTART ;
1      FORK WITH TASKNAME(TO7) QUICKTWO(QZENTRY,X,LB,JM1);
1      WAIT(FORKLOCK);
1      END;
1
1  AWAIT(TO6,TO7);
1  LOCK(FINISH);
1  SIGNAL(FINISH);
1  QUIT;
1  FND;
1
1  *COMPILE  OPTIONS=IRSI,MVG,ELTNAME=TPFS,QUICK1
1  PROGRAM DEFINE(QUICKONF);
1  CONST N=1000;
1  TYPE QENTRIES =( Q1INIT , Q1START );
1  Q2ENTRIES =( Q2INIT , Q2START );
1  ARRAYTYPE = ARRAY [ 1 .. N ] OF INTEGER ;
1  PROCEDURE QUICKONE(VAR ENTRY : QENTRIES; VAR X:ARRAYTYPE; LOWER : INTEGER; UPPER : INTEGER);
1  TYPE STACKITEM = RECORD
1      LB : INTEGER ;
1      UP : INTEGER ;
1      END;
1  STACK = RECORD
1      TOP : INTEGER ;
1      ITEM : ARRAY [ 1 .. N ] OF STACKITEM ;
1      END;
1  VAR S : STACK ;
1  NEWBND : STACKITEM ;
1  I,J,A : INTEGER ;
1  UP,DOWN : INTEGER ;
1  TEMP : INTEGER ;
1  BEGIN
1  CASE ENTRY OF
1  Q1START:
1  BEGIN
1  LOCK(FORKLOCK);
1  SIGNAL(FORKLOCK);
1  DEDICATE PROCESSOR(O);
1  WITH NEWBND
1  DO BEGIN
1  LB := LOWER;  UB := UPPER;
1  S.TOP := S.TOP + 1;
1  S.ITEM[S.TOP] LB := LB;
1  S.ITEM[S.TOP] UB := UB;
1  WHILE NOT(S.TOP = 0)
1  DO BEGIN
1  LB := S.ITEM[S.TOP] LB;
1  UB := S.ITEM[S.TOP] UB;
1  S.TOP := S.TOP - 1;
1  WHILE UB > LB DO BEGIN
1  A := X [ LB ]; J:= LB; UP := UB; DOWN:= LB;

```

```

1      Q2ENTRIES = ( Q2INIT , Q2START );
1      ARRAYTYPE = ARRAY [ 1 .. N ] OF INTEGER ;
1      PROCEDURE QUICKTWO (VAR ENTRY : Q2ENTRIES; VAR X:ARRAYTYPE; LOWER : INTEGER; UPPER : INTEGER);
1      TYPE STACKITEM = RECORD
1          LB : INTEGER ;
1          UB : INTEGER ;
1          END;
1      STACK = RECORD
1          TOP : INTEGER ;
1          ITEM : ARRAY [ 1 .. N ] OF STACKITEM ;
1          END;
1      VAR S : STACK ;
1          NEWBND : STACKITEM ;
1          I,J,A : INTEGER ;
1          UP,DOWN : INTEGER ;
1          TEMP : INTEGER ;
1      BEGIN
1      CASE ENTRY OF
1      Q2START:
1      BEGIN
1      LOCK(FORKLOCK);
1      SIGNAL(FORKLOCK);
1      DEDICATE PROCESSOR(1);
1      WITH NEWBND
1      DO BEGIN
1      LB := LOWER; UB := UPPER;
1      S.TOP := S.TOP + 1;
1      S.ITEM[S.TOP] LB := LB;
1      S.ITEM[S.TOP] UB := UB;
1      WHILE NOT(S.TOP = U)
1      DO BEGIN
1      LB := S.ITEM[S.TOP] LB;
1      UB := S.ITEM[S.TOP] UB;
1      S.TOP := S.TOP + 1;
1      WHILE UB > LB DO BEGIN
1      A := X [ LB ]; J := LB; UP := UB; DOWN := LB;
1      REPEAT
1      WHILE (UP > DOWN) AND (X [ UP ] >= A ) DO UP := UP + 1;
1      J := UP ;
1      IF UP <> DOWN THEN BEGIN
1      TEMP := X [ UP ];
1      X [ UP ] := X [ DOWN ];
1      X [ DOWN ] := TEMP ;
1      END;
1      WHILE (DOWN < UP) AND (X[DOWN] <= A)
1      DO DOWN := DOWN + 1;
1      J := DOWN;
1      IF DOWN <> UP THEN BEGIN
1      TEMP := X [ UP ];
1      X [ UP ] := X [ DOWN ];
1      X [ DOWN ] := TEMP ;

```



27 JUL 83 08:39:01  
27 JUL 83 08:38:30  
27 JUL 83 08:38:06

040422 040423  
040424 040427  
040430 040433  
040434 140435

011715 012506 \$(0)  
012507 013115 \$(0)  
013116 013565 \$(0)  
013566 013611 \$(036)

STORAGE: 17792/4/040777/073777

\$(1)  
\$(1)  
\$(1)  
\$(3)

14,313

TIME: 0

ERRORS: 0

LEVEL

FND MAP, @USE

INFOTABLE,INFO\$

@USE PASUNDDDD,INFO\$

@SETC,D

@XOT TPF\$,MAIN

QUITCK1  
DIVIDE2  
MAIN

```

396 149 78 674 907 393 819 557 734 585 723 189 472 782 10A 962 916 397 572 710
331 114 638 311 384 641 830 207 1 703 797 388 455 845 897 357 631 368 58 56
489 927 32 697 330 33 519 936 580 928 478 757 400 22 294 945 510 292 295 950
218 769 483 461 744 453 157 940 511 718 479 847 179 482 855 267 782 148 361 96
432 528 272 946 639 679 125 342 394 823 553 828 763 206 498 989 808 991 249 437

```

```

ER TYPE 04 CODE 77 CONT 12 REENT ADR: 011152 BDI: 000004

```

```

PACKET ADR 000001 ID: 000005

```

```

PROFESSOR REQUESTED ON ADED$ CALL WAS UNAVAILABLE AT THE
TIME OF THE ER ADED$ OR WAS DOWNED BY THE SYSTEM OPERATOR.
OCCURRED IN DIVIDETWO (LINE 1)

```

```

*** WALK-BACK FROM DIVIDETWO (LINE 1) ***

```

```

DYNAMIC LEVEL: 4, STATIC LEVEL: 2.
CALLED FROM QUICKTWO (LINE 23667)
CALLED FROM QUICKONE (LINE 21649)
CALLED FROM MAIN (LINE 19631)
EXECUTION TERMINATED IN ERROR

```

\*\* UNSORTED ARRAY (1000 ELEMENTS) \*\*

674	137	901	506	283	761	416	940	896	976	311	875	367	723	967	842	993	460	703	475
835	338	788	825	743	503	814	956	746	197	193	901	36	25	881	516	889	977	262	754
436	511	486	335	328	245	388	766	885	485	474	129	462	63	269	31	313	678	717	563
366	137	117	110	890	617	459	288	144	292	837	286	679	396	1	343	70	979	899	52
440	377	465	676	240	601	687	597	328	574	402	527	506	391	47	31	243	456	979	434
427	592	807	104	572	945	511	604	996	693	0	167	91	330	63	400	894	469	548	125
756	870	909	695	12	579	163	131	142	820	559	849	998	463	203	775	288	642	130	43
579	299	241	512	685	143	803	740	406	761	97	84	161	292	115	7	171	337	75	684
420	469	836	455	748	941	171	613	975	181	551	110	593	979	175	230	974	114	40	
763	895	69	571	461	361	478	241	464	63	573	529	450	396	720	952	949	265	724	431
505	768	948	733	447	14	121	679	646	87	73	297	710	366	442	52	671	140	790	770
316	670	370	812	887	574	142	573	4	623	489	291	725	763	420	666	769	386	879	733
516	605	153	912	494	495	179	120	657	990	929	307	845	334	860	740	427	711	804	826
195	907	458	250	706	543	47	639	321	674	515	982	835	257	973	443	622	180	988	239
883	317	605	660	613	244	865	332	552	997	931	253	439	458	738	116	493	689	490	911
168	731	95	754	760	130	50	349	605	643	66	495	163	319	113	474	373	355	383	168
281	854	17	171	28	819	304	193	859	456	366	457	942	48	239	816	358	419	546	746
709	185	848	348	413	742	83	990	824	133	340	136	921	686	581	429	391	484	741	35
547	517	2	717	595	506	867	119	363	176	19	151	707	932	762	382	395	625	226	787
358	310	341	185	814	629	911	155	833	648	5	192	527	715	184	349	144	122	955	548
915	59	177	239	628	585	974	54	473	52	90	540	809	679	417	78	91	304	816	504
292	957	163	964	744	815	907	980	429	806	279	676	322	86	620	588	506	173	143	442
360	904	805	763	205	599	778	135	420	561	379	413	87	260	149	147	867	248	125	950
855	103	169	510	800	380	597	428	132	799	166	999	868	605	837	561	518	242	562	986
754	236	701	348	668	354	513	788	40	868	502	68	147	948	257	244	228	924	752	397
802	396	930	300	660	873	734	384	442	843	583	326	87	893	657	52	365	739	167	928
630	699	159	359	97	415	229	6	898	305	878	284	67	468	65	794	997	474	878	771
997	701	85	655	196	584	644	300	139	367	143	360	981	692	427	817	317	466	573	610
263	472	663	785	486	850	687	552	491	966	237	105	572	880	487	38	398	587	329	645
332	390	334	372	159	432	596	149	986	701	329	306	691	476	440	249	372	449	157	19
853	454	111	358	384	955	139	848	296	168	401	170	418	910	251	503	707	981	666	531
601	112	840	530	207	231	940	429	546	2	821	763	643	337	885	847	371	767	67	519
530	323	319	236	749	734	768	286	392	531	93	356	730	875	504	651	253	253	42	233
82	820	771	223	22	58	777	978	944	171	723	297	527	530	66	234	311	116	876	6
955	205	545	984	756	858	518	727	431	420	518	472	62	540	584	465	498	856	590	0
588	782	740	986	251	501	896	335	470	513	431	445	862	796	649	490	633	866	635	250
254	715	789	614	324	877	81	924	683	516	462	755	911	733	677	703	955	928	83	239
720	333	904	631	52	517	738	418	632	894	909	897	899	979	672	528	209	369	70	160
34	116	612	424	96	419	771	624	927	238	672	924	441	686	130	592	692	749	606	527
92	174	444	280	954	660	130	240	114	113	103	607	611	779	94	283	797	247	687	718
242	339	234	399	639	132	62	105	921	796	257	543	172	629	157	700	171	564	24	679
254	611	219	320	821	425	549	447	863	391	267	587	513	372	605	452	705	433	515	442
603	629	653	911	598	118	505	888	210	604	858	428	379	322	699	728	173	516	947	625
85	536	280	35	641	459	648	398	157	300	921	125	77	640	463	540	414	691	254	766
399	550	537	475	559	611	813	383	24	811	347	865	950	612	176	998	562	457	994	835

TIME CONSUMED : 379 MSEC

\*\* SORTED ARRAY \*\*

437	137	249	506	283	428	416	206	553	324	341	342	367	125	639	272	528	460	432	475
436	511	486	335	328	245	388	218	295	485	474	129	462	63	269	31	313	292	510	563
366	177	117	110	294	617	459	288	144	292	22	286	400	396	1	343	70	478	580	52
440	377	465	819	240	601	33	897	120	874	402	827	506	391	47	31	243	456	330	434
427	592	32	104	572	489	511	604	56	58	0	167	91	330	63	400	368	469	548	125
631	357	455	388	12	579	163	131	142	1	559	207	641	463	203	384	288	642	130	43
579	299	241	512	311	143	638	114	406	331	97	84	161	292	115	7	171	337	75	572
420	469	797	455	108	472	171	613	189	223	181	551	110	593	585	175	230	557	114	40
393	674	69	571	461	361	478	241	464	63	573	529	450	396	78	149	396	265	457	431
505	562	776	632	447	14	121	347	646	87	73	297	24	366	442	52	671	140	383	611
316	670	370	559	475	574	142	573	4	623	489	291	225	537	420	666	550	386	399	254
516	605	153	414	494	495	179	120	657	540	463	307	640	334	77	125	427	300	157	398
195	648	458	250	459	543	47	639	321	674	515	641	35	257	280	443	622	180	536	239
85	317	605	660	613	244	625	332	552	516	173	253	439	458	322	116	493	379	490	428
168	604	95	210	505	130	50	349	605	643	66	495	163	319	113	474	373	355	383	168
281	118	17	171	28	598	304	193	653	456	366	457	629	48	239	603	358	419	546	442
515	185	433	348	413	452	83	605	372	133	340	136	513	587	581	429	391	484	267	35
547	517	2	391	595	506	447	119	363	176	19	151	549	425	320	382	395	625	226	219
358	310	341	185	611	629	254	155	24	648	5	192	527	564	184	349	144	122	171	548
157	59	177	239	628	585	629	54	473	52	90	540	172	629	417	78	91	304	543	504
292	257	163	105	62	132	639	399	429	234	279	339	322	86	620	588	506	173	143	442
360	242	247	283	205	599	94	135	420	561	379	413	87	260	149	147	611	248	125	607
103	103	169	510	133	380	597	428	132	318	166	240	130	605	660	561	518	242	562	280
446	236	124	348	668	354	513	92	40	527	502	68	147	606	257	244	228	592	130	397
441	396	672	300	660	238	624	384	412	419	583	326	87	96	657	52	365	424	167	612
630	116	159	359	97	415	229	6	34	305	160	284	67	468	65	70	369	474	209	528
672	632	85	655	196	584	644	300	139	367	143	360	418	517	427	52	317	466	573	610
263	472	663	631	486	333	239	552	491	83	237	105	572	462	487	38	398	587	329	645
332	390	334	372	159	432	596	149	516	81	329	306	324	476	440	249	372	449	157	19
614	454	111	358	384	254	139	250	796	168	401	170	418	635	251	503	633	490	666	531
601	112	649	530	207	231	445	429	546	2	431	513	643	317	470	335	371	501	67	519
430	323	319	236	251	588	0	286	392	531	93	356	590	498	504	651	253	253	42	233
82	465	584	223	22	58	540	62	472	171	518	297	527	530	66	234	111	116	420	6
431	205	545	518	674	676	676	677	678	679	679	679	679	682	683	684	685	686	686	687
687	687	689	691	691	692	693	695	697	699	699	700	701	701	701	703	703	703	705	706
707	707	709	710	710	711	715	715	717	717	718	720	720	723	723	724	727	728	730	731
733	733	733	734	734	734	736	738	738	739	740	740	740	741	742	743	744	744	746	746
748	749	749	752	754	754	754	755	756	756	757	760	761	761	762	763	763	763	763	763
766	766	767	768	768	769	769	770	771	771	771	775	777	778	779	782	782	782	785	787
788	788	789	790	794	796	796	797	797	799	800	802	803	804	805	806	807	808	809	811

812	813	814	814	815	816	816	817	819	819	820	820	821	821	823	824	825	825	826	828
830	833	835	835	835	836	837	837	840	842	843	845	845	847	847	848	848	849	850	853
854	855	855	856	858	858	859	860	862	863	865	865	866	867	867	868	868	870	873	875
876	877	878	878	879	880	881	883	885	885	887	888	889	890	892	893	894	894	895	896
896	897	897	898	899	899	901	901	904	904	907	907	907	909	909	910	911	911	911	911
912	914	915	916	921	921	921	924	924	924	927	927	928	928	928	929	930	931	932	936
940	940	940	941	942	945	945	946	947	948	948	949	950	950	950	952	954	955	955	955
955	956	957	962	964	966	967	973	974	974	975	976	977	978	979	979	979	979	980	981
981	982	984	986	986	986	988	989	990	990	991	993	994	996	997	997	997	998	998	999

## E.4.1 - CASE STUDY FOUR: Gaussian Elimination (Without Processor Dedication)

