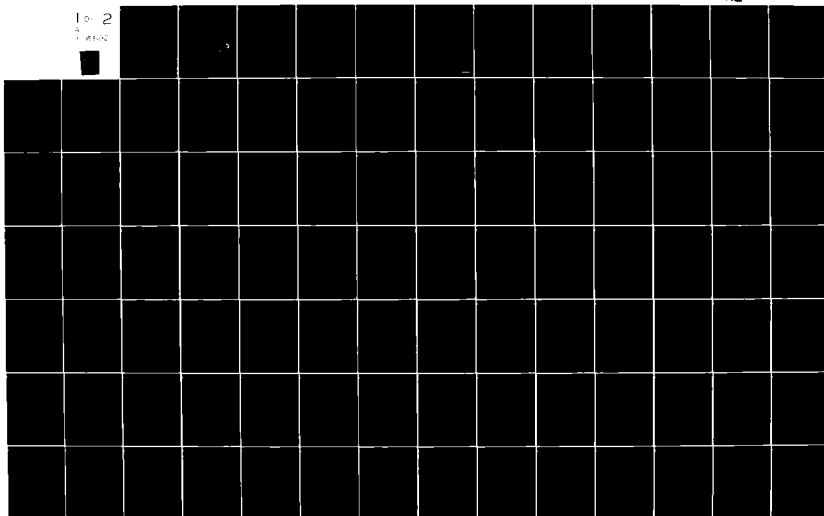


AD-A116 592 ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/8 9/2
SHELL: A SIMULATOR FOR THE SOFTWARE TEST VEHICLE OF THE INFOPL--ETC(U)
JAN 82 Y TO N00039-81-C-0663
UNCLASSIFIED CISR-M010-8201-08 NL

1 of 2
4 1/2

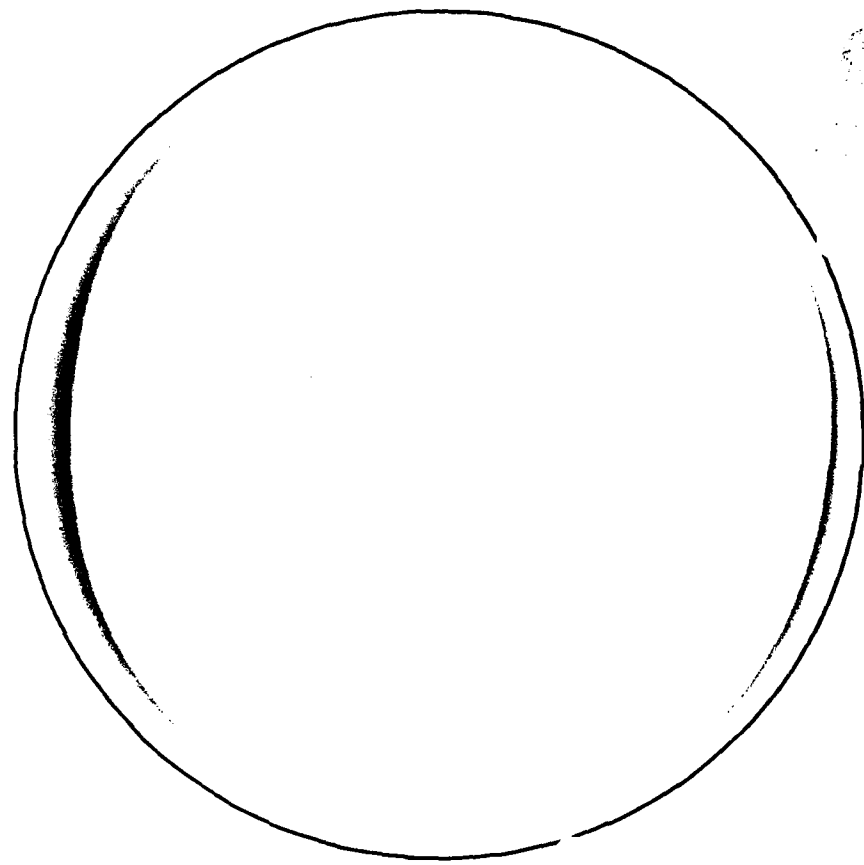


AD A116592

DTIC FILE COPY



12



DTIC

1967

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

Center for Information Systems Research

Massachusetts Institute of Technology
Sloan School of Management
77 Massachusetts Avenue
Cambridge, Massachusetts, 02139

82

Contract Number N00039-81-C-0663 (MIT # 91445)
Internal Report Number M010-8201-08
Deliverable Number 7

12

SHELL:
A SIMULATOR FOR THE
SOFTWARE TEST VEHICLE OF THE
INFOPLEX DATABASE COMPUTER

Technical Report #8

By
Tak To

January, 1982

DTIC
SELECTED
JUL 7 1982
H

Principal Investigator:
Professor Stuart E. Madnick

Prepared for:
Naval Electronics System Command
Washington, D.C.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report #8	2. GOVT ACCESSION NO. AD-A115 592	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SHELL: A Simulator for the Software Test Vehicles of the INFOPLEX Database Computer		5. TYPE OF REPORT & PERIOD COVERED
7. AUTHOR(s) Tak To		6. PERFORMING ORG. REPORT NUMBER M010-8201-08
9. PERFORMING ORGANIZATION NAME AND ADDRESS Sloan School of Management 50 Memorial Drive Cambridge, MA 02139		8. CONTRACT OR GRANT NUMBER(s) N0039-81-C-0663
11. CONTROLLING OFFICE NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE January 1982
		13. NUMBER OF PAGES 161
		15. SECURITY CLASS. (of this report) unclassified
		16a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) database computer, database management system, Software Test Vehicle, distributed operating system		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The INFOPLEX database computer is a special computer designed for large scale information management. The information management functions are decomposed into a functional hierarchy implemented by a hierarchy of micro-processors. Decentralized control mechanisms are used to coordinate the activities of individual modules in the hierarchy.		

Before realizing INFPLEX in hardware, it is essential to validate all the design details via a software test vehicle (STV). A simulator (SHELL) is built to provide the necessary facilities for the operating of this software test vehicle. It has two parts: an event simulator which simulates the operation of the afore mentioned hardware configuration; and an operating system emulator which provides the environment for testing the multi-threading, parallel processing application programs. SHELL is meant to be used as the Control Structure portion of the STV project, which includes two additional parts, the Functional Hierarchy STV and the Storage Hierarchy STV.

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	



**SHELL:
A SIMULATOR FOR THE
SOFTWARE TEST VEHICLE OF THE
INFOPLEX DATABASE COMPUTER**

by

Tak To

Submitted to the Department of Electrical Engineering and
Computer Science, on May 22, 1981
in partial fulfillment of the requirements
for the degree of the Bachelor of Science

ABSTRACT

The INFOPLEX database computer is a special computer designed for large scale information management. The information management functions are decomposed into a functional hierarchy implemented by a hierarchy of micro-processors. Decentralized control mechanisms are used to coordinate the activities of individual modules in the hierarchy.

Before realizing INFOPLEX in hardware, it is essential to validate all the design details via a software test vehicle. A simulator (SHELL) is built to provide the necessary facilities for the operating of this software test vehicle. It has two parts: an event simulator which simulates the operation of the afore mentioned hardware configuration; and an operating system emulator which provides the environment for testing the multi-threading, parallel processing application programs.

SHELL facilitates the following goals: 1) the validation of the logic and algorithm of the application programs; 2) the validation of the multi-threaded, distributed control design of the functional hierarchy; 3) the investigation of detailed hardware designs (e.g., buffer sizes); and 4) the collection of performance data as a basis for further design considerations.

Table of Content

ABSTRACT	i
ACKNOWLEDGEMENTS	ii
TABLE OF CONTENT	iii
TABLE OF FIGURES	vi
TABLE OF TABLES	vi
1.0 <u>INTRODUCTION</u>	1
1.1. <u>INFOPLEX</u>	1
1.2 Scope of the Project	2
1.3. Structure of this Paper	3
2.0 <u>THE HARDWARE ENVIRONMENT OF INFOPLEX</u>	4
2.1. Overview	4
2.2. GC-LOS Protocol	7
3.0 <u>THE SIMULATOR</u>	10
3.1. The Structure of the Simulator	10
3.1.1. Synchronous and Asynchronous Activity	10
Handlers	
3.1.2. Events	11
3.1.3. Event and Server Queues	12
3.2. Operation of the Simulator	14
3.3. SEND	15
3.4. GCER: Outgoing Messages	16
3.5. GBER	16
3.6. GCER: Incoming Messages	17
3.7. AAHER	18
3.8. SAHER	18
3.9. Data Transfer	18
4.0 <u>THE LOCAL OPERATING SYSTEM</u>	20
4.1. Overview	20
4.2. System Service Requests	21
4.3. The Local Operating System as a Network of	22
Co-Routines	
5.0 <u>THE LOCAL OPERATING SYSTEM EMULATOR</u>	24
5.1. Overview	24
5.2. Process Control	24
5.2.1. Context Switching	24
5.2.2. The Virtual Process Status Table	25
5.2.3. Scheduling	26
5.2.4. Process Termination	26
5.3. Inter-Process Communication	27
5.3.1. Outgoing Mail: SEND	27

5.3.2.	Incoming Mail	27
5.3.3.	Indefinite Wait	28
5.4.	Inter-Process Synchronization	28
5.4.1.	SYNC	29
5.5.	System Initialization	29
6.0	<u>PROGRAM STRUCTURE AND EXECUTION LOGIC OF SHELL</u>	31
6.1.	Program Structure of SHELL	31
6.2.	Format of Major Data Structures	33
6.2.1.	LOS	33
6.2.2.	VP	35
6.2.3.	SVR	37
6.3.	Message Prefixes	39
6.3.1.	PF_ADDR	39
6.3.2.	PF_S	42
6.3.3.	PF_MSG	42
6.3.4.	PF_LEVEL	42
6.4.	The Execution Logic of the Context Switching Mechanism	42
6.4.1.	The Stack Space	42
6.4.1.1.	GET4	43
6.4.2.	Saving a Process	46
6.4.3.	Restoring a Process	46
7.0	<u>INTERFACE WITH APPLICATION PROGRAMS -- A USER'S GUIDE TO SHELL</u>	52
7.1.	To Register a Top Level Procedure with SHELL	53
7.2.	Parameters	53
7.2.1.	Compile Time Parameters	53
7.2.2.	Run Time Parameters	56
7.2.2.1.	Number of Processors	56
7.2.2.2.	RRATE	56
7.2.2.3.	THRU PUT	56
7.2.2.4.	DELAY_GB_GC, DELAY_GC_GB	56
7.2.2.5.	TERMINALS	57
7.3.	Initialization	57
7.4.	Utility Procedures	57
7.4.1.	STIMER	57
7.4.2.	SEND	58
7.4.3.	SYNC	58
7.4.4.	WAIT	58
7.4.5.	FINISH	59
7.5.	System Service Requests	59
7.5.1.	Start a New Virtual Process	59
7.6.	External Variables	61
7.6.1.	VP: Information about the Current Virtual Process	61
7.6.1.1.	VP.WAIT.MSG	61
7.6.1.2.	VP.LEVEL	61
7.6.1.3.	VP.VPID	62
7.6.1.4.	VP.VTIME	62
7.7.	Caveats & Restrictions	62