```

#CPS,S,S ,CPS,S/O,
MACRO INTERPRETER LEVEL 7R1 07/28/83 11:13:09
1 *COMPILE OPTIONS=IRSLM,ELTNAME=TPFS,MAIN
1 PPROGRAM MAIN;
1 CONST N=20; NP1=21;
1 TYPE ENTRIES =(ENTRY1,ENTRY2,INIT);
1 ARRTYPE=ARRAY [1,N,1,NPI] OF REAL;
1 PROCEDURE GAUSS(ENTRY1; ENTRIES; VAR A: ARRTYPE); PROCFS;
1 VAR A: ARRTYPE;
1 I,J,K: INTEGER;
1 L,M,CP,Z: INTEGER;
1 ENTRY: ENTRIES;
1 GLOCK: QUEUE;
1 ACT1,ACT2: QUEUE;
1 WAITWD: QUEUE;
1 FINISH: QUEUE;
1 S: REAL;
1 X: ARRAY [1,N] OF REAL;
1 BFGIN
1 CREGION;
1 READLN;
1 WRITELN; WRITELN; WRITELN(' ** INPUT MATRIX ** ');
1 WRITELN; WRITELN(' 20 X 20 COEFFICIENTS AND RIGHT HAND SIDE');
1 WRITELN; WRITELN;
1 CEND;
1 FOR I := 1 TO N
1 DO BEGIN
1 CREGION; WRITELN; WRITE(' '); CEND;
1 FOR J := 1 TO (N+1)
1 DO BEGIN
1 Z := 5;
1 IF J = (N+1) THEN Z := 8;
1 CREGION; READ(A(I,J)); WRITE(A(I,J):Z:1); CEND;
1 END;
1 END;
1 CREGION; CP := CPTIME(3); CEND;
1 LOCK(WAITWD);
1 LOCK(FINISH);
1 STARTUP WITH TASKNAME(G0) GAUSS (INIT,A FORWARD);
1 LOCK(GLOCK);
1 FORK WITH TASKNAME(G1) GAUSS(ENTRY1,A);
1 WAIT(GLOCK);
1 LOCK(GLOCK);
1 FORK WITH TASKNAME(G2) GAUSS(ENTRY2,A);
1 WAIT(GLOCK);
1 WAIT(WAITWD);
1 CREGION; CP := CPTIME(3) - CP; CEND;
1 XCNJ := A(N,N+1);
1 FOR L := (N-1) DOWNTO 1
1 DO BEGIN
1 S := 0;

```

```

1         FOR J:= (L+1) TO N
1         DO BEGIN
1             S := S + X(L) * A(L,J);
1             END;
1         X(L) := A(L, N+1) = S;
1     END;
1 CREGION;
1     WRITELN; WRITELN(' TIME CONSUMED : ',CP,' MSEC');
1     WRITELN; WRITELN(' ** SOLUTION ** ');
1     WRITELN('----- '); WRITELN ;
1
1     CEND;
1     FOR I := 1 TO N
1     DO BEGIN CREGION; WRITELN(X(I)); CEND; FND;
1     LOCK(FINISH);
1     SIGNAL(FINISH);
1     QUIT;
1     END;
1 *COMPILE OPTIONS=IRSLMV,ELTNAME=TPF,GAUSS
1 PROGRAM DEFINE(GAUSS);
1 CONST N=20; NP1=21;
1 TYPE ENTRIES = (ENTRY1,ENTRY2,INIT);
1     ARRTYPE = ARRAY [1..N, 1..NP1] OF REAL;
1 PROCEDURE GAUSS(ENTRY : ENTRIES; VAR A : ARRTYPE);
1 VAR I,J,K,L,M : INTEGER;
1     PERMISSION : BOOLEAN;
1     P,Q,R : REAL ;
1 BEGIN CASE ENTRY OF
1     ENTRY1:
1     BEGIN
1     LOCK(GLOCK);
1     SIGNAL(GLOCK);
1     FOR I:= 1 TO N
1     DO BEGIN
1     P := A(I,I);
1     FOR J := I TO (N+1)
1     DO BEGIN
1     A(I,J) := A(I,J) / P ;
1     END;
1     IF I = N THEN BEGIN
1     LOCK(WAITWD);
1     SIGNAL(WAITWD);
1     QUIT;
1     END;
1     LOCK(ACT2);
1     PERMISSION := TRUE ;
1     SIGNAL(ACT2);
1     K := I+1 ;
1     WHILE K <= N
1     DO BEGIN
1     Q := A(K,I);
1     FOR J:= I TO (N+1)

```

```

1          DO BEGIN
1              ATK, JJ := ATK, JJ + ACT, JJ * 0 ;
1              END;
1          K := K + 2 ;
1          END;
1          LOCK( ACT1 );
1          IF PERMISSION THEN WAIT( ACT1 )
1              ELSE UNLOCK( ACT1 );
1          END;
1          ENTRY2:
1          BEGIN
1              LOCK( GLOCK );
1              SIGNAL( GLOCK );
1              WHILE TRUE
1                  DO BEGIN
1                      LOCK( ACT2 );
1                      IF NOT PERMISSION THEN WAIT( ACT2 )
1                          ELSE UNLOCK( ACT2 );
1                      L := I + 2 ;
1                      WHILE L <= N
1                          DO BEGIN
1                              R := ACL, IJ;
1                              FOR M := I TO ( N + 1 )
1                                  DO BEGIN
1                                      ACL, MJ := ACL, MJ + ACT, MJ * R ;
1                                  END;
1                              L := L + 2 ;
1                              END;
1                              LOCK( ACT1 );
1                              PERMISSION := FALSE;
1                              SIGNAL( ACT1 );
1                              IF I >= ( N - 1 ) THEN QUIT;
1                          END;
1          END;
1          INIT:
1          BEGIN
1              PERMISSION := FALSE;
1              RELEASE( GAUSS );
1              WAIT( FINISH );
1              QUIT;
1          END;
1          END;
1          END;
1          ACPS, S, N SCRATCH, SCRATCH1, CPS, S / 1, SCRATCH, SCRATCH2
1          MACRO INTERPPETER LEVEL 7R1 07/28/83 11:14:57
1          ACPS, S, N SCRATCH, SCRATCH2, CPS, S / 2,
1          MACRO INTERPPETER LEVEL 7R1 07/28/83 11:15:15
1          ACPS, X
1          ABU, PASCAL, S, IRSLM
1          PASCAL BRIA PLIB THURSDAY, 1983 JULY 28, 11:16:04

```



```

$ (3) 003164 003221
P%LEVEL/PASBP1 $ (1) 003222 003235
P%NUM/PASBR1 $ (1) 003236 003411 $ (2) 040067 040076 23 OCT 78 11:52:20
P%STDERR/PASRR1 $ (1) 003412 003461 17 JUL 78 13:55:46
$ (3) 003462 003515 10 DEC 79 17:45:17
P%TRACE/PASBP1 $ (1) 003516 003671 23 OCT 78 11:52:21
$ (3) 003672 003724
P%WRITE/PASBP1 $ (1) 003725 005125 $ (2) 040077 040212 11 DEC 79 16:52:28
$ (3) 005126 005152
P%INTRPT/PASRR1 $ (1) 005153 005742 $ (2) 040213 040313 23 APR 82 10:37:37
$ (3) 005743 006037
P%RS/PASRR1 $ (1) 006040 007176 $ (2) 040314 040417 23 JUL 80 13:49:59
$ (3) 007177 007406
P%IO/PASRR1 $ (1) 007407 010717 23 APR 82 10:40:32
$ (3) 010720 011122
GAUSS $ (1) 011123 011731 $ (0) 040420 040420 28 JUL 83 11:17:29
MAIN $ (1) 011732 012567 $ (0) 040421 040424 28 JUL 83 11:17:01
$ (3) 012570 012621 $ (036) 040425 046425

```

SYSS\*RLIR% LEVEL

END MAP, ERRORS: 0 TIME: 13,806 STORAGE: 17792/4/040777/073777

0USE INFOTABLE,INFO\$,

0USE PASUPDDUO,INFO\$,

0SFIC,D

0XOT TPF\$,MAIN

\*\* INPUT MATRIX \*\*

20 X 20 COEFFICIENTS AND RIGHT HAND SIDE

9.26	5.4	9.7	1.6	9.2	8.8	3.4	0.3	1.0	3.6	2.5	8.2	4.5	6.4	8.6	9.2	1.5	0.1	4.2	8.8	425.1
1.6	9.4	0.3	5.4	3.5	3.7	2.6	5.1	8.4	2.9	3.8	1.9	1.3	6.3	1.9	1.7	3.4	1.0	5.6	1.1	314.5
7.0	4.9	3.9	0.7	7.8	1.0	7.2	0.0	5.1	0.7	2.3	3.1	6.4	8.4	4.6	4.5	6.4	1.2	5.9	2.7	349.0
5.3	9.8	6.1	8.5	6.3	3.6	7.4	9.5	4.9	2.7	9.5	3.1	4.4	4.7	6.5	3.6	8.4	1.1	9.2	9.5	514.6
3.3	0.3	7.0	0.8	8.2	3.1	9.5	1.6	3.8	7.7	7.4	4.0	3.2	6.3	2.2	2.7	9.5	7.2	3.7	1.5	384.6
6.6	7.8	1.5	6.8	9.3	7.8	7.7	4.5	9.2	5.6	2.9	8.3	0.2	7.3	0.9	9.5	7.1	9.5	2.9	5.1	508.0
8.6	0.0	7.7	7.4	2.9	6.8	4.7	6.1	7.4	9.6	0.6	2.3	0.4	2.1	7.9	8.7	2.1	5.3	5.1	1.0	334.7
7.7	6.3	9.4	3.0	0.2	7.2	3.1	3.1	6.2	4.8	9.0	1.1	2.3	8.0	0.8	8.7	6.1	5.6	6.2	5.2	458.4
3.2	2.5	0.8	0.6	3.7	7.5	1.5	3.9	0.3	4.3	4.9	1.9	5.6	5.8	5.8	9.3	9.0	4.9	2.7	5.0	353.4
8.8	7.3	9.5	9.0	2.1	8.9	7.6	1.0	9.4	1.1	2.4	0.1	3.4	3.6	1.2	4.4	2.8	4.7	4.9	7.9	382.2
5.0	5.7	6.1	4.5	1.4	0.7	8.4	5.6	0.4	0.3	5.7	4.0	0.6	4.9	2.9	1.4	3.9	8.9	6.4	2.9	333.6
1.0	8.5	2.3	1.3	0.7	2.6	3.0	8.8	3.0	2.4	3.4	1.8	0.1	8.2	9.4	8.4	2.1	8.6	5.1	9.5	390.4
1.5	1.7	9.7	8.8	0.4	1.0	5.6	2.7	0.0	8.5	7.1	3.9	3.2	1.7	6.7	2.8	9.6	9.3	0.0	4.7	322.6
6.8	0.1	7.6	8.4	3.1	0.6	0.1	1.7	1.0	6.1	6.7	8.9	8.6	1.6	9.2	9.5	8.8	8.4	4.1	8.3	419.6
5.8	8.6	0.8	8.8	2.9	5.7	3.2	2.6	2.1	3.1	5.5	4.8	8.6	7.4	4.2	5.2	2.9	0.0	3.4	2.6	378.7
6.0	1.1	1.1	5.5	2.9	8.3	5.2	2.5	0.4	5.8	8.2	8.0	7.5	4.3	8.4	1.9	0.9	8.5	9.0	9.1	421.4
3.9	6.6	5.2	4.0	2.3	6.2	2.4	0.5	7.8	1.9	0.4	3.1	7.3	5.4	5.4	6.7	2.7	5.3	3.7	6.0	348.5
4.9	5.6	0.6	3.8	9.7	8.4	3.7	7.8	6.3	0.7	5.4	2.3	4.1	2.9	9.9	7.8	6.1	4.1	7.0	4.5	427.4
6.3	2.0	4.4	4.4	4.3	3.4	5.5	1.9	4.9	1.8	9.6	0.3	0.8	6.0	6.0	1.8	7.7	2.6	4.2	5.1	335.9

3.1 8.5 6.7 8.2 9.7 3.6 6.4 6.1 4.6 4.1 4.5 9.2 9.3 3.3 3.8 1.5 0.0 6.1 5.7 2.7 419.3  
TIME CONSUMED : 451 MSEC

\*\* SOLUTION \*\*

-----  
1,348940  
~~1,348940~~  
2,388854  
1,755760  
2,180330  
4,095750  
4,803126  
3,328051  
4,114559  
2,013716  
7,999725  
6,916009  
3,315470  
5,775178  
1,406205  
7,269488  
3,910773  
2,801161  
5,863460  
3,406344

## E.4.2 - CASE STUDY FOUR: Gaussian Elimination (With Processor Dedication)

```

1      FOR J:= (L+1) TO N
1      DO BEGYN
1          S := S + X(L,J) * A(L,J);
1      END;
1      X(L,J) := A(L, N+1) - S;
1      END;
1  CREGION;
1      WRITELN; WRITELN(' TIME CONSUMED : ',CP,' MSEC');
1      WRITELN; WRITELN(' ** SOLUTION ** ');
1      WRITELN('-----'); WRITELN;
1  CEND;
1  FOR I := 1 TO N
1  DO BEGIN CREGION; WRITELN(X(I)); CEND; END;
1  LOCK(FINISH);
1  SIGNAL(FINISH);
1  QUIT;
1  END;
1  *COMPILE OPTIONS=IRSLMV,ELTNAME=TPF%,GAUSS
1  PROGRAM DEFINE(GAUSS);
1  CONST N=20; NP1=21;
1  TYPE ENTRIES = (ENTRY1,ENTRY2,INIT);
1  ARRTYPE = ARRAY [1..N, 1..NP1] OF REAL;
1  PROCEDURE GAUSS(ENTRY : ENTRIES; VAR A : ARRTYPE);
1  VAR I,J,K,L,M : INTEGER;
1  PERMISSION : BOOLEAN;
1  P,Q,R : REAL;
1  BEGIN CASE ENTRY OF
1  ENTRY1:
1  BEGIN
1  LOCK(GLOCK);
1  SIGNAL(GLOCK);
1  DEDICATE PROCESSOR(0);
1  FOR I:= 1 TO N
1  DO BEGIN
1  P := A(I,I);
1  FOR J := I TO (N+1)
1  DO BEGIN
1  A(I,J) := A(I,J)/ P;
1  END;
1  IF I = N THEN BEGIN
1  LOCK(WAITWD);
1  SIGNAL(WAITWD);
1  QUIT;
1  END;
1  LOCK(ACT2);
1  PERMISSION := TRUE;
1  SIGNAL(ACT2);
1  K := I+1;
1  WHILE K <= N
1  DO BEGIN
1  Q := A(K,I);

```

```

1
1      FOR J:= 1 TO (N+1)
1      DO BEGIN
1          ACK,JJ := ACK,JJ + ACT,JJ* 0 ;
1          END;
1          K := K +2 ;
1          END;
1          LOCK(ACT1);
1          IF PERMISSION THEN WAIT(ACT1)
1          ELSE UNLOCK(ACT1);
1      END;
1  END;
1  ENTRY2:
1  BEGIN
1  LOCK(GLOCK);
1  SIGNAL(GLOCK);
1  DEDICATE PROCESSOR(1);
1  WHILE TRUE
1  DO BEGIN
1  LOCK(ACT2);
1  IF NOT PERMISSION THEN WAIT(ACT2)
1  ELSE UNLOCK(ACT2);
1
1  L := I+2 ;
1  WHILE L <= N
1  DO BEGIN
1      R := ACL,IJ;
1      FOR M:= 1 TO (N+1)
1      DO BEGIN
1          ACL,MJ := ACL,MJ + R ;
1          END;
1      L := L+2 ;
1      END;
1  LOCK(ACT1);
1  PERMISSION := FALSE;
1  SIGNAL(ACT1);
1  IF I >= (N-1) THEN QUIT;
1  END;
1  END;
1  INIT:
1  BEGIN
1  PERMISSION := FALSE;
1  RELEASE(GAUSS);
1  WAIT(FINISH);
1  QUIT;
1  END;
1  END;
1  END;
1
@CPS=S,N SCRATCH$,SCRATCH1,CPS=S/1,SCRATCH$,SCRATCH2
MACRO INTERPRETER LEVEL 7R1 07/28/83 11:19:33
@CPS=S,N SCRATCH$,SCRATCH2,CPS=S/2,
MACRO INTERPRETER LEVEL 7R1 07/28/83 11:19:58
@CPS=X

```

```

P&RS/PAS&RI      $(1)   006040 007176      $(2)   040314 040417      23 JUL 80   13:49:59
                  $(3)   007177 007406
P&IO/PAS&RI      $(1)   007407 010717      23 APR 82   10:40:32
                  $(3)   010720 011122
GAUSS             $(1)   011123 011741      $(0)   040420 040420      28 JUL 83   11:22:37
MAIN             $(1)   011742 012577      $(0)   040421 040424      28 JUL 83   11:22:03
                  $(3)   012600 012631      $(036) 040425 046425

```