7.7.1. Local Variables	62
7.7.2. Messages	62
7.7.3. The REPORT option	63
7.8. STATS	63
7.9. A Sample Program	63
7.9.1. Command BUILD	67
7.9.2. Command SEND	67
7.9.3. Command WAIT	68
7.9.4. Commands SYNC and FINISH	68
7.10. A Sample Simulation Session	68
8.0 <u>CONCLUSION & OBSERVATIONS</u>	75
BIBLIOGRAPHY	77
APPENDIX A: Listings of PL/1 Procedures	78
APPENDIX B: Listings of Assembly Language Routines	124
APPENDIX C: Listings of Macro Files (Data Declarations)	129
APPENDIX D: Listings of Macro Files (Entry Declarations)	150

Table of Figures

Figure 2.1	Hardware Configuration of the Functional . . .	5
	Hierarchy of INFOPLEX	
Figure 2.2	Protocol Between the Gateway Controller & . . .	9
	the Local Operating System	
Figure 6.1	Functional Relationship of the Major Modules . . .	32
	of SHELL	
Figure 6.2	Format of LOS	34
Figure 6.3	Format of VP	36
Figure 6.4	Format of SVR	38
Figure 6.5.1-	Data Structures Used at Various Stages . . .	40
Figure 6.5.2	of Message Transmission	
Figure 6.6	Format of the Dynamic Storage Area (DSA) . . .	44
Figure 6.7	Format of the Task Communication Area (TCA) . . .	45
Figure 6.8.1-	State of the Stack During Context	47
Figure 6.8.4	Switching	
Figure 7.1	Listing of Procedure EXECUTE	54
Figure 7.2	Format of Argument for System Service . . .	60
	Request 1 (Starting a New Virtual Process)	
Figure 7.3.1-	Listing of Procedure TERM	64
Figure 7.3.3		
Figure 7.4.1-	Sample Simulation Session	69
Figure 7.4.3		
Figure 7.5	Statistics of the Sample Simulation Session . . .	73
Figure 7.6	Chronology of Events in the Sample	74
	Simulation Session	

Table of Tables

Table 7.1	List of External Identifiers	55
-----------	--	----

1.0 INTRODUCTION

1.1. INFOPLEX

The INFOPLEX database computer is a special purpose computer designed for large-scale information management. [Madnick] The specific objective of INFOPLEX includes providing substantial information management improvement over conventional architecture (e.g., up to 1000-fold increases in throughput) supporting very large complex databases (e.g., over 100 billion bytes of structured data), and providing extremely high reliability.

To achieve these goals, the architecture of INFOPLEX is built around the concept of hierarchical decomposition. The functions of information management are decomposed into a functional hierarchy. Each sub-function with a level of the hierarchy is implemented by means of a complex of processors. Highly parallel operations at each level and among levels boost the performance as well as the reliability. [Hsu 1]

A large capacity, cost-effective memory with rapid access time is realized using an 'intelligent' storage hierarchy. With a high degree of parallelism in operation, the storage hierarchy is able to support the storage requirements of the INFOPLEX functional hierarchy. The control of the storage hierarchy is distributed and micro-processors are used to implement

these control mechanisms.

1.2. Scope of the Current Project

Before realizing INFOPLEX in hardware, it is essential to validate all the design details via a software test vehicle. The current project is to design and implement a set of programs so as to provide the necessary facilities for the operation of this software test vehicle.

SHELL is a set of PL/1 programs on an IBM 370 VM/CMS system. It can be divided logically into two parts. The first is a special purpose event simulator which simulates the parallel operating environment in which a number of machines -- processors, gateway controllers etc. -- execute independently and compete for resources. Of specific interest is the communication mechanism among the processors. The interaction among the different machines along the data path is detailedly tracked.

The simulator can realistically reflect the actual operation of the hardware environment, hence the hardware design can be tested before it is actually built. In addition, performance data are gathered to serve as a basis for further detailed specifications (e.g., sizes of various buffers). Even though the main purpose is to test the functional hierarchy, some of the results can be applied to that of the storage hierarchy as well.

The second part of SHELL is an operating system emulator. It provides the parallel operating environment of a multi-processor, multi-process operating system, as well as the facilities of inter-process communication and inter-process synchronization. Application programs written in PL/1 can run directly under this emulated operating system.

Thus, the algorithms in the program modules of the functional hierarchy can be fully tested. In addition, since the simulator provides a realistic reflection of the actual execution speed, the relative efficiencies of the different algorithms and the different approaches of functional decomposition can be measured.

1.3. Structure of this paper

Following this introduction, Chapter 2 outlines the hardware configuration of INFOPLEX and Chapter 3 describes how it is simulated by the simulator. Chapter 4 describes the functions of the local operating system in INFOPLEX, and Chapter 5 describes how they are emulated by the emulator. Chapter 6 describes the program organization of SHELL, its execution logic, and the data structures it uses. Chapter 7 presents the user-interface aspects of SHELL as a user's guide. A test program and a sample simulation session are included as examples. Chapter 8 is the conclusion.

2.0 THE HARDWARE ENVIRONMENT OF INFOPLEX

2.1. Overview

The hardware environment of the INFOPLEX datacomputer is shown in Figure 2.1. [Hsu 2] It is composed of multiple processors and memory modules. Processors and memory modules are grouped into clusters, each cluster corresponding to a level in either the storage heirarchy or the functional heirarchy. At each level, there is a common pended bus that joins all the hardware. For communication among levels, there is a global bus that connects to the local bus of each level via a gateway controller.

At each level, a number of processors sharing a large memory module operate a multi-processor operating system. (The local operating system of that level.) Each processor may have additional private RAMs, ROMs, or co-processors. The processors are homogeneous -- each processor executes the same operating system code to dispatch itself and there is no overall master supervisor processor.

Each data path connecting to the local pended bus (including that of the gateway controller) is buffered by a latch/buffer so that each device can access the bus asynchronously. It eliminates the necessity of special software buffering while waiting for a resource -- the shared memory or the gateway controller. The connection

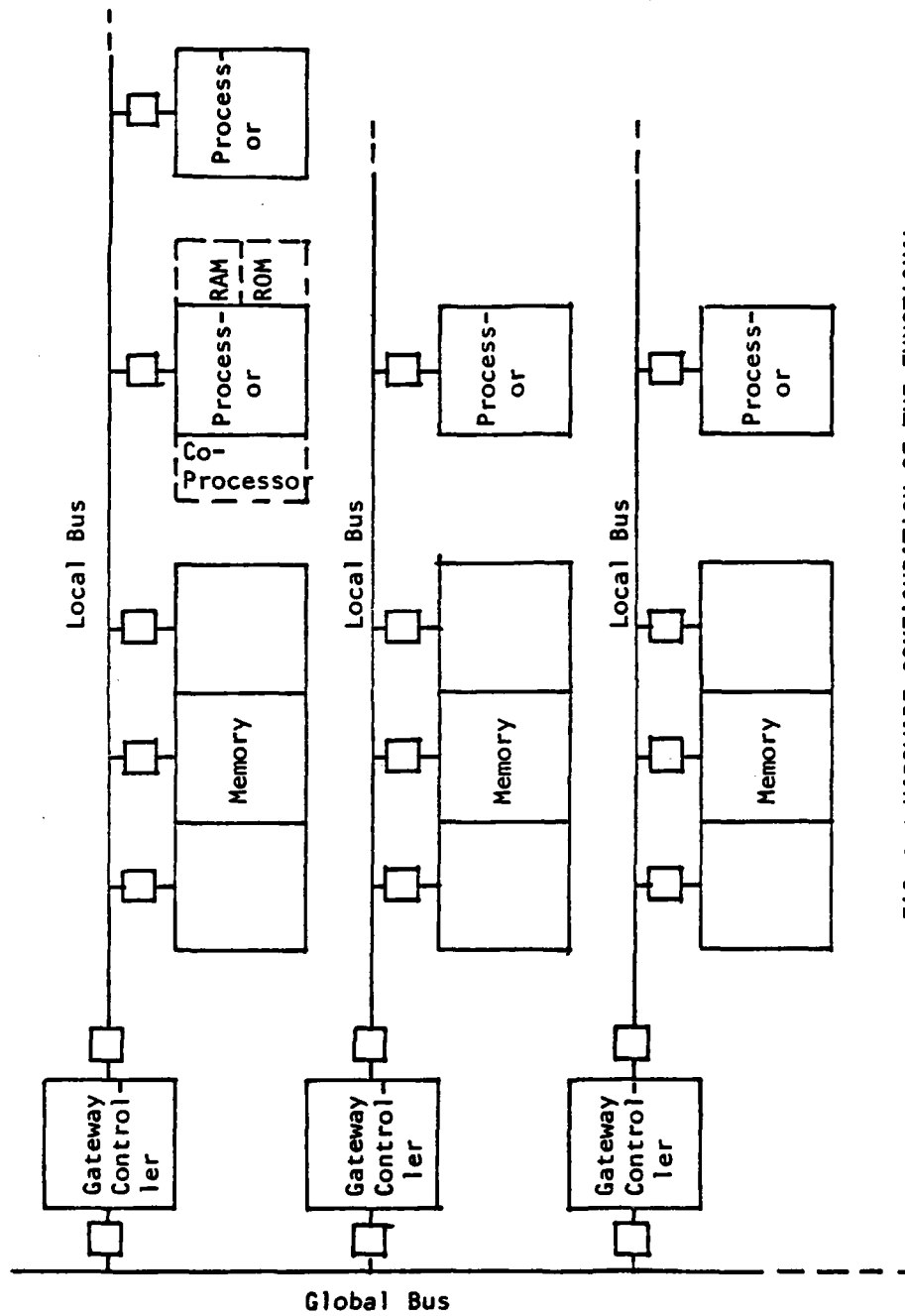


FIG. 2.1 HARDWARE CONFIGURATION OF THE FUNCTIONAL HIERARCHY OF INFOPLEX

between the global bus and the gateway controllers is similar in nature.

There is no shared memory between levels. Data are sent from the processor of one level, through the local gateway controller, the global bus, and the foreign gateway controller to the memory module of the destination level.

The data transfer from the processor to the gateway controller is memory-mapped. When a processor wants to transfer data to another level, it sends the data to a virtual address which will be recognized by the local gateway controller. (The destination level will either be coded in the address, or be a prefix in the data stream.) The gateway controller, serving as an entrepot between the local bus and the global bus, will send the data to the global bus.

The global bus is interleaved: data streams with different sources and destinations can share the global bus at the same time.

The gateway controller at the destination level will pick up the data stream from the global bus and deposit it (without the intervention of a processor) into the local shared memory module.

Because software buffering is not necessary along the path between the sending processor and the destination memory, the data transfer between two levels is

immediate in nature. To maintain a high throughput rate of interlevel data transfer, a special protocol is set up between a gateway controller and the local operating system.

2.2. GC-LOS Protocol

Two types of data transfer are recognized in this protocol: 'data blocks' and 'service requests.' Data blocks (or type 'D') are fixed sized, while service requests (or type 'S') are variable length data streams.

The local operating system maintains the following: a Data Requst Queue (DRQ), a Service Requst Queue (SRQ), and a Data Block Buffer (DBB).