```

SYS&RLIB& LEVEL
END MAP, ERRORS: 0 TIME: 13.953 STORAGE: I7792/4/040777/073777
&USE INFOTABLE,INFOS
&USE PASUNDDDD,INFOS
&SETC,D
&XQT TPFS,MAIN

```

\*\* INPUT MATRIX \*\*

20 X 20 COEFFICIENTS AND RIGHT HAND SIDE

9.76	5.4	9.7	1.6	9.2	8.8	3.4	0.3	1.0	3.6	2.5	9.2	4.5	6.4	8.6	9.2	1.5	0.1	4.2	8.8	425.1
1.6	9.4	0.3	5.4	3.5	3.7	2.6	5.1	8.4	2.9	3.8	1.9	1.3	6.3	1.9	1.7	3.4	1.0	5.6	1.1	314.5
7.0	4.9	3.9	0.7	7.8	1.0	7.2	0.0	5.1	0.7	2.3	3.1	6.4	8.4	4.6	4.5	6.4	1.2	5.9	2.7	349.0
5.3	9.8	6.1	8.8	6.3	3.6	7.4	9.8	4.9	2.7	9.8	1.1	4.4	4.7	8.9	1.6	8.4	1.1	9.2	9.8	414.8
3.3	0.8	7.0	0.8	8.2	3.1	9.5	1.6	3.8	7.7	7.4	4.0	3.2	6.3	2.2	2.7	9.5	7.2	1.7	3.4	384.8
6.6	7.8	1.5	6.8	9.3	7.8	7.7	4.5	9.2	5.6	2.9	8.3	0.2	7.3	0.9	9.5	7.1	9.5	2.9	5.1	508.0
8.6	0.8	7.7	7.4	2.9	6.8	4.7	6.1	7.4	9.6	0.6	2.3	0.4	2.1	7.9	8.7	2.1	5.3	5.1	1.0	334.7
7.7	6.3	9.4	3.0	0.2	7.2	3.1	3.1	6.2	4.8	9.0	1.1	2.3	8.0	0.8	8.7	6.1	5.6	6.2	5.2	458.4
3.2	2.5	0.8	0.6	3.7	7.5	1.5	3.9	0.3	4.3	4.9	1.9	5.6	5.8	5.8	9.3	9.0	4.9	2.7	5.0	353.4
8.8	7.3	9.5	9.0	2.1	8.9	7.6	1.0	9.4	1.1	2.4	0.1	3.4	3.6	1.2	4.4	2.8	4.7	4.9	7.9	382.2
5.0	5.2	6.1	4.5	1.4	0.7	8.4	5.6	0.4	0.3	5.7	4.0	0.6	4.9	2.9	1.4	3.9	8.9	6.4	2.9	333.6
1.0	8.5	2.3	1.3	0.7	2.6	3.0	8.8	3.0	2.4	3.4	1.8	0.1	8.2	9.4	8.4	2.1	8.6	5.1	9.5	390.4
1.5	1.7	9.7	8.8	0.4	1.0	5.6	2.7	0.0	8.5	7.1	3.9	3.2	1.7	6.7	2.8	9.6	9.3	0.0	4.7	322.6
6.8	0.8	7.6	8.4	3.1	0.6	0.1	1.7	1.0	6.1	6.7	8.9	8.6	1.6	9.2	9.5	8.8	8.4	4.1	8.3	419.6
5.8	8.6	0.8	8.8	2.9	5.7	3.2	2.6	2.1	3.1	5.5	4.8	8.6	7.4	4.2	5.2	2.9	0.0	3.4	2.6	378.7
6.0	1.1	1.1	5.5	2.9	8.3	5.2	2.5	0.4	5.8	8.2	8.0	7.5	4.3	8.4	1.9	0.9	8.5	9.0	9.1	421.4
3.9	6.6	5.2	4.0	2.3	6.2	2.4	0.5	7.8	1.9	0.4	3.1	7.3	5.4	5.4	6.7	2.7	5.3	3.7	6.0	348.5
4.9	5.6	0.6	3.8	9.7	8.4	3.7	7.8	6.3	0.7	5.4	2.3	4.1	2.9	9.9	7.8	6.1	4.1	7.0	4.5	427.4
6.3	2.0	4.4	4.4	4.3	3.4	5.5	1.9	4.9	1.8	9.6	0.3	0.8	6.0	6.0	1.8	7.7	2.6	4.2	5.1	335.9
3.1	8.5	6.7	8.2	9.7	3.6	6.4	6.1	4.6	4.1	4.5	8.2	9.3	3.3	3.8	1.5	0.0	6.1	5.7	2.7	419.3

```

ER TYPE 04 CODE 77 CONT 12 REENT ADR: 011513 BDI: 000004
PACKET ADR 000001 ID: 000003
PROCESSOR REQUESTED ON ADED& CALL WAS UNAVAILABLE AT THE
TIME OF THE FR ADED& OR WAS DOWNED BY THE SYSTEM OPERATOR.
OCCURRED IN GAUSS (LINE 1)
*** WALKBACK FROM GAUSS (LINE 1) ***
DYNAMIC LEVEL: 2, STATIC LEVEL: 2,
CALLED FROM MAIN (LINE 19506)
EXECUTION TERMINATED IN ERROR
AD AMBIGUITY ABO&T ADR: 011472 BDI:000004

```

ID: 000002  
AWAITS - DACTS AMBIGUITY:

X 000000 000156 000000 000052 000001 000005 000001 060663 000000 007045 000000 064061 000000 000000 000001 000001 000003  
 000000 000000 040425 000000 046062 000000 000002 000000 000000 000026 000000 000024 463146 314632 201042 242631  
 A 000000 000026 000000 000024 463146 314632 201042 242631 000000 040470 000000 001211 777777 777776 000000 000001  
 000000 000071 000000 000054 000000 040470 000000 000040 000000 000052 000000 046424 000000 000001 000000 000024  
 000000 000000 000000 000000

R 000000 000000 000000 000040 073423 120010 000004 121746 000000 066243 000000 000064 000000 040470 000000 000003  
 000000 000000 000000 000000 000000 000000 000000 000000 000000 444444 000000 000002 050505 050505 000000 000134  
 ABORT ADR: 012310 BDI:000004  
 X 000000 000156 000000 044264 000001 000005 000001 060663 000000 007045 000000 064061 000000 000000 000001 000003  
 000000 000000 040425 000000 046425 000000 012220 000000 046062 000000 046062 000000 000002 000000 000002  
 A 000000 046062 000000 000002 000000 046425 000000 000002 000000 001211 000000 001211 777777 777776 000000 000001  
 000000 000071 000000 000054 000000 040470 000000 000040 000000 040352 000000 046426 000000 000001 000000 001000  
 000000 000000 000000 000000

R 000000 000000 000000 000040 073423 120010 000004 121746 000000 066243 000000 000064 000000 040470 000000 000003  
 000000 000000 000000 000000 000000 000000 000000 000000 000000 444444 000000 000000 050505 050505 000000 046062  
 ABORT ADR: 011727 BDI:000004  
 ID: 000001  
 X 000000 000156 000000 000052 000001 000005 000001 060663 000000 007045 000000 064061 000000 000000 000001 000003  
 000000 000000 040425 000000 046062 000000 012220 000000 000002 000000 000002 000000 011132 000000 040425 000001 006430  
 A 000000 000002 000000 011132 000000 040425 000001 006430 000000 005464 000000 000314 777777 777776 000000 000001  
 000000 000071 000000 000054 000000 040470 000000 000040 000000 040352 000000 046426 000000 000001 000000 000000  
 000000 000000 000000 000000

R 000000 000000 000000 000040 073423 120010 000004 121746 000000 066243 000000 000064 000000 040470 000000 000003  
 000000 000000 000000 000000 000000 000000 000000 000000 000000 444444 000000 000001 050505 050505 000000 030254

## E.5 - CASE STUDY FIVE: A Model Operating System

```

@FREE TPF$.
@ASS,T TPF$,F///1500
@CPS,S,S ,CPS,5/0,
MACRO INTERPRETER LEVEL 7R1 07/31/83 19:42:01
1 *COMPILE OPTIONS=IRSLMG,ELTNAME=TPF$,MAIN
1 PROGRAM MAIN(INPUT,OUTPUT);
1 TYPE ARRTYP1 = ARRAY [ 0 .. 19 ] OF INTEGER ;
1 ARRTYP2 = ARRAY [ 1 .. 20 ] OF INTEGER ;
1 MOSENTRIES =(MOSINIT,CLOCKSTART,REGULARINT,IOINITIATE,IOTERMINATE);
1 ARRGENTRIES =(ARRGINIT ,ARRGLIST);
1 PROCEDURE MOS( VAR MOSENTRY : MOSENTRIES ); PROCESS;
1 PROCEDURE ARRANGE( VAR ARRANGENTRY : ARRGENTRIES ; VAR LIST : ARRTYP1;
1 LISHEAD : INTEGER; LISTAIL : INTEGER ; VAR SORTKEY : ARRTYP2); PROCESS;
1 VAR MOSLOCK : QUEUE ;
1 CLOCKLOCK : QUEUE ;
1 ACTIVATOR : QUEUE ;
1 FINISH : QUEUE ;
1 ARRANGELOCK : QUEUE ;
1 ARRFLAG : QUEUE ;
1 CLOCKINTRPT : QUEUE ;
1 MASTERLOCK : IOINITIATION LOCK OF QUEUE ;
1 TERMLOCK : IOTERMINATION LOCK OF QUEUE ;
1 STARTER : QUEUE ;
1 IOINIT : IOINITIATION FLAG OF QUEUE ;
1 IOTERM : IOTERMINATION FLAG OF QUEUE ;
1 EMPTYRQ : QUEUE ;
1 I : INTEGER;
1 MOSENTRY : MOSENTRIES;
1 ARRANGENTRY : ARRGENTRIES ;
1 DUMINT : INTEGER ;
1 DUMARR1 : ARRTYP1 ;
1 DUMARR2 : ARRTYP2 ;
1 BEGIN
1 I := 1;
1 WHILE I <= 18 DO BEGIN
1 STARTUP SLAVETASK(I);
1 I := I + 1;
1 END;
1
1 ARRANGENTRY := ARRGINIT ;
1 STARTUP WITH TASKNAME(A00)
1 ARRANGE(ARRANGENTRY,DUMARR1 FORW,DUMINT FORW,DUMINT FORW,DUMARR2 FORW);
1
1 MOSENTRY := MOSINIT ;
1 STARTUP WITH TASKNAME(T00) MOS(MOSENTRY);
1
1
1 LOCK(MOSLOCK);
1 MOSENTRY := CLOCKSTART;
1 FORK WITH TASKNAME(T01) MOS(MOSENTRY);
1 WAIT(MOSLOCK);

```

```

1
1 LOCK(MOSLOCK);
1 MOSENTRY := REGULARINT;
1 FORK WITH TASKNAME(T02) MOS(MOSENTRY);
1 WAIT(MOSLOCK);
1
1 LOCK(MOSLOCK);
1 MOSENTRY := IOINITIATE;
1 FORK WITH TASKNAME(T03) MOS(MOSENTRY);
1 WAIT(MOSLOCK);
1
1 LOCK(MOSLOCK);
1 MOSENTRY := IOTERMINATE;
1 FORK WITH TASKNAME(T04) MOS(MOSENTRY);
1 WAIT(MOSLOCK);
1
1 LOCK(STARTER);
1 SIGNAL(STARTER);
1 LOCK(FINISH);
1 WAIT(FINISH);
1 QUIT;
1
1 END.
1 *COMPILE OPTIONS=IPSLMVG,ELTNAME=TPFS,ARRANG
1 PROGRAM DEFINE(ARRANGE);
1
1 TYPE ARRTYP1 = ARRAY [ 0 .. 19 ] OF INTEGER ;
1 ARRTYP2 = ARRAY [ 1 .. 20 ] OF INTEGER ;
1 ARGENTRIES = ( ARGINIT , ARRGLIST ) ;
1 PROCEDURE ARRANGE (VAR ENTRY : ARGENTRIES ; VAR LIST : ARRTYP1;
1 LISHEAD : INTEGER; LISTAIL : INTEGER ; VAR SORTKEY : ARRTYP2);
1
1 VAR I,J,K : INTEGER ;
1 TEMP : INTEGER ;
1
1 BEGIN
1 CASE ENTRY OF
1 ARRGLIST:
1 BEGIN
1 LOCK(ARRFLAG);
1 SIGNAL(ARRFLAG);
1 J := (LISTAIL+20-1) MOD 20;
1 I := (LISTAIL+20-2) MOD 20;
1 WHILE ( J <> LISHEAD ) AND
1 ( SORTKEY [ LISTE J ] < SORTKEY [ LISTE I ] )
1 DO BEGIN
1 TEMP := LISTE J ; LISTE J := LISTE I ; LISTE I := TEMP ;
1 I := ( I+20-1 ) MOD 20 ;
1 J := ( J+20-1 ) MOD 20 ;
1 END;
1
1 QUIT;
1 END;
1 ARGINIT:
1 BEGIN

```



```

1      RELEASE(ARRANGE);
1      LOCK(FINISH);
1      WAIT(FINISH);
1      END;
1      FND;
1      END;
1      *COMPILE      OPTIONS=IPSLMVG,ELTNAM=TPF%,MODELOS
1      PROGRAM DEFINE(MOS);
1      TYPE MOSENTRIES =(MOSINIT,CLOCKSTART,REGULARINT,IOTINITIATE,IOTERMINATE);
1      ARRGENTRIES =( ARRGINIT , ARRGLIST) ;
1      ARRTYP1 = ARRAY [ 0 .. 19 ] OF INTEGER ;
1      ARRTYP2 = ARRAY [ 1 .. 20 ] OF INTEGER ;
1      PROCEDURE MOS(VAR ENTRY : MOSENTRIES);
1      PROCEDURE ARRANGE(VAR ARRANGENTRY : ARRGENTRIES ; VAR LIST : ARRTYP1;
1      LISHEAD : INTEGER;LISTAIL : INTEGER ;VAR SORTKEY : ARRTYP2); PROCESS;
1      TYPE SYSTEMTABLE = RECORD
1      CCU : INTEGER;
1      CIOJ : INTEGER;
1      FND;
1      VAR LASTINTRPT : INTEGER ;
1      LASTSWAP : INTEGER ;
1      CLOCK : INTEGER ;
1      Y,J,M,II : INTEGER ;
1      RQHEAD,ROTAIL : INTEGER ;
1      BQHEAD,BOTAIL : INTEGER ;
1      DIOQHEAD : INTEGER ;
1      DIOQTAIL : INTEGER ;
1      MAXRQLEN : INTEGER ;
1      RQLIST : ARRTYP1 ;
1      RQWAIT : ARRTYP2 ;
1      BQLIST : ARRTYP1 ;
1      DIOQLIST : ARRTYP1 ;
1      LRS : INTEGER ;
1      SHIFT : INTEGER ;
1      MAXWAIT : INTEGER ;
1      CONTROLTABLE : SYSTEMTABLE;
1      TIME : ARRAYC [0..100] OF INTEGER;
1      CPUUSER : ARRAYC [0..100] OF INTEGER;
1      DIOJOB : ARRAYC [0..100] OF INTEGER;
1      MEMIDLE : ARRAYC [0..100] OF INTEGER;
1      READYQUE : ARRAYC [0..100,0..19] OF INTEGER;
1      BLOCKEDQUE : ARRAYC [0..100,0..19] OF INTEGER;
1      DIOQUE : ARRAYC [0..100,0..19] OF INTEGER;
1      PRIORITY : ARRTYP2 ;
1      STARTTIME : ARRAYC [0..20] OF INTEGER;
1      MEMREQ : ARRAYC [0..20] OF INTEGER;
1      STATUS : ARRAYC [0..20] OF BOOLEAN ;
1      MEMUNUSED : INTEGER;
1      TIMEINCREMENT : INTEGER;
1      TIMEDECREMENT : INTEGER;
1      YEMP : INTEGER;

```