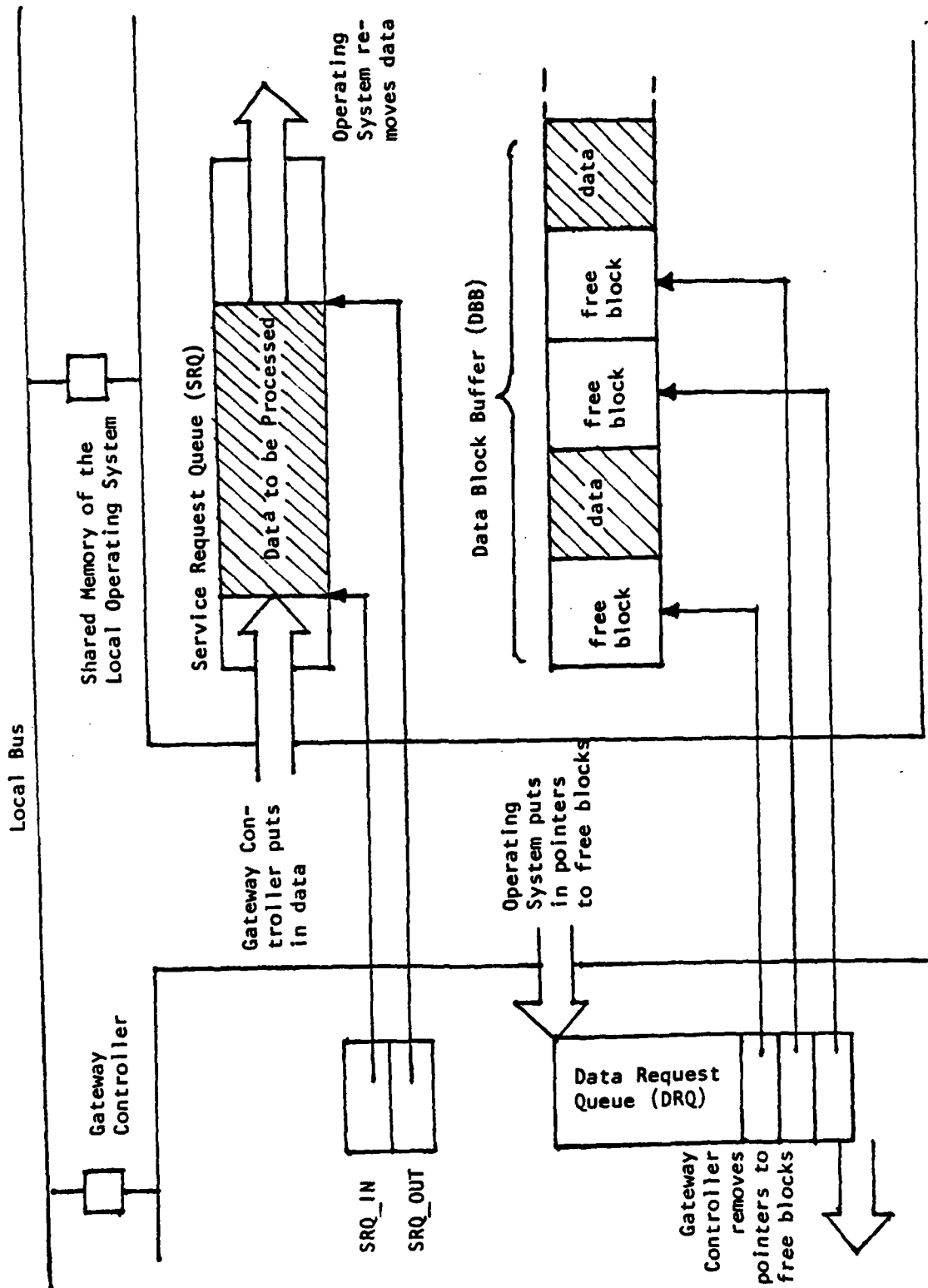
The SRQ is a software FIFO buffer (ring buffer) residing in the shared memory, with the port pointers SRQ_IN and SRQ_OUT residing in the gateway controller (as special registers, for example). The processors of the local operating system can access the pointers directly. SRQ_IN points to the next available location in the SRQ for storing incoming S-type data, and SRQ_OUT points to the first location of the next type S-type data stream for the operating system to process. SRQ_OUT chases SRQ_IN as S-type data are put in and processed. The local operating system is responsible for processing the data (or at least removing it from the SRQ) fast enough, and for maintaining a reasonable size of empty space in the SRQ.

The DRQ is a FIFO buffer (either software or hardware) residing in the gateway controller. The DRQ contains pointers to the free blocks in the DBB into where the gateway controller can put incoming data blocks (type 'D'). When a D-type data block arrives, the gateway controller obtains a pointer from the DRQ and put the data block into the free block in DBB pointed to by the pointer. The local operating system is responsible for maintaining a reasonable number of free blocks in the DBB and also for putting the pointers into the DRQ.

The DBB need not be contiguous. In fact, the DBB need not be in a fixed location. The local operating system does not have to process the incoming blocks as long as it can allocate enough free block and fill the DRQ with enough pointers.

The protocol is illustrated in Figure 2.2.

FIG 2.2 PROTOCOL BETWEEN GATEWAY CONTROLLER & THE LOCAL OPERATING SYSTEM



3.0 THE SIMULATOR

3.1. The Structure of the Simulator

The simulator consists of an event scheduler and a set of simulation procedures, each simulates the operation of a specific type of machine (processor, gateway controller, etc).

In the hardware environment of INFOPLEX, more than one machine (processor or gateway controller) can be active at the same time. This concurrency is simulated by dividing the operation of each machine into time slices, and by having the simulator executing the time slices of each machine in an interleaving manner.

3.1.1. Synchronous and Asynchronous Activity Handlers

Because in reality, a processor running a user program can be interrupted to handle an incoming message or serve other operating system functions, user programs and operating system functions can run 'simultaneously' within the same processor. Thus, for convenience and clarity, each processor is divided logically into two parts: an Asynchronous Activity Handler (AAH) and a Synchronous Activity Handler (SAH). Events which can happen and must be handled asynchronously with the scheduling cycle (of the operating system), such as the arrival of a message from another level, are handled by the AAH. (In essence, the AAH is like an interrupt

handler.) Scheduling and running user programs, and other events which could be conveniently processed at scheduling cycle breaks, are handled by the SAH. These two parts are treated by the simulator as if they were two separate and independent machines.

This separation bases on the assumption that the time consumed by handling asynchronous events is much smaller than that consumed by synchronous events, so that the work of the AAH would not slow down the SAH. The performance data gathered during simulation (see Chapter 7) will be used to verify this assumption.

3.1.2. Events

In many cases, during a time slice, a machine may stay idle indefinitely waiting for some events to happen. For example, a gateway controller idles until either an incoming or outgoing message arrives. It would be wasteful to simulate this kind of indefinite loop. Moreover, the simulator has to stop the simulation of the waiting machine at some point so that it can run the time slice of another machine to materialize the expected event. Therefore, time slices are scheduled only when a machine has something meaningful to do.

To formalize this concept, we say that time slices are triggered by events. An event is a condition which causes a machine to do something, and will usually cause other events to occur. The machine is known as the ser-

ver or the handler of the event.

3.1.3. Event and Server Queues

To keep track of all the events in a system, the simulator maintains a set of event queues. Events in a queue are ordered by the time they occurred. An event can appear in only one queue and the grouping of events into queues is such that two events are put in the same queue if and only if they can be handled by the same set of machines (servers).

The simulator also maintains a set of server queues, which keep track of when each machine will be available to handle an event. (It is assumed that a machine can handle only one event at a time.) A machine can appear in only one server queue and the grouping of machines into queues is such that two machines are in the same queue if and only if they handle the same set (i.e., type) of events.

In the current configuration of the simulator, there are three event queue - server queue pairs for each level.

In the first server queue, the gateway controller is the sole server. The corresponding event queue contains the events which are the arrivals of messages to the gateway controller, from either the global bus or the processors of the local operating system at that level.

In the second queue pair, the server queue contains the AAH's of the level. The event queue contains the events which are the arrival of messages at the local operating system. (The messages are already in the DBB or the SRQ, being put there by the gateway controller.)

In the third queue pair, the server queue contains the SAH's of the level. The event queue contains several types of events: LOGMSG, SYNC, WAIT, FINISH and INIT. An event of type LOGMSG is the arrival of a logical message at the mail box of one of the virtual processes. A logical message is a logical unit of data transfer between two virtual processes, and can be queued by either the SAH or the AAH. Events of type SYNC, WAIT and FINISH are issued by virtual processes to control the scheduling process. Events of type INIT are queued by the simulator at the beginning of simulation for initializing the local operating system. (Chapter 5 will deal with the operation of a local operating system in detail.)

In addition to the queues mentioned above, the simulator also maintains an event queue for the arrival of messages at the global bus. The global bus simulator is the sole server in the corresponding server queue.

3.2. Operation of the Simulator

The simulator runs by simulation cycles. At the beginning of each cycle, the start time of the event at the head of each event queue is calculated.

The start time of an event is the earliest time at which a machine is available to handle the event. This may be later than when the event occurs because there may not be any free machines then. The simulator selects the earliest serviceable event and 'dispatches' the server machine by executing the procedure which simulates the machine. The event is removed from the queue and is passed to the simulation procedure as an argument.

The simulation procedure will most probably cause more events to occur. Simulation will stop when there are no events in all the queues.

For each machine, the simulator keeps track of how far into simulation time (STIME) scale the machine has run in an STIME clock. The simulation procedure for the machine is responsible for updating the STIME clock to reflect the operating speed of that machine.

When the simulation procedure returns, it will be requeued into the server queue with its updated STIME clock. The simulator will then go into another cycle.

3.3. SEND

SEND is the procedure which simulates the action of a processor in transferring data to the gateway controller. Further description of SEND is found in Chapter 5.

When a virtual process wants to send a message to another level, it calls the procedure SEND. SEND queues the message as an event of type MSGOUT in the event queue of the local gateway controller and stamps it with the current STIME clock of the processor plus a delay parameter. The STIME clock of the processor is then incremented by an amount which is a function of the length of the message and an overall throughput parameter.

Note that this simulation algorithm corresponds to three different scenarios of how the message is sent in reality. In the first one, when the user program wants to send a message, the processor immediately gains access to the local gateway controller and performs the data transfer. In the second one, the message from the virtual process is queued by the operating system in the memory while it keeps trying to get access to the gateway controller. Within a reasonable delay, the operating system gains control of the gateway controller and transfer the message. In the third scenario, the message is queued in a specific place in the memory from where it will be picked up by the gateway controller at

a later time without the intervention of a processor. By varying the delay parameter, the effect of all of the three alternative configurations can be satisfactorily simulated.

3.4. GCER: Outgoing Messages

GCER is the procedure which simulates the action of a gateway controller. For an outgoing message, GCER queues it in the event queue of the global bus and stamps it with its current STIME clock of the gateway controller plus a delay parameter. The STIME clock of the gateway controller is then incremented by an amount which is a function of the length of the message and an overall throughput rate parameter.

Note that since GCER handles only one message (either outgoing or incoming) at a time, the start time of an event handled by GCER can be considerably later than the sending time of the message. It is assumed that this delay in the start time is small enough so that it would not affect the sending machine. To verify this assumption, GCER keeps track of the average delay in start time for both incoming and outgoing messages.

3.5. GBER

GBER is the simulation procedure for the global bus. GBER queues incoming messages to the event queues of their respective destination gateway controllers and

stamps them with their arrival time plus a delay parameter. Since the global bus can handle interleaved messages, the service delay is assumed to be always zero and the STIME clock of the global bus is set to the arrival time of each message.

It is assumed that the overall effect of the global bus between the sending and receiving gateway controllers is a simple delay with a reasonable standard deviation. This assumption is valid if the time distribution of the messages is sparse enough and that the nature of the pending bus will even out the throughput rate of messages at different load conditions of the global bus. To verify this assumption, GBER keeps track of the arrival time and duration of each message.

3.6. GCER: Incoming Messages

GCER handles an incoming message in a way similar to that of handling an outgoing one. It queues the message in the event queue of the local AAH's, and time stamps it with the current STIME clock plus an amount which is a function of the length of the message and an overall throughput rate parameter. The STIME clock is incremented by the same amount.

Note that it is assumed that the gateway controller can handle interleaved incoming messages.

3.7. AAHER

AAHER is the procedure which simulates the action of an AAH. AAHER removes messages from the SRQ or the DBB and reassembles them if necessary to form logical messages. It checks to see if the logical message is addressed to the operating system itself: if it is, AAHER will process it immediately. Otherwise it will be queued in the event queue of the local SAH's with type equal to LOGMSG and STIME equal to the STIME clock of the AAH. The LOGMSG event will be handled by the SAHER.

3.8. SAHER

SAHER is the procedure which simulates the action of a SAH. A full description of its operation is given in Chapter 5.

In short, each activation of SAHER resembles a scheduling cycle (of the operating system): logical messages are distributed to their respective mailboxes, the waiting condition of each process is then evaluated, and if there is any runnable process, one is selected and executed.

3.9. Data Transfer

In the simulator, a message is represented by a POINTER to a data structure containing the content of the message. The simulator does not access the data structure directly, and a message transmission is

simulated by passing the pointer from one place to another.

Also, the simulator does not maintain the SRQ's, the DRQ's or the DBB's. Instead, for each level, the simulator keeps track of the size of a virtual SRQ and that of a virtual DBB. When a S-type message arrives to a local operating system, the size of the virtual SRQ will be incremented (by the length of the S-type message); and when a message is processed, the size will be decremented. Thus, the simulator knows how much data is supposed in the SRQ even though there is no copying of data involved. D-type messages are treated similarly, except that the size of a DBB is kept in terms of blocks (not bytes).

4.0 THE LOCAL OPERATING SYSTEM

4.1. Overview

The local operating system at each level is a multi-processor, multiprocess operating system. Each processor shares the duty of scheduling and handling inter-process communication. There is no one processor which is dedicated to running system programs. Operating system data such as the status of each virtual process reside in the shared memory and can be accessed by all the processors. Each processor schedules its own activities and updates the shared data.

Each virtual process in the local operating system has its own virtual addressing space and is identified by a unique ID number (VPID). A process communicates with another by sending and receiving logical messages. The receiving process can be in the same level (intra-level mail) or at another level (inter-level mail).

A logical message can be broken up into several smaller physical messages to be transmitted in intervals to lighten the load on the global bus or the gateway controller. In this case, the receiving operating system is responsible for reassembling the incoming messages to recover the original logical message).

At present, SHELL does not break up logical messages for transmission.

A process has an array of virtual mail boxes. Each mail box is identified by a mail box number and can hold more than one logical message. The address of a logical message identifies the level, the VPID and a mail box number of the destination process. When a process is expecting a message, it specifies to the operating system to which mail box is the expected message addressed. It is then put into a suspended state until a message arrives at that mail box. It will then be waken up by the operating system.

The format of a message and to which mail box it should be sent is settled entirely between the sender and receiver process. Usually, when a process expects a reply, it will send a mail box number as part of the message, indicating to which mail box the reply should be sent. By convention, box No. 1 is used by the operating system to pass data to a process when it is initially created.

4.2. System Service Requests

Virtual process 0 (zero) is recognized as the operating system itself. A process can request a service from an operating system by sending a message to virtual process 0. The request can be made to the operating system at the local level or that at a foreign level. In either case, the box number will be decoded as the type of service requested. Currently, only one

type of system service is recognized: creating a new process with a user specified top level procedure.

4.3. The Local Operation System as a Network of Co-Routines

The set of virtual processes in essence forms a network of concurrent coroutines (or modules, as in MODULA [Wirth]). There are two general approaches of allocating co-routines to perform the different functions required by INFOPLEX. One is to allow each transaction to have separate co-routine instances, thus forming a set of parallel, non-intervening chains of activations. The advantage of this arrangement is that the co-routines can be made to fit the characteristic of each individual transaction, and pipeling can be more readily realized.

The other approach is to arrange a co-routine instance to handle a data structure, possibly being shared by several transactions. The co-routine in this case will act as the data handler and the arbiter (as in the MONITORS of Hoare [Hoare]). The advantage of this arrangement is that the co-routines can be made to fit the characteristics of each data structure. In this case, starting a new process is analogous to opening a file, and subsequent requests to the co-routine is like accessing a file which is already opened. After the initial set up, records are retained by the co-routine

so that subsequent processing can be more efficient.

The structure of the operating system allows the writer of application programs to experiment with both approaches.

5.0 THE LOCAL OPERATING SYSTEM EMULATOR

5.1. Overview

The local operating system emulator performs the following functions: 1) scheduling and executing processes, and providing the basic context switching mechanism; 2) inter-process communication; 3) inter-process synchronization; and 4) handling system service requests.

5.2. Process Control

5.2.1. Context Switching

In the current implementation, a process is the execution of a PL/1 procedure. Different processes running the same code corresponds to different instances of the same PL/1 procedure with different arguments. When a process is suspended, everything on the stack -- the AUTOMATIC variables, subroutine call records, the exception handling blocks -- are copied by the local operating system emulator to a storage area, and the stack space is cleared for the activation of the next process. When the process is resumed, the content of the stack at the time of its suspension are restored. The saving and restoring of the stack is completely transparent to the process. (The context switching mechanism will be described in detail in Chapter 6.)

Because STATIC and CONTROLLED variables are not preserved between context switching, the use of these types should be exercised with great caution.

5.2.2. The Virtual Process Status Table

The operating system maintains a Virtual Process Status Table (VPST) to keep track of the execution of each process. Each process is given a status variable, which can take the following values: RUNNING, BLOCKED, RUNNABLE, NASCENT or VOID. RUNNING indicates that a process is currently executing under one of the processors at that level. BLOCKED indicates that the process is suspended and cannot be scheduled because it is still waiting for a message. RUNNABLE indicates that a process is not currently running but can be scheduled in the next cycle. NASCENT is a special form of RUNNABLE and is used when the process has never been run before. (The distinction between NASCENT and RUNNABLE is necessary because starting a new process is handled quite differently from resuming an existing process. See Chapter 6 for details.) VOID indicates that the process has finished and that its slot in the VPST can be recycled.

In addition to the status, the VPST also keeps track for each virtual process: 1) the saved content of the stack; 2) how much time each process has run (VTIME); 3) the waiting box number; and 4) the list of

arrived mail

5.2.3. Scheduling

SAHER is the procedure which schedules and executes the virtual processes. A scheduling cycle is initiated whenever SAHER is dispatched to handle an event. When SAHER finished handling that event, it checks the list of incoming mail for each process to see if there is any mail going to the waiting box. If the waiting box has mail, the status of the process will be reset to RUNNABLE.

After checking the mail, SAHER will select from the set of RUNNABLE or NASCENT processes one process to run. In the current implementation, the scheduling algorithm is to choose the one with the least VTIME.

5.2.4. Process Termination

When a process has finished its task and wants to terminate, it calls the procedure FINISH and then does a RETURN to the calling procedure. FINISH queues an event which is of type FINISH and has the current STIME. The purpose of this event is to synchronize modification to the VPST (see below for details). The status of the process will be reset to VOID and its slot will be recycled.

5.3. Inter-Process Communication

5.3.1. Outgoing Mail: SEND

A process sends a message by calling the procedure SEND. It should specify as arguments to SEND: the level, VPID and mail box number of the destination process. The message is passed as a pointer to a data structure (whose format is of concern only to receiving process). SEND checks to see if the message is addressed to a foreign level. If it is, SEND will queue it as a MSGOUT event to the event queue of the gateway controller of the level. If the message is addressed to the local level, SEND further checks to see if it is a system service request. If it is, the process is first synchronized (see below for detailed explanation), after which the service request will be processed immediately. If the message is a local mail to another process, it will be queued as an LOGMSG event to the local SAH's.

5.3.2. Incoming Mail

An incoming message to a level is first handled by an AAH, which queues it as LOGMSGs to the event queue of the SAH's. If the message is an operating system request, it is handled immediately by the AAH.

SAHER receives the LOGMSG's and puts them in the respective mail boxes. An error is flagged when a

message is sent to non-existing virtual process.

5.3.3. Indefinite Wait

When a process is expecting a message, it calls the procedure WAIT and passes to it as argument the waiting box number. WAIT will queue a WAIT event with the current STIME. (The purpose of the WAIT event is to synchronize the modification to the VPST -- see below for details.) The process is then suspended within the call to WAIT. When the waiting box has mail, WAIT will return.

5.4. Inter-Process Synchronization

Because the local operating system emulator executes in only a simulated multi-processor environment, the actual execution order of a series of operations by different processes may differ from the order which would be carried out if the operating system were in the real world. This can produce results that are inconsistent with the real environment.

To avoid this problem, one must make sure that an operation which is to be executed at $STIME = T$ will be carried out only after all other operations which are to be executed before T have been executed. The operation is then termed synchronized.

Not all operations need synchronization, but any access to data structures shared among processes should

be synchronized. Note that inter-level communication is implicitly synchronized.

5.4.1. SYNC

When a process wants to synchronize an operation, it precedes the operation with a call to the procedure SYNC. SYNC queues an event which is of type SYNC and has the current STIME of the SAH. The SYNC event will be handled after all other events with earlier STIMES have been handled, thus ensuring a strict STIME chronological order. In the mean time, the process is suspended but its status is still RUNNING. When the SYNC event arrives, SAHER changes the status to RUNNABLE and the process is eventually rescheduled, at which point SYNC will return to its caller.

It is important that any access to a data structure shared among processes (e.g. a lock or a semaphore) be synchronized. In the operating system emulator, all the modification to the VPST are synchronised. WAIT and FINISH are special cases of SYNC which change the status of the issuing process to BLOCKED and VOID, respectively.

5.5. System Initialization

At the beginning of simulation, an event type of INIT is placed by the simulator in the event queue to the SAH's of the lowest level. This is analogous to a

'power-on' interrupt to the processors. Its purpose is to trigger the procedure which is responsible for initializing the local operating system.