```

1      MEMSWAPABLE : INTEGER;
1      INFOTABLE   : TEXT   ;
1      ARRANSETRY  : ARRGENTRIES ;
1      QUANTUM     : ARRTYP2  ;
1      QUANTMAX    : INTEGER  ;
1      CPUIDLE     : INTEGER  ;
1      CPUIDLECOM  : REAL     ;
1      MEMIDLECOM  : REAL     ;
1      MEMUTILIZE  : REAL     ;
1      CPUUTILIZE  : REAL     ;
1
1      BEGIN
1      CASE ENTRY OF
1      CLOCKSTART:
1      BEGIN
1      LOCK(MOSLOCK);
1      SIGNAL(MOSLOCK);
1      WHILE TRUE DO BEGIN
1      SUSPEND(50);
1      LOCK(CLOCKLOCK);
1      CLOCK := CLOCK + 50;
1      UNLOCK(CLOCKLOCK);
1      IF CLOCK >= 3000 THEN TERMINATE;
1      LOCK(CLOCKINTRPT);
1      SIGNAL(CLOCKINTRPT);
1      END;
1      END;
1      REGULARINT:
1      BEGIN
1      LOCK(MOSLOCK);
1      SIGNAL(MOSLOCK);
1      WHILE TRUE
1      DO BEGIN
1      LOCK(CLOCKINTRPT);
1      WAIT(CLOCKINTRPT);
1      IF ( RQTAIL = RQHEAD ) AND
1      ( CONTROLTABLE[CCU] = 0 )
1      THEN BEGIN
1      LOCK(EMPTYRQ);
1      WAIT(EMPTYRQ);
1      END;
1      LOCK(MASTERLOCK);
1      LOCK(CLOCKLOCK);
1      IF ( BQHEAD <> BQTAIL )
1      AND (MEMREQ[BOLISTEBQHEAD]] <= MEMSWAPABLE) AND ((CLOCK-LASTSWAP) >= 500)
1      THEN BEGIN
1      CREGION;
1      WRITELN(' SWAPPING ');
1      CEND;
1      LASTSWAP := CLOCK;
1      WHILE (RQTAIL <> RQHEAD) AND

```

```

1      (MEMREQBQLISTCBOHEADJ) >= MEMUNUSED)
1      DO BEGIN
1          LRO := ( 20+RQTAIL-ROHEAD ) MOD 20;
1          MAXWAIT := RWAITCRLISTEROHEADJJ; J := ROHEAD;
1          IF LRO > 1
1              THEN BEGIN
1                  FOR I:=1 TO (LRO-1) DO
1                      BEGIN;
1                          K := (ROHEAD+I) MOD 20;
1                          IF MAXWAIT > RWAITCRLISTC K JJ
1                              THEN BEGIN
1                                  MAXWAIT := RWAITCRLISTC K JJ;
1                                  J := K;
1                                  END;
1                          END;
1                  END;
1          MEMUNUSED:=MEMUNUSED+MEMREQCRLISTCJJJ;
1          MEMSWAPABLE:=MEMSWAPABLE-MEMREQCRLISTCJJJ;
1          BQLISTC BOTAIL JJ:= RQLISTC J J;
1          BOTAIL :=(BOTAIL+1) MOD 20;
1          LOCK(ARRANGFLOCK);
1          LOCK(ARRFLAG);
1          ARRANGENTRY := ARRGLIST ;
1          FORK WITH TASKNAME(A01) ARRANGE(ARRANGENTRY,BOLIST,BQHEAD,BOTAIL,QUANTUM);
1          WAIT(ARRFLAG);
1          AWAIT(A01);
1          UNLOCK(ARRANGFLOCK);
1          SHIFT :=(RQTAIL - J +19) MOD 20 ;
1          IF SHIFT > 0
1              THEN BEGIN
1                  FOR I:=1 TO SHIFT DO
1                      BEGIN;
1                          K := I+J-1;
1                          RQLISTC K MOD 20 := RQLISTC(K+1) MOD 20;
1                          END;
1                  END;
1          END;
1          RQTAIL :=(RQTAIL+20-1) MOD 20;
1          END;
1      WHILE (MEMREQBQLISTCBOHEADJJ <= MEMUNUSED)
1          AND (BOHEAD <> BQTAIL)
1          DO BEGIN
1              RQLISTC RQTAIL JJ := BQLISTCBOHEADJJ;
1              RWAITCRLISTERQTAILJJ := CLOCK;
1              RQTAIL :=(RQTAIL+1) MOD 20;
1              LOCK(ARRANGFLOCK);
1              LOCK(ARRFLAG);
1              ARRANGENTRY := ARRGLIST ;
1              FORK WITH TASKNAME(A06) ARRANGE(ARRANGENTRY,RQLIST,ROHEAD,RQTAIL,QUANTUM);
1              WAIT(ARRFLAG);

```

```

1      AWAIT(A06);
1      UNLOCK(ARRANGELOCK);
1      MEMUNUSED:=MEMUNUSED+MEMREQ(BQLISTCBOHEAD);
1      MEMSWAPABLE:=MEMSWAPABLE+MEMREQ(BQLISTCBOHEAD);
1      BQLISTCBOHEAD := 0;
1      BQHEAD :=(BQHEAD+1) MOD 20;
1      END;
1      END;
1      IF ((CLOCK-TIMEDECREMENT)>= 50) OR(TIMEDECREMENT=0) THEN
1      BEGIN
1      TIMEDECREMENT := CLOCK;
1      WITH CONTROLTABLE
1      DO BEGIN
1      IF CCU <> 0 THEN QUANTUM CCU J := QUANTUM CCU J + 1 ;
1      IF ( RQTAIL <> RQHEAD )
1      THEN BEGIN
1      IF( CCU <> 0 ) AND ((QUANTUM CCU J MOD QUANTMAX) = 0 )
1      THEN BEGIN
1      DEACTIVATE SLAVETASK(CCU);
1      CREGION;
1      WRITELN('                REGULAR INTRPT');
1      CEND;
1      FETCH STATUS OF SLAVETASKS;
1      IF STATUS CCU J
1      THEN BEGIN
1      RQLISTERQTAILJ:=CCU;
1      ROWAIT(RQLISTERQTAILJ):=CLOCK;
1      RQTAIL:=(RQTAIL+1) MOD 20;
1      LOCK(ARRANGELOCK);
1      ARRANGENTRY := ARRGLIST ;
1      LOCK(ARRFLAG);
1      FORK WITH TASKNAME(A07)
1      ARRANGE(ARRANGENTRY,RQLIST,RQHEAD,RQTAIL,QUANTUM);
1      WAIT(ARRFLAG);
1      AWAIT(A02);
1      UNLOCK(ARRANGELOCK);
1      END;
1      MEMSWAPABLE:=MEMSWAPABLE+MEMREQ(CCU);
1      CCU := 0 ;
1      IF RQTAIL =RQHEAD THEN CPUIDLE := CLOCK;
1      END;
1      END;
1      IF(RQTAIL <> RQHEAD) AND ( CCU = 0)
1      THEN BEGIN
1      CCU := RQLISTERQHEADJ;
1      RQLISTERQHEADJ := 0;
1      RQHEAD :=(RQHEAD+1) MOD 20;
1      MEMSWAPABLE:=MEMSWAPABLE+MEMREQ(CCU);
1      ACTIVATE SLAVETASK(CCU);
1      END;
1      END;
1      END;

```

```

1                                     END;
1 /C
1 /C ** DUMPING SECTION **
1 /C
1 II := II+1;
1 IF II <= 100
1 THEN BEGIN
1     TIMEC[II]:= CLOCK; CPUUSERC[II]:= CONTROLTABLE,CCU;
1     DIOJOB[II]:= CONTROLTABLE,CIOJ;
1     FOR I:= 0 TO 19 DO BEGIN
1         READYQUE[II,I]:= RQLISTC[I];
1         BLOCKEDQUE[II,I]:= BQLISTC[I];
1         DIOQUE[II,I]:= DIOQLISTC[I];
1         MEMIDLEC[II]:= MEMUNUSED;
1     END;
1     IF II>1 THEN MFIDLECCUM := MEMIDLECCUM+MEMUNUSED*(TIMEC[II]-TIMEC[II-1]);
1 END;
1 ELSE BEGIN
1     CPUIDLECCUM := 0; MEMIDLECCUM:= 0;
1     FOR II:= 1 TO 100 DO BEGIN
1         CREGION;
1         WRITELN;
1         WRITELN(' CLOCK ',TIMEC[II],' CCU ',CPUUSERC[II],' DIOJ ',DIOJOB[II],' IDLE STORE ',MEMIDLEC[II]);
1         WRITELN; WRITE(' RQUEUE ');
1         CEND;
1         FOR I :=0 TO 19 DO BEGIN CREGION; WRITE(READYQUE[II,I]:3); CEND; END;
1         CREGION; WRITELN; WRITE(' BQUEUE '); CEND;
1         FOR I :=0 TO 19 DO BEGIN CREGION; WRITE(BLOCKEDQUE[II,I]:3); CEND; END;
1         CREGION; WRITELN; WRITE(' DIOQUE '); CEND;
1         FOR I :=0 TO 19 DO BEGIN CREGION; WRITE(DIOQUE[II,I]:3);CEND; END;
1         CREGION; WRITELN; CEND;
1     END;
1     II := 0;
1 END;
1 /C
1 /C ** END OF DUMPING SECTION **
1 /C
1     UNLOCK(CLOCKLOCK);
1     UNLOCK(MASTERLOCK);
1 END;
1 END;
1 IOTINITIATE:
1 BEGIN
1     LOCK(MOSLOCK);
1     SIGNAL(MOSLOCK);
1     WHILE TRUE
1     DO BEGIN
1         WAIT(IOINIT);
1         LOCK(IOINIT);
1         CREGION;
1         WRITELN('
1                                     I/O STARTED');

```

```

1      CEND;
1      LOCK(CLOCKLOCK);
1      WITH CONTROLTABLE
1      DO BEGIN
1          DIOQLIST(DIOQTAIL) := CCU;
1          DIOQTAIL := (DIOQTAIL+1) MOD 20;
1          IF RQHEAD <> RQTAIL
1              THEN BEGIN
1                  CCU := RQLIST(RQHEAD);
1                  RQLIST(RQHEAD) := 0;
1                  RQHEAD := (RQHEAD+1) MOD 20;
1                  ACTIVATE SLAVETASK(CCU);
1              END
1          ELSE BEGIN
1              CCU := 0;
1              CPUIDLE := CLOCK;
1          END;
1          IF CIOJ = 0
1              THEN BEGIN
1                  CIOJ := DIOQLIST(DIOQHEAD);
1                  DIOQLIST(DIOQHEAD) := 0;
1                  DIOQHEAD := (DIOQHEAD+1) MOD 20;
1                  ACTIVATE SLAVETASK(CIOJ);
1              END;
1          FND;
1      /C
1      /C  ** DUMPING SECTION **
1      /C
1      II := II+1;
1      IF II <= 100
1          THEN BEGIN
1              TIMEC(II) := CLOCK; CPUUSERC(II) := CONTROLTABLE.CCU;
1              DIOJOB(II) := CONTROLTABLE.CIOJ;
1              FOR I := 0 TO 19 DO BEGIN
1                  READYQUE(II,I) := RQLIST(I);
1                  BLOCKEDQUE(II,I) := BQLIST(I);
1                  DIOQUE(II,I) := DIOQLIST(I);
1                  MEMIDLEC(II) := MEMUNUSED;
1              END;
1              IF II > 1 THEN MEMIDLECUM := MEMIDLECUM+MEMUNUSED+(TIMEC(II)-TIMEC(II-1));
1          END
1      ELSE BEGIN
1          CPUIDLECOM := 0; MEMIDLECUM := 0;
1          FOR II := 1 TO 100 DO BEGIN
1              CREGION;
1              WRITELN;
1              WRITELN(' CLOCK ', TIMEC(II), ' CCU ', CPUUSERC(II), ' DIOJ ', DIOJOB(II), ' IDLE STORE ', MEMIDLEC(II));
1              WRITELN; WRITE(' RQUEUE ');
1          END;
1          FOR I := 0 TO 19 DO BEGIN CREGION; WRITE(READYQUE(II,I)); CEND; END;
1          CREGION; WRITELN; WRITE(' BQUEUE '); CEND;

```

```

1      FOR I := 0 TO 19 DO BEGIN CREGION; WRITE(BLOCKEDQUE[II,IJ:3]); CEND; END;
1      CREGION; WRITELN; WRITE(' DIOQUE '); CEND;
1      FOR I := 0 TO 19 DO BEGIN CREGION; WRITE(DIOQUE[II,IJ:3]); CEND; END;
1      CREGION; WRITELN; CEND;
1      END;
1      II := 0;
1      END;
1      /C
1      /C ** END OF DUMPING SECTION **
1      /C
1      UNLOCK(CLOCKLOCK);
1      UNLOCK(MASTERLOCK);
1      END;
1      END;
1      IOTERMINATE:
1      BEGIN
1      LOCK(MOSLOCK);
1      SIGNAL(MOSLOCK);
1      WHILE TRUE
1      DO BEGIN
1      WAIT(IOTERM);
1      LOCK(IOTERM);
1      CREGION;
1      WRITELN(' I/O FINISHED');
1      CEND;
1      LOCK(CLOCKLOCK);
1      WITH CONTROLTABLE
1      DO BEGIN
1      LOCK(MASTERLOCK);
1      REGISTER[RTAIL] := CIOJ;
1      RQWAITER[QLISTER[RTAIL]] := CLOCK;
1      RTAIL := (RTAIL + 1) MOD 20;
1      LOCK(ARRANGELOCK);
1      ARRANGENTRY := ARRGLIST ;
1      LOCK(ARRFLAG);
1      FORK WITH TASKNAME(A03) ARRANGE(ARRANGENTRY,RQLIST,RQHEAD,RTAIL,QUANTUM);
1      WAIT(ARRFLAG);
1      AWAIT(A03);
1      UNLOCK(ARRANGELOCK);
1      IF PQHEAD <> PQTAIL THEN BEGIN
1      LOCK(EMPTYRO);
1      SIGNAL(EMPTYRO);
1      CPUIDLENUM := CPUIDLENUM + (CLOCK - CPUIDLE);
1      END;
1      MEMSWAPABLE := MEMSWAPABLE + MEMRECCIOJ;
1      IF DIOQHEAD = DIOQTAIL THEN CIOJ := 0
1      ELSE BEGIN
1      CIOJ := DIOQLIST[DIOQHEAD];
1      DIOQLIST[DIOQHEAD] := 0;
1      DIOQHEAD := (DIOQHEAD + 1) MOD 20;
1      ACTIVATE SLAVETASK(CIOJ);

```

```

1                                     END;
1 /C
1 /C ** DUMPING SECTION **
1 /C
1 II := II+1;
1 IF II <= 100
1 THEN BEGIN
1     TIMEC[II]:= CLOCK; CPUUSER[II]:= CONTROLTABLE.CCU;
1     DIOJOB[II]:= CONTROLTABLE.CIOJ;
1     FOR I:= 0 TO 19 DO BEGIN
1         READYQUE[II,I]:= RQLIST[I];
1         BLOCKEDQUE[II,I]:= BQLIST[I];
1         DIOQUE[II,I]:= DIOQLIST[I];
1         MEMIDLE[II]:= MEMUNUSED;
1     END;
1     IF II>1 THEN MEMIDLECM := MEMIDLECM+MEMUNUSED*(TIMEC[II]-TIMEC[II-1]);
1 END
1 ELSE BEGIN
1     CPUIDLECM := 0; MEMIDLECM:= 0;
1     FOR II:= 1 TO 100 DO BEGIN
1         CREGION;
1         WRITELN;
1         WRITELN(' CLOCK ',TIMEC[II], ' CCU ',CPUUSER[II], ' DIOJ ',DIOJOB[II], ' IDLE STORE ',MEMIDLE[II]);
1         WRITELN; WRITE(' RQUEUE ');
1         CEND;
1         FOR I :=0 TO 19 DO BEGIN CREGION; WRITE(READYQUE[II,I]:3); CEND; END;
1         CREGION; WRITELN; WRITE(' BQUEUE '); CEND;
1         FOR I :=0 TO 19 DO BEGIN CREGION; WRITE(BLOCKEDQUE[II,I]:3); CEND; END;
1         CREGION; WRITELN; WRITE(' DIOQUE '); CEND;
1         FOR I :=0 TO 19 DO BEGIN CREGION; WRITE(DIOQUE[II,I]:3);CEND; END;
1         CREGION; WRITELN; CEND;
1     END;
1     II := 0;
1 END;
1 /C
1 /C ** END OF DUMPING SECTION **
1 /C
1     UNLOCK(CLOCKLOCK);
1     UNLOCK(MASTERLOCK);
1     UNLOCK(TERMLOCK);
1     FND;
1
1 FND;
1 END;
1 MOSINIT:
1 BEGIN
1     CONTROLTABLE.CCU := 0;
1     CONTROLTABLE.CIOJ := 0;
1     LASTINTRPT := 0;
1     LASTSWAP := 0;
1     CLOCK := 0;
1     ROHEAD := 0;

```