6.0 PROGRAM STRUCTURE AND EXECUTION LOGIC OF SHELL

6.1. Program Structure of SHELL

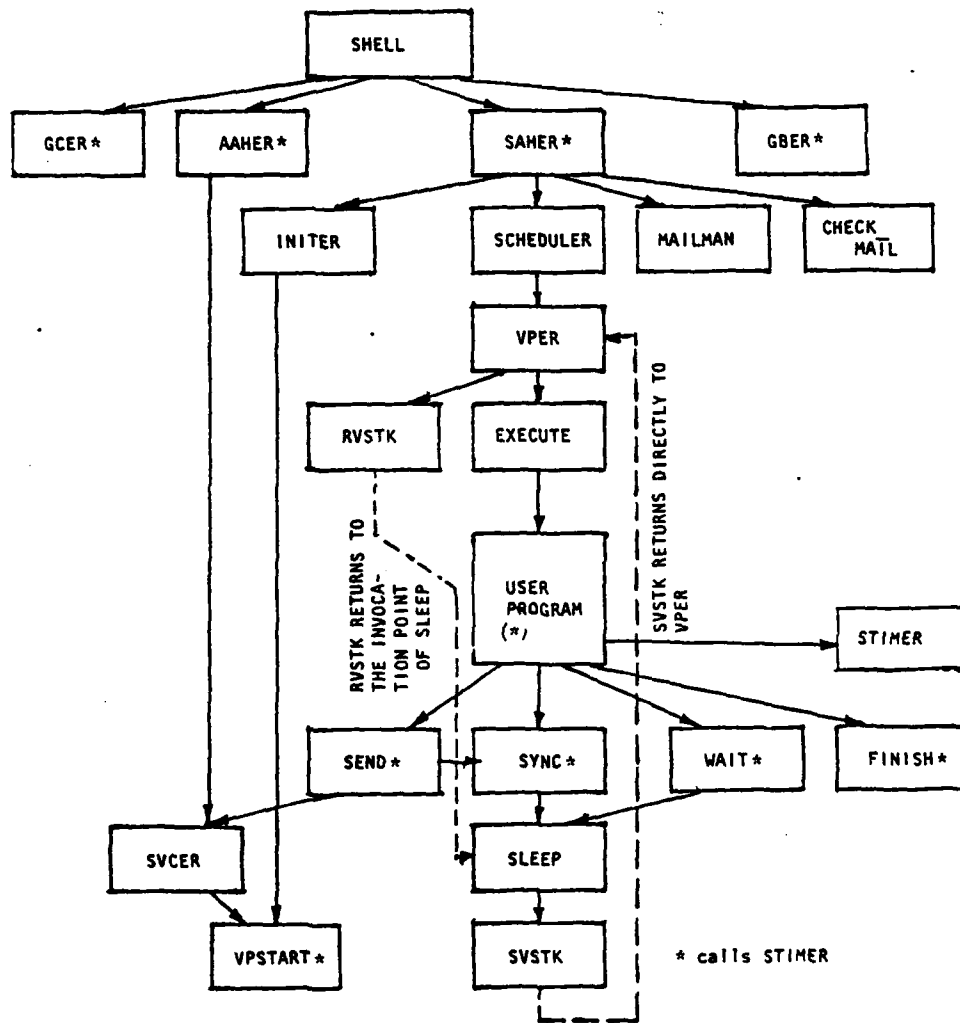
The functional relationship among the major modules of SHELL is depicted in Figure 6.1. SHELL, the event scheduler, dispatches events to one of the four simulation procedures GBER, GCER, AAHER and SAHER. Their functions are described in Chapter 3.

SAHER carries out the majority of the tasks of an operating system. (The other tasks are carried out by AAHER.) It in turn may call its own subroutines. Events of type SYNC, WAIT, and FINISH are handled by SAHER. For events of type LOGMSG, MAILMAN is called to distribute the mail. INITER is called to handle the event type INIT.

After incoming events are handled, SAHER calls CHECK_MAIL to check if any waiting virtual process has received its expected message. It then calls SCHEDULER to select a virtual process to run.

SCHEDULER in turn calls VPER to handle the virtual process. For a NASCENT process, VPER calls EXECUTE which calls the top level procedure of the process. For an old process, VPER calls RTSTK to restore the stack. RTSTK returns directly into the middle of the invocation of the newly restored process.

FIG. 6.1 FUNCTIONAL RELATIONSHIP OF MAJOR MODULES OF SHELL



The user program may call any of the utility procedures STIMER, SEND, WAIT, SYNC and FINISH. STIMER and FINISH both execute and return immediately. SEND may call SYNC if a system service request is involved, otherwise it should return immediately. SYNC and WAIT both call SLEEP, which in turn calls SVSTK to save the stack. SVSTK returns directly to VPER.

6.2. Format of Major Data Structures

6.2.1. LOS

SHELL maintains a giant table which contains the status of all the local operating systems. The format of an entry in the table (a LOS) is shown in Figure 6.2.

LOS.LEVEL is the level of the local operating system. LOS.SRQ.SIZE keeps the current amount of data (in bytes) in the SRQ. LOS.SRQ.MAX keeps the record of the maximum size the SRQ has reached, and is a good reference for finding the optimal size of the SRQ. Similarly, LOS.DBB.SIZE and LOS.DBB.MAX keep track of the current size and maximum size (in number of blocks) of the DBB.

LOS.VPS.TABLE is the virtual process status table of the local operating system. The number of slots in it, MAXVP, is a compile time parameter defined in the macro file CONFIG.

THISLOS (set by AAHER, SAHER)

LEVEL		FIXED BIN
SRQ	SIZE	FIXED BIN
	MAX	FIXED BIN
DBB	SIZE	FIXED BIN
	MAX	FIXED BIN
VPS.TABLE (MAXVP*)		BDSA FIXED BIN (31)
		TDSA FIXED BIN (31)
		SAVESIZE FIXED BIN (31)
		SAFE PTR
		PROCNAME CHAR(7) VAR
		STATUS CHAR(12) VAR
		MAIL PTR
		WAIT BOX FIXED BIN
		MSG PTR
		VPID FIXED BIN
		LEVEL FIXED BIN
		VTIME FIXED BIN (31)

* a compile time parameter, defined in CONFIG

FIG 6.2 FORMAT OF LOS (Local Operating System)

The external POINTER variable THISLOS points to the current LOS. A user program can access the data of the current LOS via THISLOS.

6.2.2. VP

Each entry in the LOS.VPS.TABLE represents a virtual process, and its format is shown in Figure 6.3.

The fields VP.BDSA, VP.TDSA, VP.SAVESIZE and VP.SAFE are used by the context switching mechanism, and their use will be discussed in Section 6.4.

VP.PROCNAME holds the name of the top level procedure.

VP.STATUS holds the status of the virtual process, which must be one of the following: VOID, NASCENT, RUNNABLE, RUNNING, and BLOCKED. (See Chapter 5 for details.)

VP.MAIL points to a chain of mail boxes, each containing a chain of messages (Figure 6.5.2).

VP.WAIT.BOX holds the number of the waiting box. VP.WAIT.MSG points to the expected message when it arrives. It also holds the initial message passed to it by the system when the process is newly created. (See Chapter 7 for details.)

VP.VPID contains the ID of virtual process. VP.LEVEL holds the level number. VP.VTIME holds the cumulative CPU time of the process (in micro-seconds).

The external POINTER variable THISVP points to the current VP entry. A user program can access the data of current virtual process via THISVP.

THISVP (set by VPER)

BDSA	FIXED BIN (31)
TDSA	FIXED BIN (31)
SAVESIZE	FIXED BIN (31)
SAFE	PTR
PROCNAME	CHAR(7) VAR
STATUS	CHAR(12) VAR
MAIL	PTR
WAIT	BOX FIXED BIN
	MSG PTR
VPID	FIXED BIN
LEVEL	FIXED BIN
VTIME	FIXED BIN (31)

FIG 6. 3 FORMAT OF VP (Virtual Process)

6.2.3. SVR

The status of each machine (gateway controller, global bus, AAH, or SAH) in the system is kept in a SVR (Server). SVR's are chained together in the server queues kept by SHELL. (See Chapter 3.) The format of a SVR is shown in Figure 6.4.

SVR.NEXT points to the next server in the same server queue. SVR.STIME holds the base simulation time while SVR.RTIME holds the base real CPU time (in microseconds). SVR.STIMEQ holds the cumulative simulation time of the machine. SVR.RRATE is the ratio of the simulation clock speed to that of the real CPU clock (Chapter 7).

When a machine is dispatched to handle an event, SVR.STIME is set to the start time of the event, and SVR.RTIME is set to value of the real CPU clock at that time. Within the same time slice, the simulation time at any time t is given by:

$$(\text{realtime}(t) - \text{SVR.RTIME}) \times \text{SVR.RRATE} + \text{SVR.STIME}$$

At the end of the time slice, SVR.STIME is updated to show when the machine will be available to handle the next event. Each type of machine has a different update algorithm.

The external POINTER variable THISSVR points to the current SVR. For a user program, this will always points to the SVR of a processor (SAHER).

THISSVR (set by AAHER, SAHER)



NEXT	PTR
STIME	FIXED BIN (31)
STIMEQ	FIXED BIN (31)
RTIME	FIXED BIN (31)
RRATE	FIXED BIN (15,7)
ID	FIXED BIN

FIG. 6.4 FORMAT OF SVR (Server)

6.3. Message Prefixes

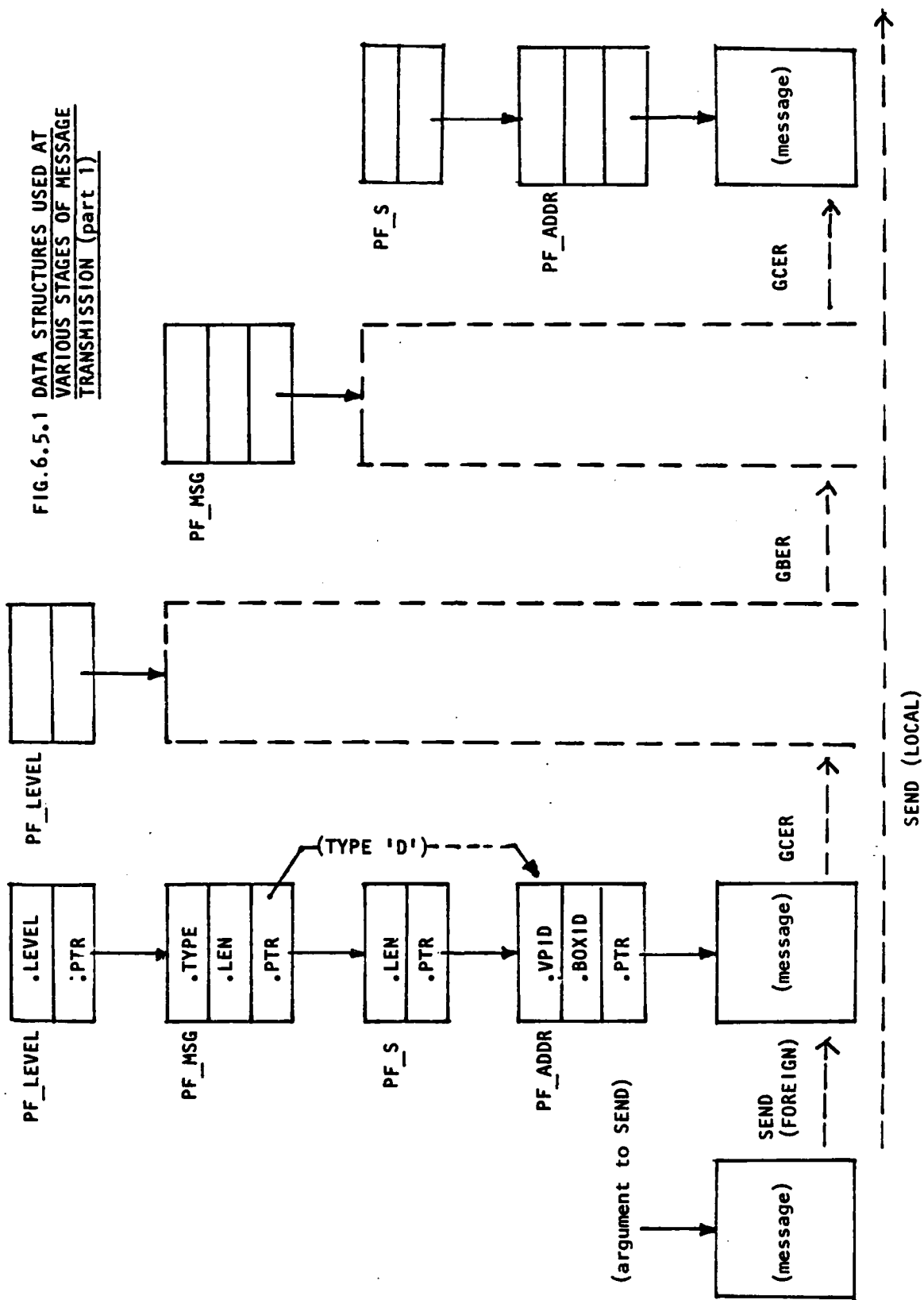
SHELL does not handle the data structure of a message, it only handles the POINTER to that data structure. At the beginning of a message transmission, several different types of data structures (prefixes) are chained to the beginning of the message. These prefixes are later on inspected and stripped off by the procedures in the receiving stages. Typically, a procedure looks only at the foremost prefix and does not care about the format of the data structures that follow.

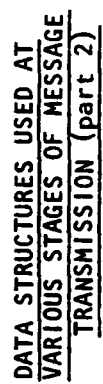
The prefix convention reflects the nature of data transmission in the real world, where descriptor records are usually added to the beginning of a data stream to indicate its format or content.

Figures 6.5.1-2 depicts a pictorial history of a message transmission. The format of the prefixes involved are shown.

6.3.1. PF_ADDR

It contains the destination VPID and box number. It is added on by SEND and stripped off by MAILMAN.





6.3.2. PF_S

It contains the length of the message (including that of PF_ADDR). Only S-type messages have it since D-type messages are fixed in length. It is added on by SEND. AAHER uses it to update the size of the SRQ, after which it is stripped off.

6.3.3. PF_MSG

It contains the type (either 'S' or 'D') of the message as well as its length (including that of PF_S and PF_ADDR). It is added on by SEND and stripped off by GCER.

6.3.4. PF_LEVEL

It contains the destination level of the message. It is added on by SEND. It is inspected by both GCER and GBER. It is stripped off by GBER.

6.4. The Execution Logic of the Context Switching Mechanism

The context switching mechanism works only in the IBM OS PL/1 execution environment. (For details of the environment, see [IBM].)

6.4.1. The Stack Space

Every time a PL/1 procedure is invoked, it allocates a temporary storage area on the stack. This area is headed by the DSA (Dynamic Storage Area) of the procedure, followed by the AUTOMATIC variables and other

temporary data. The DSA holds the NAB (Next Available Byte), a pointer to the end of the temporary storage area. If the procedure calls another procedure, the DSA will also hold the return address for the called procedure.

The stack space in the PL/1 execution environment is initially contiguous but may later become segmented. Each segment of the stack space is identified by a segment number which is also contained in the DSA. The context switching mechanism requires a contiguous stack space. Hence, every time before saving or restoring the stack, the segment number is checked to make sure that the stack space is still contiguous.

The end of the stack space is marked by the EOS (End of Segment) slot in the TCA (Task Communication Area) of the PL/1 execution environment. EOS changes as BASED or CONTROLLED storages are ALLOCATE'd and FREE'd.

The format of the DSA is shown in Figure 6.6, and the format of the TCA is shown in Figure 6.7. For further information about the internal data format of the PL/1 execution environment, see [IBM].

6.4.1.1. GET4

GET4 is an assembly language subroutine. It obtains the following information: the address of the DSA (of the calling PL/1 procedure), the NAB, the segment number and the EOS.

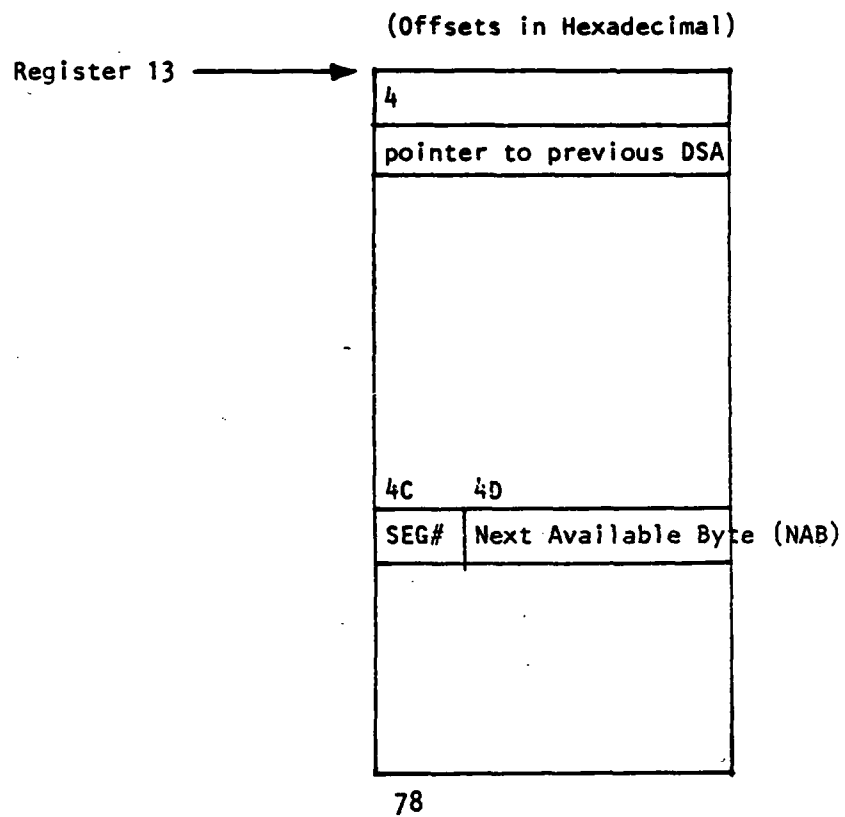


FIG. 6.6 FORMAT OF THE DYNAMIC STORAGE AREA (DSA)
 (for further details of the format, see [IMB])

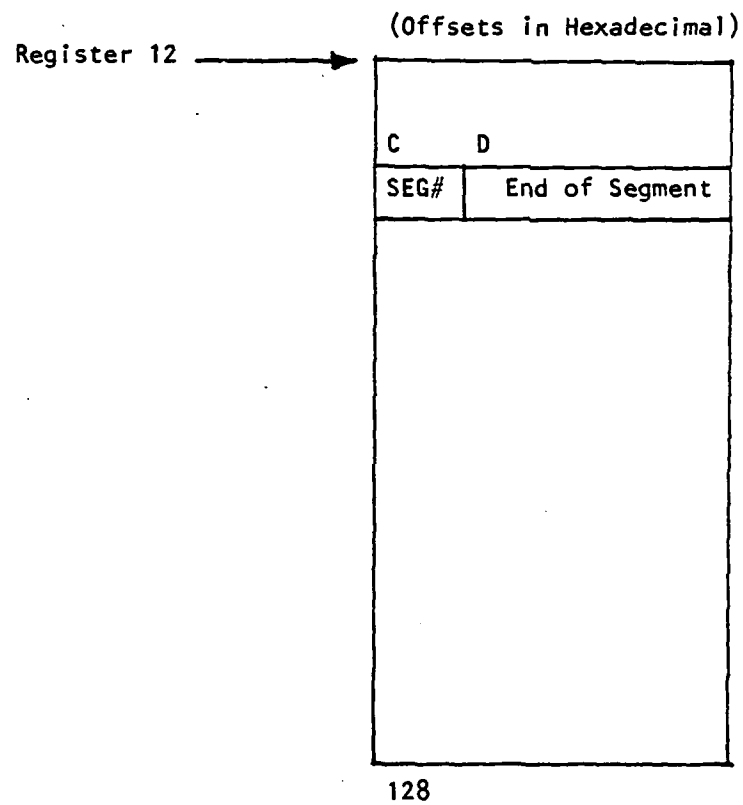


FIG. 6.7 FORMAT OF THE TASK COMMUNICATION AREA (TCA)
(for further details of the format, see [IMB])

6.4.2. Saving a Process

Before invoking a process, VPER first gets (via GET4) the address of its own DSA and puts it in the BDSA (Bottom DSA) field of the current VP. It then calls EXECUTE, which in turn calls the user program.

Control eventually reaches SLEEP. SLEEP gets (via GET4) the address of its own DSA and puts it in the TDSA (Top DSA) field of the current VP. The amount of data to be saved (between the BDSA and the NAB of the TDSA) is calculated and put in the SAVESIZE field of the VP.

SLEEP then allocates a number of save blocks, which are chunks of BASED storage chained together. VP.SAFE points to the first save block.

SLEEP then calls SVSTK. SVSTK copies the content of the stack (from the BDSA to the NAB of the TDSA) to the save blocks. SVSTK returns directly the the BDSA, as if EXECUTE were returning to VPER.

6.4.3. Restoring a Process

Before restoring a process, VPER checks (via GET4) to see if there is enough space left in the stack space for restoration. If not, an error will be signaled.

VPER then calls RTSTK, which copies the data from the save blocks to the stack space. Every datum is put back at the exact same address. Note that the DSA of VPER is 'painted over' by RTSTK, so that the state of

VPER is restored to the state when it called EXECUTE. RTSTK returns to the TDSA, as if SVSTK were returning to SLEEP. Control will eventually return to the user program. The saving and restoring is transparent to the user program; and as far as the user program is concerned, it called a utility procedure (SYNC or WAIT) and it has just returned.

The state of the stack space at various stages is depicted in Figures 6.8.1-4.