```

1      RQTAIL           := 0;
1      DIOQHEAD        := 0;
1      DIOQTAIL        := 0;
1      BQHEAD          := 0;
1      BQTAIL          := 0;
1      MAXRQLEN        := 18;
1      II              := 0;
1      MEMUNUSED       := 4000;
1      MEMSWAPABLE     := 0;
1      TIMEINCREMENT   := 0;
1      TIMEDECREMENT   := 0;
1      QUANTMAX        := 4;
1      CPUIDLECM       := 0;
1      MEMIDLECM       := 0;
1      LOCK(IOWT); LOCK(IOTERM);
1      I := 0;
1      WHILE I <= 19 DO BEGIN
1          RQLISTE I J := 0;
1          RQLISTE I J := 0;
1          RQWAITE I+1 J := 0;
1          DIOQLISTE I J := 0;
1          PRIORITYE I+1 J := 0;
1          STARTTIMEE I+1 J := 0;
1          QUANTUME I+1 J := 0;
1          I := I+1;
1      END;
1      RESET(INFOTABLE);
1      READLN(INFOTABLE);
1      FOR I := 1 TO 18 DO BEGIN
1          READ(INFOTABLE, MEMREQ[I], PRIORITY[I], STARTTIME[I]);
1      END;
1      FOR I := 1 TO 18
1      DO BEGIN
1          FOR J := (I+1) TO 18
1          DO BEGIN
1              IF STARTTIMEE I J > STARTTIMEE J J
1              THEN BEGIN
1                  TEMP := STARTTIMEE J J; STARTTIMEE J J := STARTTIMEE I J; STARTTIMEE I J := TEMP;
1                  TEMP := MEMREQE J J; MEMREQE J J := MEMREQE I J; MEMREQE I J := TEMP;
1                  TEMP := PRIORITYE J J; PRIORITYE J J := PRIORITYE I J; PRIORITYE I J := TEMP;
1              END;
1          END;
1      END;
1      I := 1;
1      WHILE ( I <= MAXRQLEN ) AND
1      ( MEMUNUSED > MEMREQ[I] ) DO BEGIN
1          RQLISTE I := I;
1          RQWAITE I J := STARTTIMEE I J;
1          MEMUNUSED := MEMUNUSED - MEMREQ[I];
1          MEMSWAPABLE := MEMSWAPABLE + MEMREQ[I];
1          RQTAIL := RQTAIL + 1;

```

```

1                                     Y := I+1;
1                                     END;
1
1      K := 18;
1      FOR J:=0 TO K DO BEGIN
1          BOLISTEJ:=I;
1          BOTAIL := BOTAIL +1;
1          I := I+1;
1          END;
1
1      LOCK(FINISH);
1      RELEASE(MOS);
1      LOCK(FINISH);
1      WAIT(FINISH);
1      QUIT;
1      END;
1      END;
1      END;
1
1 *COMPILE  OPTIONS=IRNLV,ELTNAME=TPF%USER1,USER=BIR,SIZE=330,PRIORITY=2,STARTTIME=0000
1      PROGRAM MAIN;
1      VAR A,B,J : INTEGER ; C : REAL ;
1      BEGIN
1          B:= 1;
1          WHILE TRUE
1          DO BEGIN
1              FOR J :=1 TO 999999
1              DO BEGIN
1                  A := B ; C:=5*1023+14/(13.3*990/200) ;
1                  IF (J MOD 333)=0
1                  THEN BEGIN
1                      WRITELN(A,C);
1                      END;
1                  B := A;
1                  END;
1              END;
1          END;
1      END;
1
1 *COMPILE  OPTIONS=IRNLV,ELTNAME=TPF%USER2,USER=IKI,SIZE=420,PRIORITY=1,STARTTIME=0000
1      PROGRAM MAIN;
1      VAR A,B,J : INTEGER ; C : REAL ;
1      BEGIN
1          B:= 2;
1          WHILE TRUE
1          DO BEGIN
1              FOR J := 1 TO 999999
1              DO BEGIN
1                  A := B ; C := 403*70024/(12,3*1408*102);
1                  B := A;
1                  END;
1              END;
1          END;
1      END;
1
1 *COMPILE  OPTIONS=IRNLV,ELTNAME=TPF%USER3,USER=UC,SIZE=140,PRIORITY=5,STARTTIME=0000

```

```

1 PROGRAM MAIN;
1 VAR A,B,J : INTEGER ; C : REAL ;
1 BEGIN
1   B:= 3;
1   WHILE TRUE
1     DO BEGIN
1       FOR J := 1 TO 999999
1         DO BEGIN
1           A := B ; C := 5.23*18.2*0.03/(3.8*8.8*0.44);
1           IF (J MOD 444)=0
1             THEN BEGIN
1               WRITELN(A,C);
1               END;
1           B := A;
1         END;
1       END;
1     END;
1   END;
1 *COMPILE OPTIONS=IRNLHV,ELTNAME=TPF%USER4,USER=DRT,SIZE=435,PRIORITY=3,STARTTIME=0000
1 PROGRAM MAIN;
1 VAR A,B,J : INTEGER ; C : REAL ;
1 BEGIN
1   B:= 4;
1   WHILE TRUE
1     DO BEGIN
1       FOR J := 1 TO 999999
1         DO BEGIN
1           A := B ; C := 6.4*8.1*0.005/(3.3*8.3*0.7);
1           B := A;
1         END;
1       END;
1     END;
1   END;
1 *COMPILE OPTIONS=IRNLHV,ELTNAME=TPF%USER5,USER=BES,SIZE=200,PRIORITY=1,STARTTIME=0000
1 PROGRAM MAIN;
1 VAR A,B,J : INTEGER ; C : REAL ;
1 BEGIN
1   B:= 5;
1   WHILE TRUE
1     DO BEGIN
1       FOR J := 1 TO 999999
1         DO BEGIN
1           A := B ; C := 0.44*1.5*9.99*9.2/(3.7*0.99);
1           IF (J MOD 555)=0
1             THEN BEGIN
1               WRITELN(A,C);
1               END;
1           B := A;
1         END;
1       END;
1     END;
1   END;
1 END;

```

```

1      END.
1 *COMPILE  OPTIONS=IPNIMV,ELTNAME=TPF%USER6,USER=ALT,SIZE=290,PRIORITY=3,STARTTIME=0000
1      PROGRAM MAIN;
1      VAR A,B,J : INTEGER ; C : REAL ;
1      BEGIN
1          B := 6;
1          WHILE TRUE
1              DO BEGIN
1                  FOR J := 1 TO 999999
1                      DO BEGIN
1                          A := B ; C := 0.91*3.5*9.4/(15.3+8.2*0.45);
1                          P := A;
1                      END;
1                  END;
1      END.
1
1      END.
1 *COMPILE  OPTIONS=IRNLHV,ELTNAME=TPF%USER7,USER=YED,SIZE=400,PRIORITY=1,STARTTIME=0000
1      PROGRAM MAIN;
1      VAR A,B,J : INTEGER ; C : REAL ;
1      BEGIN
1          B := 7;
1          WHILE TRUE
1              DO BEGIN
1                  FOR J := 1 TO 999999
1                      DO BEGIN
1                          A := B ; C := 7.2*7.1*0.03/(15.3+9.9*0.07);
1                          IF (J MOD 400)=0
1                              THEN BEGIN
1                                  WRITELN(A,C);
1                              END;
1                          P := A;
1                      END;
1                  END;
1      END;
1
1      END.
1 *COMPILE  OPTIONS=IPNLMV,ELTNAME=TPF%USER8,USER=SEK,SIZE=350,PRIORITY=4,STARTTIME=0000
1      PROGRAM MAIN;
1      VAR A,B,J : INTEGER ; C : REAL ;
1      BEGIN
1          B := 8;
1          WHILE TRUE
1              DO BEGIN
1                  FOR J := 1 TO 999999
1                      DO BEGIN
1                          A := B ; C := 7.1*0.92*0.65/(8.1+9.3*0.56);
1                          P := A;
1                      END;
1                  END;
1      END;
1
1      END.
1 *COMPILE  OPTIONS=IPNLMV,ELTNAME=TPF%USER9,USER=DOK,SIZE=360,PRIORITY=3,STARTTIME=0000

```

```

1 PROGRAM MAIN;
1 VAR A,B,J : INTEGER ; C : REAL ;
1 BEGIN
1   B:= 9;
1   WHILE TRUE
1     DO BEGIN
1       FOR J :=1 TO 999999
1         DO BEGIN
1           A := B ; C:= 9.9+0.2*0.7/(0.67+0.89);
1           B := A;
1         END;
1       END;
1     END;
1   END;
1 *COMPILE OPTIONS=IRNLMV,ELTNAME=TPF%USER10,USER=04,SIZE=220,PRIORITY=3,STARTTIME=0000
1 PROGRAM MAIN;
1 VAR A,B,J : INTEGER ; C : REAL ;
1 BEGIN
1   B:= 10;
1   WHILE TRUE
1     DO BEGIN
1       FOR J :=1 TO 99999
1         DO BEGIN
1           A := B ; C := 1,1+2,2+3,3/(6.4+0.7+8.1);
1           B := A;
1         END;
1       END;
1     END;
1   END;
1 *COMPILE OPTIONS=IRNLMV,ELTNAME=TPF%USER11,USER=08B,SIZE=300,PRIORITY=2,STARTTIME=0000
1 PROGRAM MAIN;
1 VAR A,B,J : INTEGER ; C : REAL;
1 BEGIN
1   B:= 11;
1   WHILE TRUF
1     DO BEGIN
1       FOR J :=1 TO 999999
1         DO BEGIN
1           A := B ; C := 8,1+8,8+0,5/(7.4+0.1+0.6);
1           B := A;
1         END;
1       END;
1     END;
1   END;
1 *COMPILE OPTIONS=IRNLMV,ELTNAME=TPF%USER12,USER=041,SIZE=270,PRIORITY=1,STARTTIME=0000
1 PROGRAM MAIN;
1 VAR A,B,J : INTEGER ; C : REAL ;
1 BEGIN
1   B:= 12;
1   WHILE TRUE
1     DO BEGIN

```

```

1          FOR J :=1 TO 999999
1          DO BEGIN
1              A := B ; C := 7.3*9.3*0.04/(4.3*9.1*0.03);
1              B := A;
1          END;
1
1      END;
1
1  *COMPILE  OPTIONS=IRNLHV,ELTNAME=TPF%USER13,USER=ONU,SIZE=420,PRIORITY=2,STARTTIME=0000
1  PROGRAM MAIN;
1  VAR A,B,J : INTEGER ; C : REAL ;
1  BEGIN
1      B:= 13;
1      WHILE TRUE
1      DO BEGIN
1          FOR J :=1 TO 999999
1          DO BEGIN
1              A := B ; C := 4.1*0.044*9.1/(4.2*0.8*9.3);
1              B := A;
1          END;
1      END;
1
1  END;
1
1  *COMPILE  OPTIONS=IRNLHV,ELTNAME=TPF%USER14,USER=OND,SIZE=300,PRIORITY=3,STARTTIME=0000
1  PROGRAM MAIN;
1  VAR A,B,J : INTEGER ; C : REAL ;
1  BEGIN
1      B:= 14;
1      WHILE TRUE
1      DO BEGIN
1          FOR J := 1 TO 999999
1          DO BEGIN
1              A := B ; C := 8.2*9.1*0.055/(4.3*0.94*0.56);
1              IF (J MOD 500)=0
1              THEN BEGIN
1                  WRITELN(A,C);
1              END;
1              B := A;
1          END;
1      END;
1
1  END;
1
1  *COMPILE  OPTIONS=IRNLHV,ELTNAME=TPF%USER15,USER=OSS,SIZE=490,PRIORITY=2,STARTTIME=0000
1  PROGRAM MAIN;
1  VAR A,B,J : INTEGER ; C : REAL;
1  BEGIN
1      B:= 15;
1      WHILE TRUE
1      DO BEGIN
1          FOR J := 1 TO 999999
1          DO BEGIN

```

```

1           A := B ; C := 0.45*0.91*8.34/(5.3*0.99);
1           B := A;
1           END;
1       END;
1
1   END.
1 *COMPILE  OPTIONS=IRNLMV,FLTNAME=TPFS,USER16,USER=ONA,SIZE=360,PRIORITY=2,STARTTIME=0000
1 PROGRAM MAIN;
1 VAR A,B,J : INTEGER ; C : REAL ;
1 BEGIN
1     B := 16;
1     WHILE TRUE
1     DO BEGIN
1         FOR J := 1 TO 999999
1         DO BEGIN
1             A := B ; C := 9.2*0.06*0.45/(0.23*9.6);
1             B := A;
1         END;
1     END;
1
1   END.
1
1 *COMPILE  OPTIONS=IRNLMV,FLTNAME=TPFS,USER17,USER=ONY,SIZE=210,PRIORITY=5,STARTTIME=0000
1 PROGRAM MAIN;
1 VAR A,B,J : INTEGER ; C : REAL ;
1 BEGIN
1     B := 17;
1     WHILE TRUE
1     DO BEGIN
1         FOR J := 1 TO 999999
1         DO BEGIN
1             A := B ; C := 0.05*8.4*0.76/(0.3*7.8*0.4);
1             IF (J MOD 999)=0
1             THEN BEGIN
1                 WRITELN(A,C);
1             END;
1             B := A;
1         END;
1     END;
1
1   END.
1
1   END.
1 *COMPILE  OPTIONS=IRNLMV,FLTNAME=TPFS,USER18,USER=ONS,SIZE=145,PRIORITY=1,STARTTIME=0000
1 PROGRAM MAIN;
1 VAR A,B,J : INTEGER ; C : REAL ;
1 BEGIN
1     B := 18;
1     WHILE TRUE
1     DO BEGIN
1         FOR J := 1 TO 999999
1         DO BEGIN
1             A := B ; C := 9.5*9.3*0.56/(5.5*0.3*0.7);
1             B := A;

```

```

1          END;
1          END;
1          END.
@CPS,S,N  SCRATCH$,SCRATCH1,CPS,S/1,SCRATCH$,SCRATCH2
MACRO INTERPPETER LEVEL 7R1 07/31/83 19:51:27
@CPS,S,N  SCRATCH$,SCRATCH2,CPS,S/2,
MACRO INTERPRETER LEVEL 7R1 07/31/83 19:53:07
@CPS,X
@BU*PASCAL,S,IRSLMG
PASCAL 8R1A PLTB SUNDAY, 1983 JULY 31, 19:57:27
1

```

```

2  PROGRAM MAIN(INPUT,OUTPUT);
3  TYPE  APRTYP1 = ARRAY [ 0 .. 19 ] OF INTEGER ;
4        ARRTYP2 = ARRAY [ 1 .. 20 ] OF INTEGER ;
5        MOSENTRIES =(MOSINIT,CLOCKSTART,REGULARINT,IOINITIATE,IOTERMINATE);
6        ARRGENTRIES =( ARRGNIT , ARRGLIST) ;
7  PROCEDURE MOS(VAR MOSENTRY ; MOSENTRIES);EXTERNAL ;
8  PROCEDURE ARRANGE(VAR ARRANGENTRY ; ARRGENTRIES ; VAR LIST : ARRTYP1;
9                   LISHEAD : INTEGER;LISTAIL : INTEGER ;VAR SORTKEY : ARRTYP2);E
10     VAR GLOBALS : ARRAY (,1 .. 1880,) OF INTEGER ;
11     ARRANGENTRY : ARRGENTRIES ;
12     DUMAPR1 : ARRTYP1 ;
13     DUMARR2 : ARRTYP2 ;
14     DUMINT : INTEGER ;
15     I : INTEGER ;
16     MOSENTRY : MOSENTRIES ;
17 %CODE R$CELL*      T$CELL
18 %CODE C$GNST*      T$CELL
19 %CODE S$YSTEMHOLD* T$CELL
20 %CODE A$CTIVATOR*  T$CELL
21 %CODE A$RRANGELOCK* T$CELL
22 %CODE A$RRFLAG*    T$CELL
23 %CODE C$LOCKINTRPT* T$CELL
24 %CODE C$LOCKLOCK*  T$CELL
25 %CODE E$MPTYRO*    T$CELL
26 %CODE F$INISH*     T$CELL
27 %CODE I$OINIT*     T$CELL
28 %CODE I$OTERM*     T$CELL
29 %CODE M$ASTERLOCK* T$CELL
30 %CODE M$OSLOCK*    T$CELL
31 %CODE S$ARTER*     T$CELL
32 %CODE T$ERMLOCK*   T$CELL
33 %CODE T$ARRANGE*   T$CELL
34 %CODE T$MATN*      T$CELL
35 %CODE T$MOS*       T$CELL
36 %CODE X$S$ARRANGE* RES 1
37 %CODE X$S$MAIN*    RES 1
38 %CODE X$S$MOS*     RES 1

```