FIG. 6:8.1 STATE OF THE STACK DURING CONTEXT SWITCHING (part 1)

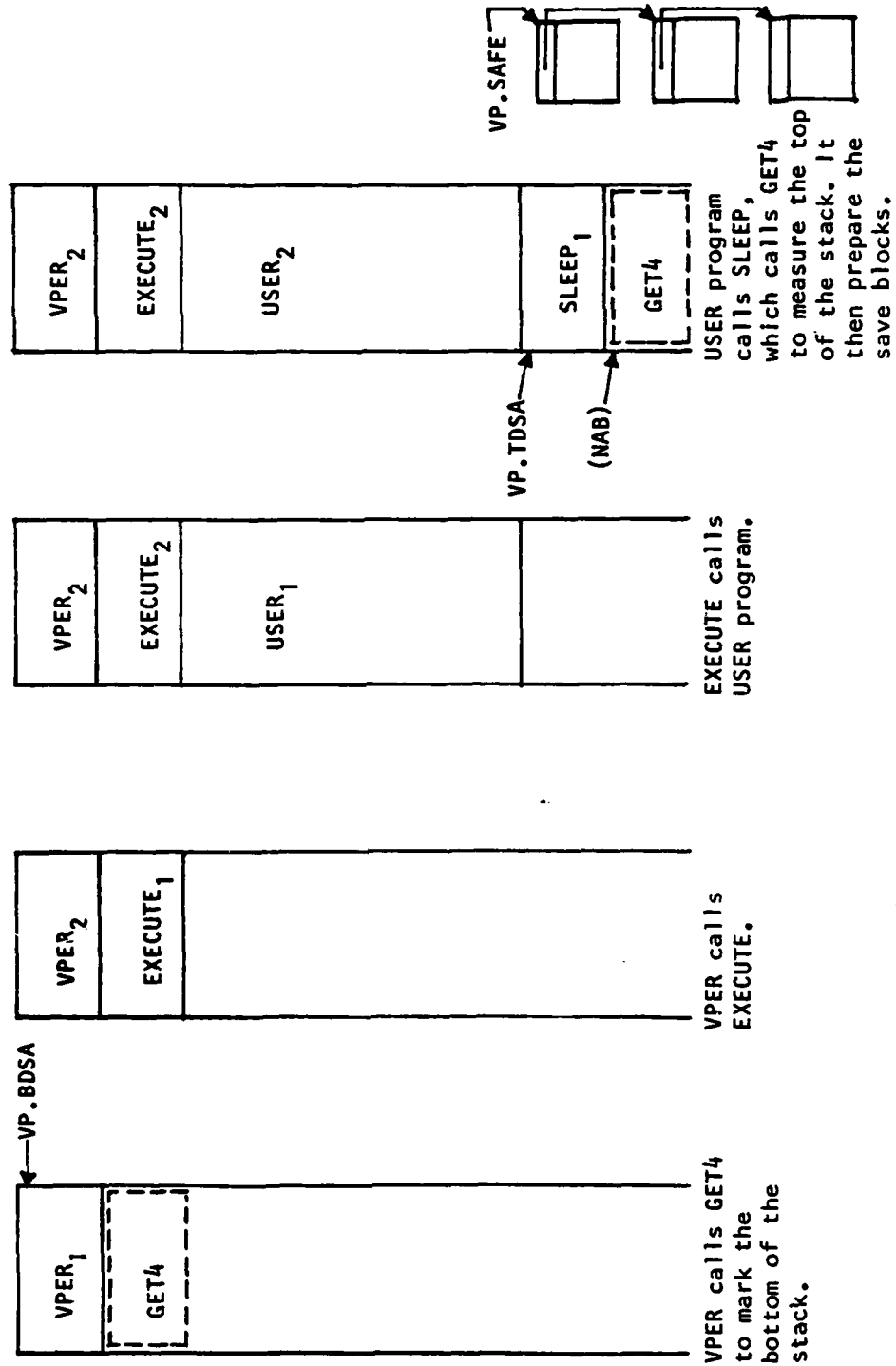


FIG. 6.8.2 STATE OF THE STACK DURING CONTEXT SWITCHING (part 2)

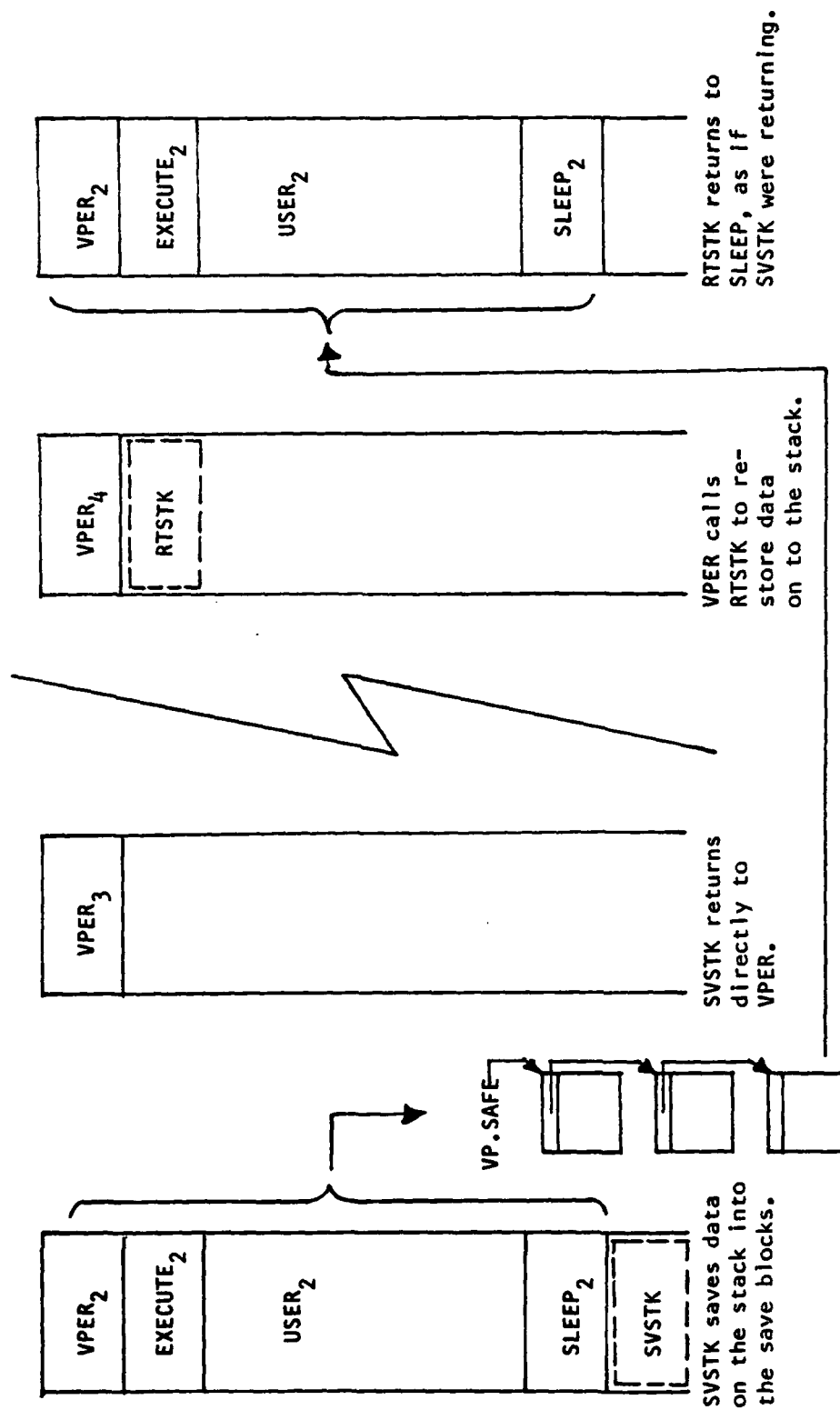


FIG. 6.8.3 STATE OF THE STACK DURING CONTEXT SWITCHING (part 3)

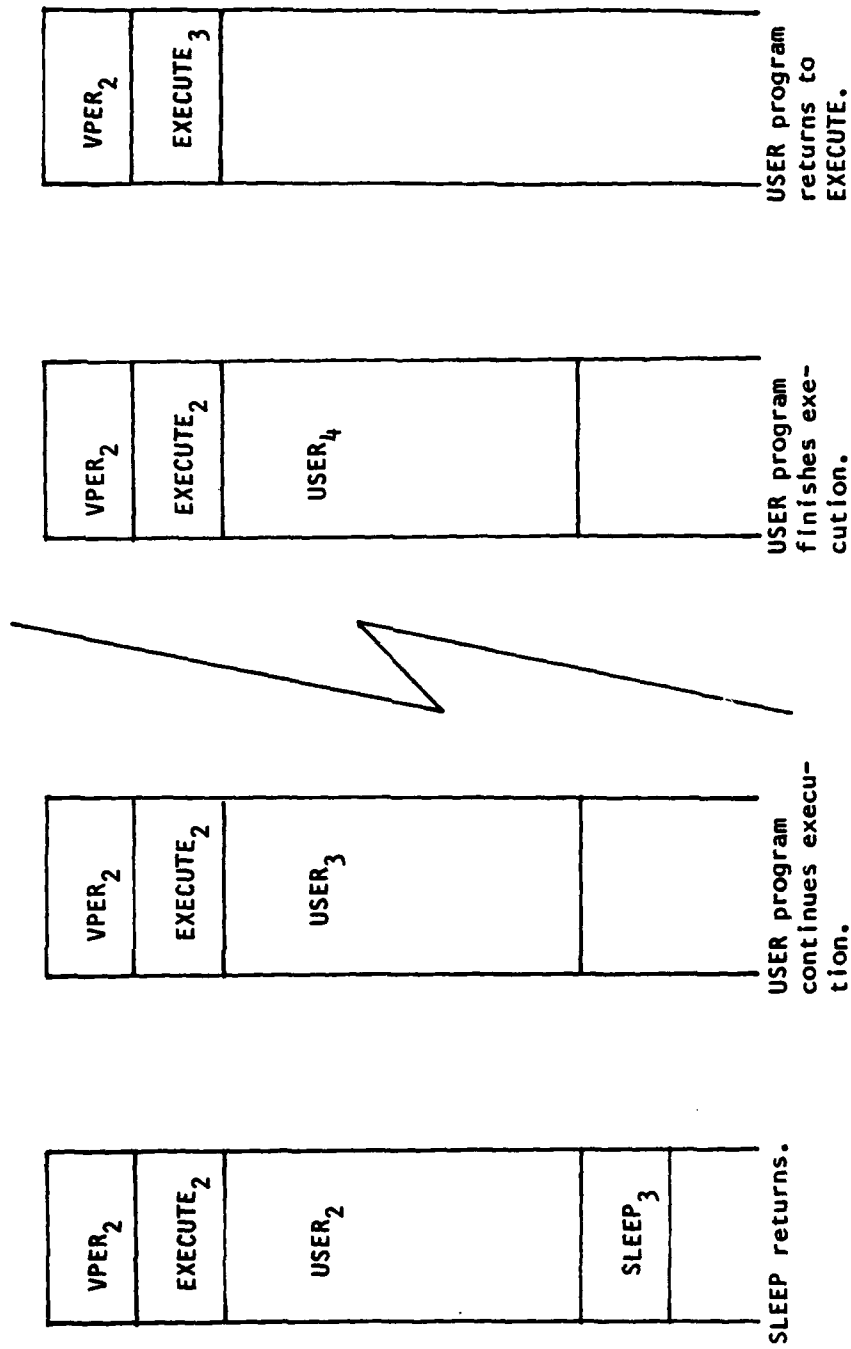
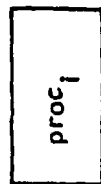


FIG. 6.8.4 STATE OF THE STACK DURING CONTEXT SWITCHING (part 4)



EXECUTE returns.

Legend:



Activation Record
of a procedure on the
stack (DSA, AUTOMATIC
variables, et al).
The subscripts dis-
guish the different states
of the procedure.



Assembly language
procedure which does
not use the stack.

7.0 INTERFACE WITH APPLICATION PROGRAMS--A USER'S GUIDE TO SHELL

To adopt a PL/1 program to run under SHELL, it must first be re-organized to fit the multi-process (co-routine) execution environment of SHELL. In specific, it should be functionally decomposed into a set of top level procedures, each top level procedure being an independent unit performing a specific function. There should be no sharing of data or subroutines among top level procedures, but within each top level procedure, there can be internal subroutines and data can be shared among the internal subroutines.

Top level procedures communicate with one another only by sending and receiving messages, using the utility procedures SEND and WAIT. (Section 7.4)

A virtual process in SHELL is an activation of a top level procedure. Several processes may share the same top level procedure. A virtual process is created by SHELL upon request from other processes. A process can request a new process to be created at its own level or at another level. It does so by sending a message to the operation system of the desired level and specifying which top level procedure is to be used. (Section 7.5)

At the beginning of simulation, SHELL creates several processes with the user-supplied procedure TERM. (Section 7.3) TERM should be programmed to handle the initial terminal I/O and start the action sequence.