```

1          06 135
1
1
1
1
2          0 5
2          0 4
2          5 9
1          136 2015
1          2016 2016
1          2017 2036
1          2017 2056
1          2057 2057
1          2058 2058
1          2059 2059
1
1          0 0
1          1 1
1          2 2
1          3 3
1          4 4
1          5 5
1          6 6
1          7 7
1          8 8
1          9 9
1          10 10
1          11 11
1          12 12
1          13 13
1          14 14
1          15 15
1          16 16
1          17 17
1          18 18
1          19 19
1          20 20
1          21 21

```



	*(3)	012704 013106					
USER18	*(1)	013107 013177	*(0)	040425 040427	31	JUL 83	20:10:12
	*(3)	013200 013214					
USER17	*(1)	013215 013344	*(0)	040430 040431	31	JUL 83	20:09:59
	*(3)	013345 013361					
USER16	*(1)	013362 013451	*(0)	040432 040433	31	JUL 83	20:09:43
	*(3)	013452 013464					
USER15	*(1)	013465 013554	*(0)	040434 040435	31	JUL 83	20:09:30
	*(3)	013555 013567					
USER14	*(1)	013570 013717	*(0)	040436 040437	31	JUL 83	20:09:18
	*(3)	013720 013734					
USER13	*(1)	013735 014025	*(0)	040440 040441	31	JUL 83	20:09:04
	*(3)	014026 014042					
USER12	*(1)	014043 014133	*(0)	040442 040443	31	JUL 83	20:08:49
	*(3)	014134 014150					
USER11	*(1)	014151 014241	*(0)	040444 040445	31	JUL 83	20:08:37
	*(3)	014242 014256					
USER10	*(1)	014257 014347	*(0)	040446 040447	31	JUL 83	20:08:25
	*(3)	014350 014364					
USER9	*(1)	014365 014454	*(0)	040450 040451	31	JUL 83	20:08:13
	*(3)	014455 014467					
USER8	*(1)	014470 014560	*(0)	040452 040453	31	JUL 83	20:08:01
	*(3)	014561 014575					
USER7	*(1)	014576 014725	*(0)	040454 040455	31	JUL 83	20:07:49
	*(3)	014726 014742					
USER6	*(1)	014743 015033	*(0)	040456 040457	31	JUL 83	20:07:36
	*(3)	015034 015050					
USER5	*(1)	015051 015200	*(0)	040450 040461	31	JUL 83	20:07:22
	*(3)	015201 015215					
USER4	*(1)	015216 015306	*(0)	040452 040463	31	JUL 83	20:07:09
	*(3)	015307 015323					
USER3	*(1)	015324 015453	*(0)	040454 040465	31	JUL 83	20:06:57
	*(3)	015454 015470					
USER2	*(1)	015471 015561	*(0)	040465 040467	31	JUL 83	20:06:42
	*(3)	015562 015576					
USER1	*(1)	015577 015732	*(0)	040470 040471	31	JUL 83	20:06:31
	*(3)	015733 015745					
MODEL05	*(1)	015746 024251	*(0)	040472 040503	31	JUL 83	20:06:18
	*(3)	024252 024351					
ARRANG	*(1)	024352 024567	*(0)	040504 040505	31	JUL 83	20:02:09
MAIN	*(1)	024570 025516	*(0)	040506 040536	31	JUL 83	20:01:54
			*(036)	040537 140541			

SYSSQLIR% LEVEL

END MAP, ERRORS: 0 TIME: 22,421 STORAGE: 18816/5/040777/075777

0USE INFOTABLE,INFO%

0USE PA%000000,INFO%

0SETC,0

0XQT TPF%,MAIN

I/O STARTED  
 REGULAR INTRPT  
 I/O STARTED

		SWAPPING
		REGULAR INTRPT
		REGULAR INTRPT
		I/O STARTED
		SWAPPING
		REGULAR INTRPT
		REGULAR INTRPT
		REGULAR INTRPT
		SWAPPING
		REGULAR INTRPT
		REGULAR INTRPT
		I/O STARTED
		SWAPPING
1	0.527089	REGULAR INTRPT
		REGULAR INTRPT
		I/O FINISHED
		SWAPPING
		REGULAR INTRPT
		I/O STARTED
3	0.194078	REGULAR INTRPT
		I/O FINISHED
		I/O STARTED
7	1.461407	I/O FINISHED
		SWAPPING
		I/O STARTED
		I/O STARTED
14	1.813149	I/O FINISHED
		I/O STARTED
17	0.341026	I/O FINISHED
		I/O STARTED
1	0.527089	I/O FINISHED
		REGULAR INTRPT
3	0.194078	I/O FINISHED
		SWAPPING
		REGULAR INTRPT
		REGULAR INTRPT
5	16.560000	I/O FINISHED
		REGULAR INTRPT
7	1.461407	I/O FINISHED
		REGULAR INTRPT
14	1.813149	I/O FINISHED
		REGULAR INTRPT
		SWAPPING
		REGULAR INTRPT
		REGULAR INTRPT

CLOCK SR CCU 1 DIOJ 0 IDLE STORF 295

ROJEUF	0	2	3	4	5	6	7	8	9	10	11	12	0	0	0	0	0	0	0	0
BOJEUE	13	14	15	16	17	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DIJQUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 100 CCU 1 DIOJ 0 IDLE STOPL 295

RQJEUE	0	2	3	4	5	6	7	8	9	10	11	12	0	0	0	0	0	0	0
BQJEUE	13	14	15	16	17	18	0	0	0	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 100 CCU 2 DIOJ 1 IDLE STOPL 295

RQJEUE	0	0	3	4	5	6	7	8	9	10	11	12	0	0	0	0	0	0	0
BQJEUE	13	14	15	16	17	18	0	0	0	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 150 CCU 2 DIOJ 1 IDLE STORE 295

RQJEUE	0	0	3	4	5	6	7	8	9	10	11	12	0	0	0	0	0	0	0
BQJEUE	13	14	15	16	17	18	0	0	0	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 200 CCU 2 DIOJ 1 IDLE STORE 295

RQJEUE	0	0	3	4	5	6	7	8	9	10	11	12	0	0	0	0	0	0	0
BQJEUE	13	14	15	16	17	18	0	0	0	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 250 CCU 2 DIOJ 1 IDLE STORE 295

RQJEUE	0	0	3	4	5	6	7	8	9	10	11	12	0	0	0	0	0	0	0
BQJEUE	13	14	15	16	17	18	0	0	0	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 300 CCU 3 DIOJ 1 IDLE STORE 295

RQJEUE	0	0	0	4	5	6	7	8	9	10	11	12	2	0	0	0	0	0	0
BQJEUE	13	14	15	16	17	18	0	0	0	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 350 CCU 3 DIOJ 1 IDLE STORE 295

RQJEUE	0	0	0	4	5	6	7	8	9	10	11	12	2	0	0	0	0	0	0
BQJEUE	13	14	15	16	17	18	0	0	0	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 350 CCU 4 DIOJ 1 IDLE STORE 295

RQJEUE	0	0	0	0	5	6	7	8	9	10	11	12	2	0	0	0	0	0	0
BQJEUE	13	14	15	16	17	18	0	0	0	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 400 CCU 4 DIOJ 1 IDLE STORE 295

ROJEUF U 0 0 0 5 6 7 A 9 10 11 12 2 0 0 0 0 0 0 0  
 BOJEUE 13 14 15 16 17 18 0 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 450 CCU 4 DIOJ 1 IDLE STORE 295

ROJEUE 0 0 0 0 5 6 7 8 9 10 11 12 2 0 0 0 0 0 0  
 BOJEUE 13 14 15 16 17 18 0 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 500 CCU 4 DIOJ 1 IDLE STORE 75

ROJEUE U 0 0 0 6 7 8 9 10 11 12 13 2 0 0 0 0 0 0  
 BOJEUE 0 14 15 16 17 18 5 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 550 CCU 6 DIOJ 1 IDLE STORE 75

ROJEUE U 0 0 0 0 7 8 9 10 11 12 13 2 4 0 0 0 0 0  
 BOJEUE 0 14 15 16 17 18 5 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 600 CCU 6 DIOJ 1 IDLE STORE 75

ROJEUE 0 0 0 0 0 7 8 9 10 11 12 13 2 4 0 0 0 0  
 BOJEUE 0 14 15 16 17 18 5 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 650 CCU 6 DIOJ 1 IDLE STORE 75

ROJEUE 0 0 0 0 0 7 8 9 10 11 12 13 2 4 0 0 0 0  
 BOJEUF 0 14 15 16 17 18 5 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 700 CCU 6 DIOJ 1 IDLE STORE 75

ROJEUF U 0 0 0 0 7 8 9 10 11 12 13 2 4 0 0 0 0  
 BOJEUF 0 14 15 16 17 18 5 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 750 CCU 7 DIOJ 1 IDLE STORE 75

ROJEUF U 0 0 0 0 0 7 8 9 10 11 12 13 2 4 6 0 0 0  
 BOJEUE 0 14 15 16 17 18 5 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 800 CCU 7 DIOJ 1 IDLE STORE 75

ROJEUE 0 0 0 0 0 0 8 9 10 11 12 13 2 4 6 0 0 0  
 BOJEUE 0 14 15 16 17 18 5 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 800 CCU 8 DIOJ 1 IDLE STORE 75

RQJEU	0	0	0	0	0	0	0	9	10	11	12	13	2	4	6	0	0	0	0	0
BOJEU	0	14	15	16	17	18	5	0	0	0	0	0	0	0	0	0	0	0	0	0
DIOJUE	0	3	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 850 CCU 8 DIOJ 1 IDLE STORE 75

RQJEU	0	0	0	0	0	0	0	9	10	11	12	13	2	4	6	0	0	0	0	0
BOJEU	0	14	15	16	17	18	5	0	0	0	0	0	0	0	0	0	0	0	0	0
DIOJUE	0	3	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 900 CCU 8 DIOJ 1 IDLE STORE 75

RQJEU	0	0	0	0	0	0	0	9	10	11	12	13	2	4	6	0	0	0	0	0
BOJEU	0	14	15	16	17	18	5	0	0	0	0	0	0	0	0	0	0	0	0	0
DIOJUE	0	3	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 950 CCU 8 DIOJ 1 IDLE STORE 75

RQJEU	0	0	0	0	0	0	0	9	10	11	12	13	2	4	6	0	0	0	0	0
BOJEU	0	14	15	16	17	18	5	0	0	0	0	0	0	0	0	0	0	0	0	0
DIOJUE	0	3	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 1000 CCU 10 DIOJ 1 IDLE STORE 135

RQJEU	0	0	0	0	0	0	0	0	11	12	13	14	2	4	6	8	0	0	0	0
BOJEU	0	0	15	16	17	18	5	9	0	0	0	0	0	0	0	0	0	0	0	0
DIOJUE	0	3	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 1050 CCU 10 DIOJ 1 IDLE STORE 135

RQJEU	0	0	0	0	0	0	0	0	11	12	13	14	2	4	6	8	0	0	0	0
BOJEU	0	0	15	16	17	18	5	9	0	0	0	0	0	0	0	0	0	0	0	0
DIOJUE	0	3	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 1100 CCU 10 DIOJ 1 IDLE STORE 135

RQJEU	0	0	0	0	0	0	0	0	11	12	13	14	2	4	6	8	0	0	0	0
BOJEU	0	0	15	16	17	18	5	9	0	0	0	0	0	0	0	0	0	0	0	0
DIOJUE	0	3	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 1150 CCU 10 DIOJ 1 IDLE STORE 135

RQJEU	0	0	0	0	0	0	0	0	11	12	13	14	2	4	6	8	0	0	0	0
BOJEU	0	0	15	16	17	18	5	9	0	0	0	0	0	0	0	0	0	0	0	0
DIOJUE	0	3	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 1200 CCU 11 DIOJ 1 IDLE STORE 135

ROJUE 0 0 0 0 0 0 0 0 12 13 14 2 4 6 8 10 0 0 0  
 BOJUE 0 0 15 16 17 18 5 9 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 1300 CCU 11 DIOJ 1 IDLE STORE 135

ROJUE 0 0 0 0 0 0 0 12 13 14 2 4 6 8 10 0 0 0  
 BOJUE 0 0 15 16 17 18 5 9 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 1350 CCU 11 DIOJ 1 IDLE STORE 135

ROJUE 0 0 0 0 0 0 0 12 13 14 2 4 6 8 10 0 0 0  
 BOJUE 0 0 15 16 17 18 5 9 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 1400 CCU 11 DIOJ 1 IDLE STORE 135

ROJUE 0 0 0 0 0 0 0 12 13 14 2 4 6 8 10 0 0 0  
 BOJUE 0 0 15 16 17 18 5 9 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 1450 CCU 12 DIOJ 1 IDLE STORE 135

ROJUE 0 0 0 0 0 0 0 13 14 2 4 6 8 10 11 0 0 0  
 BOJUE 0 0 15 16 17 18 5 9 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 1500 CCU 12 DIOJ 1 IDLE STORE 65

ROJUE 0 0 0 0 0 0 0 13 14 15 4 6 8 10 11 0 0 0  
 BOJUE 0 0 16 17 18 5 9 2 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 1550 CCU 12 DIOJ 1 IDLE STORE 65

ROJUE 0 0 0 0 0 0 0 13 14 15 4 6 8 10 11 0 0 0  
 BOJUE 0 0 16 17 18 5 9 2 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 1600 CCU 12 DIOJ 1 IDLE STORE 65

ROJUE 0 0 0 0 0 0 0 13 14 15 4 6 8 10 11 0 0 0  
 BOJUE 0 0 16 17 18 5 9 2 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 1650 CCU 13 DIOJ 1 IDLE STORE 65

ROJUE 0 0 0 0 0 0 0 14 15 4 6 8 10 11 12 0 0 0  
 BOJUE 0 0 16 17 18 5 9 2 0 0 0 0 0 0 0 0 0 0 0  
 DIOQUE 0 3 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

CLOCK 1700 CCU 13 DIOJ 1 IDLE STORE 65

ROJEU	0	0	0	0	0	0	0	0	0	0	0	14	15	4	6	8	10	11	12	0
BOJEU	0	0	0	16	17	18	5	9	2	0	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	3	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 1750 CCU 13 DIOJ 1 IDLE STORE 65

ROJEU	0	0	0	0	0	0	0	0	0	0	0	14	15	4	6	8	10	11	12	0
BOJEU	0	0	0	16	17	18	5	9	2	0	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	3	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 1800 CCU 13 DIOJ 1 IDLE STORE 65

ROJEU	0	0	0	0	0	0	0	0	0	0	0	14	15	4	6	8	10	11	12	0
BOJEU	0	0	0	16	17	18	5	9	2	0	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	3	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 1850 CCU 14 DIOJ 1 IDLE STORE 65

ROJEU	0	0	0	0	0	0	0	0	0	0	0	15	4	6	8	10	11	12	13	0
BOJEU	0	0	0	16	17	18	5	9	2	0	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	3	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 1900 CCU 14 DIOJ 1 IDLE STORE 65

ROJEU	0	0	0	0	0	0	0	0	0	0	0	15	4	6	8	10	11	12	13	0
BOJEU	0	0	0	16	17	18	5	9	2	0	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	3	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 1900 CCU 15 DIOJ 1 IDLE STORE 65

ROJEU	0	0	0	0	0	0	0	0	0	0	0	0	4	6	8	10	11	12	13	0
BOJEU	0	0	0	16	17	18	5	9	2	0	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	3	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 1950 CCU 15 DIOJ 1 IDLE STORE 65

ROJEU	0	0	0	0	0	0	0	0	0	0	0	0	4	6	8	10	11	12	13	0
BOJEU	0	0	0	16	17	18	5	9	2	0	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	3	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2000 CCU 15 DIOJ 1 IDLE STORE 140

ROJEU	0	0	0	0	0	0	0	0	0	0	0	0	16	6	8	10	11	12	13	0
BOJEU	0	0	0	0	17	18	5	9	2	4	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	3	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2050 CCU 15 DIOJ 1 IDLE STORE 140

RQJEU	0	0	0	0	0	0	0	0	0	0	0	0	0	16	6	8	10	11	12	13
BQJEU	0	0	0	0	17	18	5	9	2	4	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	3	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2100 CCU 16 DIOJ 1 IDLE STORE 140

RQJEU	15	0	0	0	0	0	0	0	0	0	0	0	0	6	8	10	11	12	13	
BQJEU	0	0	0	0	17	18	5	9	2	4	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	3	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2150 CCU 16 DIOJ 1 IDLE STORE 140

RQJEU	15	0	0	0	0	0	0	0	0	0	0	0	0	6	8	10	11	12	13	
BQJEU	0	0	0	0	17	18	5	9	2	4	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	3	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2200 CCU 16 DIOJ 1 IDLE STORE 140

RQJEU	15	0	0	0	0	0	0	0	0	0	0	0	0	6	8	10	11	12	13	
BQJEU	0	0	0	0	17	18	5	9	2	4	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	3	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2250 CCU 16 DIOJ 1 IDLE STORE 140

RQJEU	15	0	0	0	0	0	0	0	0	0	0	0	0	6	8	10	11	12	13	
BQJEU	0	0	0	0	17	18	5	9	2	4	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	3	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2300 CCU 6 DIOJ 1 IDLE STORE 140

RQJEU	15	16	0	0	0	0	0	0	0	0	0	0	0	0	8	10	11	12	13	
BQJEU	0	0	0	0	17	18	5	9	2	4	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	3	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2300 CCU 6 DIOJ 3 IDLE STORE 140

RQJEU	13	15	16	0	0	0	0	0	0	0	0	0	0	0	1	8	10	11	12	
BQJEU	0	0	0	0	17	18	5	9	2	4	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	0	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2350 CCU 6 DIOJ 3 IDLE STORE 140

RQJEU	13	15	16	0	0	0	0	0	0	0	0	0	0	0	1	8	10	11	12	
BQJEU	0	0	0	0	17	18	5	9	2	4	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	0	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2400 CCU 6 DIOJ 3 IDLE STORE 140

RQJEU	13	15	16	0	0	0	0	0	0	0	0	0	0	0	1	8	10	11	12	
BQJEU	0	0	0	0	17	18	5	9	2	4	0	0	0	0	0	0	0	0	0	0
DIOQUE	0	0	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



CLOCK 2450 CCU 6 DIOJ 3 IDLE STORE 140

RQUEUE	13	15	16	0	0	0	0	0	0	0	0	0	0	0	1	8	10	11	12
BQUEUE	0	0	0	0	17	18	5	9	2	4	0	0	0	0	0	0	0	0	0
DIOQUE	0	0	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2500 CCU 17 DIOJ 3 IDLE STORE 135

RQUEUE	12	13	15	16	6	0	0	0	0	0	0	0	0	0	0	18	1	10	11
BQUEUE	0	0	0	0	0	0	5	9	2	4	8	0	0	0	0	0	0	0	0
DIOQUE	0	0	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2600 CCU 17 DIOJ 3 IDLE STORE 135

RQUEUE	12	13	15	16	6	0	0	0	0	0	0	0	0	0	0	18	1	10	11
BQUEUE	0	0	0	0	0	0	5	9	2	4	8	0	0	0	0	0	0	0	0
DIOQUE	0	0	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2650 CCU 17 DIOJ 3 IDLE STORE 135

RQUEUE	12	13	15	16	6	0	0	0	0	0	0	0	0	0	0	18	1	10	11
BQUEUE	0	0	0	0	0	0	5	9	2	4	8	0	0	0	0	0	0	0	0
DIOQUE	0	0	7	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2650 CCU 18 DIOJ 3 IDLE STORE 135

RQUEUE	12	13	15	16	6	0	0	0	0	0	0	0	0	0	0	0	1	10	11
BQUEUE	0	0	0	0	0	0	5	9	2	4	8	0	0	0	0	0	0	0	0
DIOQUE	0	0	7	14	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2700 CCU 18 DIOJ 3 IDLE STORE 135

RQUEUE	12	13	15	16	6	0	0	0	0	0	0	0	0	0	0	0	1	10	11
BQUEUE	0	0	0	0	0	0	5	9	2	4	8	0	0	0	0	0	0	0	0
DIOQUE	0	0	7	14	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2750 CCU 18 DIOJ 3 IDLE STORE 135

RQUEUE	12	13	15	16	6	0	0	0	0	0	0	0	0	0	0	0	1	10	11
BQUEUE	0	0	0	0	0	0	5	9	2	4	8	0	0	0	0	0	0	0	0
DIOQUE	0	0	7	14	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2800 CCU 18 DIOJ 3 IDLE STORE 135

RQUEUE	12	13	15	16	6	0	0	0	0	0	0	0	0	0	0	0	1	10	11
BQUEUE	0	0	0	0	0	0	5	9	2	4	8	0	0	0	0	0	0	0	0
DIOQUE	0	0	7	14	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2850 CCU 1 DIOJ 3 IDLE STORE 135

ROJEU	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	0	10	11
BOJEU	0	0	0	0	0	0	5	9	2	4	8	0	0	0	0	0	0	0
DIOQUE	0	0	7	14	17	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2850 CCU 1 DIOJ 7 IDLE STORE 135

ROJEU	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	3	10
BOJEU	0	0	0	0	0	0	5	9	2	4	8	0	0	0	0	0	0	0
DIOQUE	0	0	0	14	17	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2900 CCU 1 DIOJ 7 IDLE STORE 135

ROJEU	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	3	10
BOJEU	0	0	0	0	0	0	5	9	2	4	8	0	0	0	0	0	0	0
DIOQUE	0	0	0	14	17	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2950 CCU 1 DIOJ 7 IDLE STORE 135

ROJEU	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	3	10
BOJEU	0	0	0	0	0	0	5	9	2	4	8	0	0	0	0	0	0	0
DIOQUE	0	0	0	14	17	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2950 CCU 3 DIOJ 7 IDLE STORE 135

ROJEU	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	0	10
BOJEU	0	0	0	0	0	0	5	9	2	4	8	0	0	0	0	0	0	0
DIOQUE	0	0	0	14	17	1	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 2950 CCU 3 DIOJ 14 IDLE STORE 135

ROJEU	10	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	7
BOJEU	0	0	0	0	0	0	5	9	2	4	8	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	17	1	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 3000 CCU 3 DIOJ 14 IDLE STORE 155

ROJEU	7	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	5
BOJEU	0	0	0	0	0	0	0	9	2	4	8	10	0	0	0	0	0	0
DIOQUE	0	0	0	0	17	1	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 3050 CCU 3 DIOJ 14 IDLE STORE 155

ROJEU	7	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	5
BOJEU	0	0	0	0	0	0	0	9	2	4	8	10	0	0	0	0	0	0
DIOQUE	0	0	0	0	17	1	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 3050 CCU 5 DIOJ 14 IDLE STORE 155

ROJEU	7	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	0
BOJEU	0	0	0	0	0	0	0	9	2	4	8	10	0	0	0	0	0	0
DIOQUE	0	0	0	0	17	1	3	0	0	0	0	0	0	0	0	0	0	0

CLOCK 3100 CCU 5 DIOJ 14 IDLE STORE 155

ROJEUE	7	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	0	0
BOJEUE	0	0	0	0	0	0	0	9	2	4	8	10	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	17	1	3	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 3150 CCU 5 DIOJ 14 IDLE STORE 155

ROJEUE	7	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	0	0
BOJEUE	0	0	0	0	0	0	0	9	2	4	8	10	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	17	1	3	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 3150 CCU 7 DIOJ 14 IDLE STORE 155

ROJEUE	0	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	0	0
BOJEUE	0	0	0	0	0	0	0	9	2	4	8	10	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	17	1	3	5	0	0	0	0	0	0	0	0	0	0	0

CLOCK 3150 CCU 7 DIOJ 17 IDLE STORE 155

ROJEUE	0	14	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	0
BOJEUE	0	0	0	0	0	0	0	9	2	4	8	10	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	0	1	3	5	0	0	0	0	0	0	0	0	0	0	0

CLOCK 3200 CCU 7 DIOJ 17 IDLE STORE 155

ROJEUE	0	14	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	0
BOJEUE	0	0	0	0	0	0	0	9	2	4	8	10	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	0	1	3	5	0	0	0	0	0	0	0	0	0	0	0

CLOCK 3250 CCU 7 DIOJ 17 IDLE STORE 155

ROJEUE	0	14	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	0
BOJEUE	0	0	0	0	0	0	0	9	2	4	8	10	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	0	1	3	5	0	0	0	0	0	0	0	0	0	0	0

CLOCK 3250 CCU 14 DIOJ 17 IDLE STORE 155

ROJEUE	0	7	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0	0
BOJEUE	0	0	0	0	0	0	0	9	2	4	8	10	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	0	1	3	5	7	0	0	0	0	0	0	0	0	0	0

CLOCK 3250 CCU 14 DIOJ 1 IDLE STORE 155

ROJEUE	0	0	17	11	12	13	15	16	18	6	0	0	0	0	0	0	0	0	0
BOJEUE	0	0	0	0	0	0	0	9	2	4	8	10	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	0	0	3	5	7	0	0	0	0	0	0	0	0	0	0

CLOCK 3300 CCU 14 DIOJ 1 IDLE STORE 155

ROJUE 0 0 17 11 12 13 15 16 18 6 0 0 0 0 0 0 0  
 ROJUE 0 0 0 0 0 0 9 2 4 8 10 0 0 0 0 0 0 0  
 DIOUE 0 0 0 0 0 0 3 5 7 0 0 0 0 0 0 0 0 0

CLOCK 3350 CCU 14 DIOJ 1 IDLE STORE 155

ROJUE 0 0 17 11 12 13 15 16 18 6 0 0 0 0 0 0 0  
 ROJUE 0 0 0 0 0 0 9 2 4 8 10 0 0 0 0 0 0 0  
 DIOUE 0 0 0 0 0 0 3 5 7 0 0 0 0 0 0 0 0 0

CLOCK 3350 CCU 17 DIOJ 1 IDLE STORE 155

ROJUE 0 0 0 11 12 13 15 16 18 6 0 0 0 0 0 0 0  
 ROJUE 0 0 0 0 0 0 9 2 4 8 10 0 0 0 0 0 0 0  
 DIOUE 0 0 0 0 0 0 3 5 7 14 0 0 0 0 0 0 0 0

CLOCK 3350 CCU 17 DIOJ 3 IDLE STORE 155

ROJUE 0 0 0 1 11 12 13 15 16 18 6 0 0 0 0 0 0  
 ROJUE 0 0 0 0 0 0 9 2 4 8 10 0 0 0 0 0 0 0  
 DIOUE 0 0 0 0 0 0 0 5 7 14 0 0 0 0 0 0 0 0

CLOCK 3400 CCU 17 DIOJ 3 IDLE STORE 155

ROJUE 0 0 0 1 11 12 13 15 16 18 6 0 0 0 0 0 0  
 ROJUE 0 0 0 0 0 0 9 2 4 8 10 0 0 0 0 0 0 0  
 DIOUE 0 0 0 0 0 0 0 5 7 14 0 0 0 0 0 0 0 0

CLOCK 3450 CCU 1 DIOJ 3 IDLE STORE 155

ROJUE 0 0 0 11 12 13 15 16 18 17 6 0 0 0 0 0 0  
 ROJUE 0 0 0 0 0 0 9 2 4 8 10 0 0 0 0 0 0 0  
 DIOUE 0 0 0 0 0 0 0 5 7 14 0 0 0 0 0 0 0 0

CLOCK 3450 CCU 1 DIOJ 5 IDLE STORE 155

ROJUE 0 0 0 11 12 13 15 16 18 17 6 0 0 0 0 0 0  
 ROJUE 0 0 0 0 0 0 9 2 4 8 10 0 0 0 0 0 0 0  
 DIOUE 0 0 0 0 0 0 0 7 14 0 0 0 0 0 0 0 0 0

CLOCK 3500 CCU 9 DIOJ 5 IDLE STORE 95

ROJUE 0 0 0 0 3 12 13 15 16 18 17 1 6 0 0 0 0  
 ROJUE 0 0 0 0 0 0 9 2 4 8 10 11 0 0 0 0 0 0  
 DIOUE 0 0 0 0 0 0 0 7 14 0 0 0 0 0 0 0 0 0

CLOCK 3600 CCU 9 DIOJ 5 IDLE STORE 95

ROJUE 0 0 0 0 3 12 13 15 16 18 17 1 6 0 0 0 0  
 ROJUE 0 0 0 0 0 0 9 2 4 8 10 11 0 0 0 0 0 0  
 DIOUE 0 0 0 0 0 0 0 7 14 0 0 0 0 0 0 0 0 0



ROJEU	0	0	0	0	0	0	0	0	14	12	13	15	16	18	17	1	9	3	5	6
BOJEU	0	0	0	0	0	0	0	0	2	4	8	10	11	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 3950 CCU 14 DIOJ 0 IDLE STORE 95

ROJEU	6	0	0	0	0	0	0	0	0	12	13	15	16	18	17	1	9	3	5	7
BOJEU	0	0	0	0	0	0	0	0	2	4	8	10	11	0	0	0	0	0	0	0
DIOQUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 4000 CCU 15 DIOJ 0 IDLE STORE 365

ROJEU	6	0	0	0	0	0	0	0	0	16	18	17	1	9	3	5	7	2	14
BOJEU	0	0	0	0	0	0	0	0	4	8	10	11	12	13	0	0	0	0	0
DIOQUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 4100 CCU 15 DIOJ 0 IDLE STORE 365

ROJEU	6	0	0	0	0	0	0	0	0	16	18	17	1	9	3	5	7	2	14
BOJEU	0	0	0	0	0	0	0	0	4	8	10	11	12	13	0	0	0	0	0
DIOQUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 4150 CCU 15 DIOJ 0 IDLE STORE 365

ROJEU	6	0	0	0	0	0	0	0	0	16	18	17	1	9	3	5	7	2	14
BOJEU	0	0	0	0	0	0	0	0	4	8	10	11	12	13	0	0	0	0	0
DIOQUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CLOCK 4200 CCU 15 DIOJ 0 IDLE STORE 365

ROJEU	6	0	0	0	0	0	0	0	0	16	18	17	1	9	3	5	7	2	14
BOJEU	0	0	0	0	0	0	0	0	4	8	10	11	12	13	0	0	0	0	0
DIOQUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SWAPPING  
 SWAPPING  
 SWAPPING  
 SWAPPING  
 REGULAR INTRPT  
 I/O STARTED  
 I/O FINISHED  
 I/O STARTED  
 I/O FINISHED  
 I/O STARTED  
 I/O FINISHED  
 SWAPPING  
 SWAPPING  
 SWAPPING  
 REGULAR INTRPT  
 SWAPPING  
 SWAPPING  
 SWAPPING

3 0719407A  
 5 16756000  
 7 17461407

SWAPPING  
REGULAR INTRPT  
SWAPPING  
SWAPPING  
SWAPPING  
REGULAR INTRPT  
SWAPPING  
SWAPPING  
SWAPPING  
REGULAR INTRPT  
SWAPPING  
SWAPPING  
SWAPPING

ER ABORT# ABORT ADR: 016017 POI:000004  
ID: 000003  
ER ABORT# ABORT ADR: 024557 POI:000004  
ID: 000001  
ER ABORT# ABORT ADR: 025512 POI:000004  
ER ABORT# ABORT ADR: 022054 POI:000004  
ID: 000006  
ER ABORT# ABORT ADR: 020620 POI:000004  
ID: 000005  
ER ABORT# ABORT ADR: 024232 POI:000004  
ID: 000002  
ER ABORT# ABORT ADR: 014222 POI:000004  
NAME 242307  
ER ABORT# ABORT ADR: 020602 POI:000004  
ID: 000004  
ER ABORT# ABORT ADR: 025052 POI:000004  
NAME 242330  
ER ABORT# ABORT ADR: 025040 POI:000004  
NAME 242336  
ER ABORT# ABORT ADR: 025026 POI:000004  
NAME 242306  
ER ABORT# ABORT ADR: 025014 POI:000004  
NAME 240730  
ER ABORT# ABORT ADR: 025002 POI:000004  
NAME 242311  
ER ABORT# ABORT ADR: 024770 POI:000004  
NAME 242332  
ER ABORT# ABORT ADR: 024756 POI:000004  
NAME 242316  
ER ABORT# ABORT ADR: 024732 POI:000004  
NAME 002423  
ER ABORT# ABORT ADR: 024720 POI:000004  
NAME 112420  
ER ABORT# ABORT ADR: 024706 POI:000004  
NAME 301220  
ER ABORT# ABORT ADR: 014712 POI:000004

NAME 161211  
FR 480RTS APORT ADR: 024662 PDI:000004  
NAME 062131  
FR 480RTS APORT ADR: 015165 PDI:000004  
NAME 071230  
FR 480RTS APORT ADR: 024636 PDI:000004  
NAME 112731  
FR 480RTS APORT ADR: 015440 PDI:000004  
NAME 003210  
FR 480RTS APORT ADR: 024612 PDI:000004  
NAME 162016  
FR 480RTS APORT ADR: 024600 PDI:000004  
NAME 071627  
PUNSTREAM ANALYSTS TERMINATED



APPENDIX F - PROGRAM TRACE AND DUMP OUTPUTS

F.1 - Program Tracing

F.2 - Snapdump Output

## F.1 - PROGRAM TRACING

```

T  TYPE LABELS=(RECEIVE,SEND,INIT);
T  PROCEDURE BUFFER(VAR ENTRY:LABELS;VAR INP:INTEGER;VAR OUTP:INTEGER);MONITOR;
T  PROCEDURE PRINTPROC;
T  VAR ENTRY : LABELS ;
T  INP : INTEGER;
T  OUTP : INTEGER;
T  BEGIN
T  RELEASE(PRINTPROC);
T  AWAIT(STARTER);
T  WHILE TRUE DO
T  BEGIN
T  SNOOPY ON;
T  LOCK(LOCKV);
T  ENTRY:= RECEIVE;
T  FORK WITH TASKNAME(P) BUFFER(ENTRY,INP FORWARD ,OUTP);
T  AWAIT(P);
T  SNOOPY OFF;
T  CREGION; WRITELN(OUTP); CEND;
T  ND;
T  END;
T  *COMPILE  OPTIONS=IRSLMV,ELTNAME=TPF,BUFFER
T  PROGRAM DEFINE(BUFFER);
T  TYPE LABELS =(RECEIVE,SEND,INIT);
T  PROCEDURE BUFFER(VAR ENTRY:LABELS;VAR INP:INTEGER;VAR OUTP:INTEGER);
T  VAR FULL : BOOLEAN ;
T  CONTENTS: INTEGER ;
T  BEGIN
T  CASE ENTRY OF
T  RECEIVE:
T  BEGIN
T  NLOCK(LOCKV);
T  LOCK(RECEIVER);
T  IF NOT FULL THEN WAIT(RECEIVER)
T  ELSE UNLOCK(RECEIVER);
T  OUTP := CONTENTS ;
T  LOCK(SENDER);
T  FULL := FALSE ;
T  SIGNAL(SENDER);
T  QUIT;
T  END;
T  SEND:
T  BEGIN
T  UNLOCK(LOCKV);
T  LOCK(SENDER);
T  IF FULL THEN WAIT(SENDER)
T  ELSE UNLOCK(SENDER);
T  CONTENTS:= INP;
T  LOCK(RECEIVER);
T  FULL := TRUE;
T  SIGNAL(RECEIVER);
T  QUIT;

```



## F.2 - SNAPDUMP OUTPUT

```

T  TYPE LABELS=(RECEIVE,SEND,INIT);
T  PROCEDURE BUFFER(VAR ENTRY:LABELS;VAR INP:INTEGER;VAR OUTP:INTEGER);MONITOR;
T  PROCEDURE PRINTPROC;
T  VAR ENTRY : LABELS ;
T  INP      : INTEGER;
T  OUTP    : INTEGER;
T  BEGIN
T  RELEASE(PRINTPROC);
T  AWAIT(STARTER);
T  WHILE TRUE DO
T  BEGIN
T  LOCK(LOCKV);
T  ENTRY:= RECEIVE;
T  FORK WITH TASKNAME(PT) BUFFER(ENTRY,INP FORWARD ,OUTP);
T  AWAIT(P);
T  CREBION; WRITELN(OUTP); CEND;
T  END;
T  END;
T  *COMPILE      OPTIONS=IRSLMV,ELTNAME=TPF1,BUFFER
T  PROGRAM DEFINE(BUFFER);
T  TYPE LABELS =(RECEIVE,SEND,INIT);
T  PROCEDURE BUFFER(VAR ENTRY:LABELS;VAR INP:INTEGER;VAR OUTP:INTEGER);
T  VAR FULL      : BOOLEAN ;
T  CONTENTS: INTEGER ;
T  BEGIN
T  CASE ENTRY OF
T  RECEIVE:
T  BEGIN
T  SNAPDUMP(DUMP1);
T  UNLOCK(LOCKV);
T  LOCK(RECEIVER);
T  IF NOT FULL THEN WAIT(RECEIVER)
T  ELSE UNLOCK(RECEIVER);
T  SNAPDUMP(DUMP2);
T  OUTP := CONTENTS ;
T  LOCK(SENDER);
T  FULL := FALSE ;
T  SIGNAL(SENDER);
T  QUIT;
T  END;
T  SEND:
T  BEGIN
T  UNLOCK(LOCKV);
T  LOCK(SENDER);
T  IF FULL THEN WAIT(SENDER)
T  ELSE UNLOCK(SENDER);
T  CONTENTS:= INP;
T  LOCK(RECEIVER);
T  FULL := TRUE;
T  SIGNAL(RECEIVER);
T  QUIT;

```

BUFFER	S(I)	010735	011001					03 FEB 84	11:59:22
PRINT	S(I)	011102	011172	S(O)	040412	040414		03 FEB 84	11:59:00
CARD	S(I)	011173	011265	S(O)	040415	040417		03 FEB 84	11:58:40
MAIN	S(I)	011266	011434	S(O)	040420	040424		03 FEB 84	11:58:17
				S(O36)	040425	046424			

SYSRLIBS LEVEL  
 END MAP ERRORS: 0 TIME: 131724 STORAGE: 17792/4/040777/073777  
 BUSE INFOYABLE,INFOS  
 BUSE PAS000000,INFOS  
 BSETC,D  
 BXQT TPFS,MAIN

BDUMP1	010762	010760		11:59:53	02/03/84		1106 TIME SHARING EXEC	
040630						005700000000	005700000000	005700000000
040640	015700000001	005700000000	005700000000	005700000000	005700000000	005700000000	005700000000	005700000000
BDUMP2	011006	011004		11:59:53	02/03/84		1106 TIME SHARING EXEC	
040630						005700000000	005700000000	005700000000
040640	005700000000	005700000000	005700000000	005700000000	005700000000	005700000000	005700000000	005700000000
I								
BDUMP1	010762	010760		11:59:53	02/03/84		1106 TIME SHARING EXEC	
040630						005700000000	005700000000	005700000000
040640	015700000001	005700000000	005700000000	005700000000	005700000000	005700000000	005700000000	005700000000
BDUMP2	011006	011004		11:59:53	02/03/84		1106 TIME SHARING EXEC	
040630						005700000000	005700000000	005700000000
040640	005700000000	005700000000	005700000000	005700000000	005700000000	005700000000	005700000000	005700000000
2								
BDUMP1	010762	010760		11:59:53	02/03/84		1106 TIME SHARING EXEC	
040630						005700000000	005700000000	005700000000
040640	015700000001	005700000000	005700000000	005700000000	005700000000	005700000000	005700000000	005700000000
BDUMP2	011006	011004		11:59:53	02/03/84		1106 TIME SHARING EXEC	

## REFERENCES

1. Dijkstra, E.W., "The Structure of THE Multiprogramming System", CACM, Vol. 11, No. 4, pp. 341-346, 1968.
2. Dijkstra, E.W., "Cooperating Sequential Processes", in Programming Languages, New York: F. Genuy ed., Academic Press, pp. 43-112, 1972.
3. Dijkstra, E.W., "Hierarchical Ordering of Sequential Processes", in Operating Systems Techniques, New York: Academic Press, pp. 72-93, 1972.
4. Hoare, C.A.R., "Towards a Theory of Parallel Programming" in Operating Systems Techniques, New York: Academic Press, pp. 61-71, 1972.
5. Hansen, P.B., Operating System Principles, New Jersey: Englewood Cliffs, Prentice Hall, 1973.
6. Dijkstra, E.W., "Guarded Commands, Nondeterminicy and Formal Derivation of Programs", CACM, Vol. 18, No. 8, pp. 453-457, 1975.
7. Univac 1100 Series Executive System Programmer Reference, UP-4144.2, Vol. 2.
8. Univac 1100 Series Executive System Programmer Reference, UP-4144.2, Vol. 3.
9. Univac 1100 Series Systems Programming Student Guide, UE-986, 1980.
10. Stodola, F.W., "The Plus Programming Language", in Sperry Univac Series 1100 Systems USE-UUA/E Conference Notes, Sperry Univac Major Systems Division, 1979.
11. Univac 1100 Series Plus Programmer Reference, UP-9198.
12. Univac 1100 Series Cobol Programmer Reference, UP-8582.
13. Univac 1100 Series PL/I Programmer Reference, UP-8277.
14. Univac 1100 Series Macro Programmer Reference, UP-8336.
15. Pascal 8R1 Programmer Reference, DIKU-Copenhagen.

16. Hansen, P.B., "The Programming Language Concurrent Pascal", IEEE Trans. on Soft. Eng., Vol. SE-1, No.2, pp. 199-207, 1975.
17. Hoare, C.A.R., "Quicksort, Algorithm 64", CACM, Vol. 4, No. 7, pp. 321-322, 1961.
18. Hildebrand, F.B., Methods of Applied Mathematics, New Delhi: Prentice Hall, 1968.
19. Corbato, F.J., "PL/I as a Tool for System Programming", Datamation, Vol. 15, No. 5, pp. 69-76, 1969.
20. Irons, E.T., "Experience with an Extensible Language", CACM, Vol.13, No. 1, pp. 31-40, 1970.
21. Lych, W.C., "An Operating System Designed for the Computer Utility Environment", in Operating Systems Techniques, New York: Academic Press, pp. 341-350, 1972.
22. Howarth, D.J., "A Re-appraisal of Certain Design Features of Atlas 1 Supervisory System", in Operating Systems Techniques, New York: Academic Press, pp. 371-377, 1972.
23. Saltzer, J.H. and J.W. Gintell, "The Instrumentation of Multics", CACM, Vol. 13, No. 8, pp. 495-500, 1970.
24. Hansen, P.B., "Structural Programming", CACM, Vol. 15, No. 7, pp. 574-578, 1972.
25. Hoare, C.A.R., "Monitors: An Operating System Structuring Concept", CACM, Vol. 17, No. 10, pp. 549-557, 1974.
26. Hansen, P.B., "The Solo Operating System: A Concurrent Pascal Program", Softw. Pract. Exp., Vol. 6, No. 2, pp. 141-150, 1976.
27. Hansen, P.B., "The Solo Operating System: Job Interface", Softw. Pract. Exp., Vol. 6, No. 2, pp. 151-164, 1976.
28. Hansen, P.B., "The Solo Operating System: Processes, Monitors and Classes", Softw. Pract. Exp., Vol. 6, No. 2, pp. 165-200, 1976.
29. Hansen, P.B., The Architecture of Concurrent Programs, New Jersey: Englewood Cliffs, Prentice Hall, 1977.
30. Kaubisch, W.H., R.H. Perrot & C.A.R. Hoare, "Quasiparallel Programming", Softw. Pract. Exp., Vol. 6, No. 3, pp. 341-356, 1976.
31. Wirth, N., "Modula: A Language for Modular Multiprogramming", Softw. Pract. Exp., Vol. 7, No. 1, pp. 3-35, 1977.

32. Wirth, N., "The Use of Modula", Softw. Pract. Exp., Vol. 7, No. 1, pp. 37-66, 1977.
33. Wirth, N., "Design and Implementation of Modula", Softw. Pract. Exp., Vol. 7, No. 1, pp. 67-84, 1977.
34. Welsh, J., and D.W. Bustard, "Pascal-Plus - Another Language for Modular Programming", Softw. Pract. Exp., Vol. 9, No. 11, pp. 947-957, 1979.
35. Conway, M.E., "Design of a Separable Transition-Diagram Compiler", CACM, Vol. 6, No. 7, pp. 396-408, 1963.
36. Knuth, D., Fundamental Algorithms, The Art of Computer Programming Vol. 1, Mass. Reading: Addison Wesley, 1973.
37. Kriz, J. and H. Sandmayer, "Extension of Pascal by Coroutines and Its Application to Quasi-parallel Programming and Simulation", Softw. Pract. Exp., Vol. 10, No. 10, pp. 773-789, 1980.
38. Holt, R.C., G.S. Graham, E.D. Lazowska, and M.A. Scott, Structured Concurrent Programming, Mass. Reading: Addison Wesley, 1978.
39. Hoare, C.A.R., "Proof of Correctness of Data Representation", Acta Informatica, Vol. 1, pp. 271-281, 1972.
40. Owicki, S., and D. Gries, "Verifying Properties of Parallel Programs - An Axiometric Approach", CACM, Vol. 19, No. 5, pp. 279-288, 1976.
41. Hoare, C.A.R., "Communicating Sequential Processes", CACM, Vol. 21, No. 8, pp. 666-667, 1978.
42. Hansen, P.B., "Distributed Processes - A Concurrent Programming Concept", CACM, Vol. 21, No. 11, pp. 934-941, 1978.
43. Barnes, J.G.P., "An Overview of Ada", Softw. Pract. Exp., Vol. 10, No. 11, pp. 851-887, 1980.
44. Roberts, E.S., J.A. Evans, C.R. Morgan and E.M. Clark, "Task Management in Ada - A Critical Evaluation for Real-time Multiprocessors", Softw. Pract. Exp., Vol. 11, No. 10, pp. 1019-1053, 1981.
45. Hansen, P.B., "Edison - A Multiprocessor Language", Softw. Pract. Exp., Vol. 11, No. 4, pp. 325-362, 1981.
46. Hansen, P.B., "The Design of Edison", Softw. Pract. Exp., Vol. 11, No. 4, pp. 363-396, 1981.



47. Andrews, G.R., "The Distributed Programming Language SR - Mechanism, Design and Implementation", Softw. Pract. Exp., Vol. 12, No. 8, pp. 719-754, 1982.
48. Cook, R.P., "\*Mod - A Language for Distributed Programming", IEEE Trans. on Soft. Engr., Vol. SE-6, pp. 563-571, 1980.
49. Straunstrup, J., "Message Passing Communication Versus Procedure Call Communication", Softw. Pract. Exp., Vol. 12, No. 3, pp. 223-234, 1982.
50. Roper, T.J. and C.J. Barter, "A Communicating Sequential Language and Implementation", Softw. Pract. Exp., Vol. 11, No. 11, pp. 1215-1234, 1981.
51. Eswaran, K.P., J.N. Gray, R.A. Lorie and I.L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System", CACM, Vol. 19, No. 11, pp. 624-633, 1976.
52. Tsichritzis, D.C., and P.A. Bernstein, Operating Systems, London: Academic Press, 1974.
53. Holt, R.C., "Some Deadlock Properties of Computer Systems", Computing Surveys, Vol. 4, No. 3, pp. 549-557, 1972.
54. Sauer, C.H., E.A. MacNair and S. Salza, "A Language for Extended Queuing Network Models", IBM Journal of Research and Development, Vol. 24, No. 6, pp. 747-755, 1980.
55. Brent, R.P., "The Parallel Evaluation of General Arithmetic Expressions", JACM, Vol. 21, No. 2, pp. 201-206, 1974.
56. Even, S., "Parallelism in Tape Sorting", CACM, Vol. 17, No. 4, pp. 202-204, 1974.
57. Garvil, F., "Merging with Parallel Processors", CACM, Vol. 18, No. 10, pp. 588-591, 1975.
58. Lewitt, K.N., and W.H. Kautz, "Cellular Arrays for the Solution of Graph Problems", CACM, Vol. 15, No. 9, pp. 789-801, 1972.
59. Muraoka, Y., and D.J. Kuck, "On the Time Required for a Sequence of Matrix Products", CACM, Vol. 16, No. 1, pp. 22-26, 1973.
60. Marsland, T.A., and M. Campbell, "Parallel Search of Strongly Ordered Game Trees", Computing Surveys, Vol. 14, No. 4, pp. 531-552, 1982.
61. Bonner, S., and K.G. Shin, "A Comparative Study of Robot Languages", Computer, Vol. 15, No. 12, pp. 82-96, 1982.