One starts the simulation by loading and running SHELL.

7.1. To Register a Top Level Procedure with SHELL

EXECUTE is the (only) procedure in SHELL that reference the user-supplied top level procedures. It is a very simple program which consists almost entirely of clauses of the form: (Figure 7.1)

```
IF (PROCNAM='X') THEN DO; CALL X; RETURN; END;
```

To add a top level procedure to SHELL, one should modify EXECUTE by adding a clause of the above form (with the appropriate name substituting the 'X') to the body of EXECUTE. EXECUTE should then be recompiled.

Table 7.1 shows a list of EXTERNAL variable and entry names used by SHELL. A user-supplied top level procedure should not have a name identical to the names in the list.

7.2. Parameters

7.2.1. Compile Time Parameters

There are only two compile time parameters: MAXLEVEL and MAXQ. MAXLEVEL is the one less the number of levels in the functional hierarchy; and MAXQ is always MAXLEVEL x 3. They are both defined in the macro file CONFIG. To set these two parameters, modify the macro file CONFIG and recompile SHELL.

SOURCE LISTING

NUMBER LEV NT

```

10 0 EXECUTE: PROC (PROCNAME);
30 1 0 DCL PROCNAME CHAR(*) VAR;
40 1 0 DCL TERM ENTRY;
60 1 0 IF PROCNAME = 'TERM' THEN DO; CALL TERM; RETURN; END;
90 1 0 PUT SKIP LIST (',' ERROR: UNKNOWN PROCNAME. ',PROCNAME,
100 1 0 STOP;
120 1 0 END EXECUTE;

```

-54-

FIG 7.1 LISTING OF PROCEDURE EXECUTE

Table 7.1 LIST OF EXTERNAL IDENTIFIERS

ENTRIES:

AAHER
ADDQ
CONS
DEBUGR
EXECUTE
FINISH
GBER
GCER
GET4
QEVENT
RTIMER
RTSTK
SAHER
SEND
SHELL
SLEEP
STIMER
SVCER
STSTK
SYNC
TERM
VPER
VPSTART
WAIT

DATA:

DEBUG
PARAMS
THISLOS
THISSVR
THISVP

PLIXOPT

7.2.2. Run Time Parameters

At the beginning of each simulation, SHELL asks for the values of a set of run time simulation parameters.

7.2.2.1. Number of Processors

Each level has a different number of processors.

7.2.2.2. RRATE

RRATE is the ratio between the simulation time clock rate and the real (CPU) time clock rate. It is declared as a FIXED BIN (15,7) variable. A RRATE greater than one means that the simulation time clock rate (of the processors) of that level is faster than the real CPU time rate (in 370 VM/CP), which means that the processor at that level are running slower than the real time clock. Each level has a different RRATE.

7.2.2.3. THRU_PUT

THRU_PUT is the overall average throughput rate at the global bus. The unit is in micro-seconds per byte. It is declared as a FIXED BIN (31,7).

7.2.2.4. DELAY_GB_GC, DELAY_GC_GB

DELAY_GB_GC is the average delay (micro-seconds) between the arrival of the first byte at the global bus and the arrival of the first byte at the destination gateway controller. DELAY_GC_GB is the counter part for outgoing messages.

7.2.2.5. TERMINALS

This is the number of 'virtual terminals' attached to the system. During initialization, SHELL starts that many number of virtual processes in the lowest level operating system. (See 7.3 for details.)

7.3. Initialization

At the beginning of simulation, the simulator puts into the lowest level operating system a number (= TERMINALS) of virtual processes with TERM as the top level procedures. TERM should be supplied by the user and should handle the initial user I/O and initialize the transactions.

7.4. Utility Procedures

There are five utility procedures which an application program can call directly: STIMER, SEND, SYNC, WAIT and FINISH. The macro file USERS contains the declarations of each of these entries and can be %INCLUDE'd to the user program.

7.4.1. STIMER

STIMER takes no argument and returns the current simulation time of current processor. It is controlled by the RRATE of the current level. The format of returned value is FIXED BIN (31).

7.4.2. SEND

The calling sequence of SEND is:

CALL SEND (level,vpid,boxid,type,len,ptr)

where level, vpid and boxid are those of the destination process. Type can be either 'D' or 'S', indicating which type of message should be sent. Len is the length, in bytes, of the message to be sent, and ptr is a POINTER pointing to the message. SHELL itself does not decode the message (except in the cases of system service requests, see Section 7.5); so the message can be of any data type as long as the receiving process knows how to decode it. Messages cannot be AUTOMATIC as the stack is modified when context switching occurs.
(Ref Section 7.7)

7.4.3. SYNC

SYNC takes no argument. The calling process is suspended until it has become the earliest event in the system. (See Section 5.4 for details.) SYNC should be used before any access to a common shared data structure such as a lock or semaphore.

7.4.4. WAIT

WAIT takes a box number as argument. The calling process is suspended until a message arrives at the indicated mail box. If there is already a message in the mailbox when WAIT is called, it will have the same ef-

fect as SYNC.

7.4.5. FINISH

FINISH takes no argument and signals the end of the calling process. Note that FINISH does not transfer control out of the calling procedure, and a RETURN or a GOTO to the last END statement should follow the call to FINISH.

7.5. System Service Requests

Virtual Process 0 of each level is recognized as the operating system itself. A message sent to virtual process 0 will be decoded as a system service request. The destination box number in the case will be decoded as the type of system service desired.

Currently only one system service (box number = 1) is implemented. It is the starting of a new virtual process.

7.5.1. Start a New Virtual Process

The format of the message required by this system service is shown in Figure 7.2. A prefix (PF_SVC) is added to the beginning of the message which is going to be passed to the newly started virtual process. The prefix should contain the name of the top level procedure. After the operating system starts the process, the prefix is stripped off and the rest is put into the waiting box of the new process.

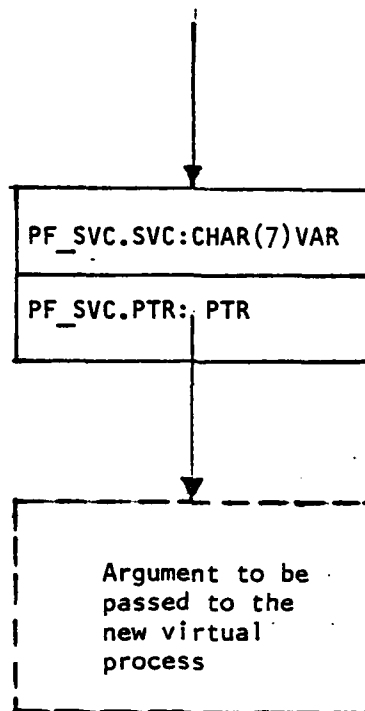


FIG.7.2 FORMAT OF ARGUMENT FOR
SYSTEM SERVICE REQUEST 1
(STARTING A NEW VIRTUAL
PROCESS)

7.6. External Variables

The EXTERNAL variables THISVP, THISLOS and THISSVR are POINTER variables pointing respectively to the current Virtual Process, Local Operating System and Server (Processor). The format of data structures VP, LOS and SVR are listed in the Chapter 6. The macro files VPX, LOSX and SVRX contain all the necessary declarations and can be %INCLUDE'd by the application programs.

7.6.1. VP: Information about the current Virtual Process

Several sub-fields in VP are of special interest to an application program:

7.6.1.1. VP.WAIT.MSG

When a process wakes up from a WAIT, this slot contains the pointer to the message arrived at the expected mail box. When a process is newly created, this slot contains the argument passed to it by the system or the parent process.

7.6.1.2. VP.LEVEL

This is the level number of the embedded local operating system. It is declared as a FIXED BIN.

7.6.1.3. VP.VPID

This is the Virtual Process ID of the current process. It is declared as a FIXED BIN.

7.6.1.4. VP.VTIME

This contains the cumulative simulation time (micro-seconds) of each process, in the format of FIXED BIN (31).

7.7. Caveats & Restrictions

7.7.1. Local Variables

The AUTOMATIC variables in a top level procedure (and everything else that is stored on the stack) are associated with the process; while the STATIC and CONTROLLED variables are associated with the procedure. Hence, if a top level procedure is expected to be shared by more than one processes, it must either use only AUTOMATIC variables or keep track of its STATIC and CONTROLLED variables very carefully.

7.7.2. Messges

SHELL keeps track of messages as POINTERS and does not care about the actual data format or the storage class of the messages. However, because the stack space is multiplexed among processes, the AUTOMATC variables of a sending process will disappear from the stack space when the receiving process is invoked. Hence, messages

should not be contained inside AUTOMATIC data structures.

The preferred way is to use BASED data structures for messages. The sending process will ALLOCATE them and the receiving process will FREE them after processing them.

7.7.3. The REPORT option

Because the PL/1 run time option REPORT interferes with the operation of the context switching mechanism, it should never be used with SHELL.

7.8. STATS

STATS is a file containing certain performance statistics of SHELL. At present, it contains the following: 1) the run time parameters; 2) the average delay time of the gateway controllers; 3) the time distribution and duration of the messages arriving at the global bus; and 4) the cumulative active time of each machine.

7.9. A Sample Program

A test program is presented here as an example. It is called TERM so that it is activated by SHELL when simulation begins. A listing of TERM is in Figure 7.3 as well as in the Appendix.

When TERM comes up, it will identify itself by printing the current level, VPID, waiting box number, the ID of the processor (SVR), and the current STIME.

SOURCE LISTING

NUMBER LEV NT

```

10      0  TERM: PROC;
                                     TER00010
                                     TER00020
                                     XINCLUDE USERS;*****TER00030
                                     XINCLUDE STIMER;*****USE00010
200010  1  0  DCL STIMER ENTRY RETURNS (FIXED BIN (31));
                                     STI00010
                                     *****USE00010
300020  1  0  DCL SEND ENTRY (FIXED BIN, FIXED BIN, FIXED BIN, CHAR(*) VAR,
                                     USE00020
                                     FIXED BIN, PTR);
                                     USE00030
300040  1  0  DCL WAIT ENTRY (FIXED BIN);
                                     USE00040
300050  1  0  DCL SYNC ENTRY;
                                     USE00050
300060  1  0  DCL FINISH ENTRY;
                                     USE00060
                                     USE00070
                                     TER00030
400040  1  0  DCL (NULL,LENGTH) BUILTIN;
                                     TER00040
                                     TER00050
                                     XINCLUDE VPX;*****TER00060
500010  1  0  DCL THISVP PTR EXTERNAL STATIC; /* -> CURRENT VP, SET BY VPER */VPX00010
                                     VPX00020
500030  1  0  DCL 1 VP BASED (THISVP), XINCLUDE VP;*****VPX00030
                                     /* ADDR (BOTTOM DSA) */VP 00010
                                     80 BDSA FIXED BIN (31), /* ADDR (TOP DSA) */VP 00020
                                     80 TDSA FIXED BIN (31), /* SIZE OF AREA TO BE SAVED */VP 00030
                                     80 SAVESIZE FIXED BIN (31), /* -> 1ST SAVBLK */VP 00040
                                     80 SAFE PTR, /* NAME OF TOP LEVEL PROC */VP 00050
                                     80 PROCNAME CHAR (7) VAR, VP 00060
                                     80 STATUS CHAR (12) VAR, /* CHAIN OF INCOMING MAIL */VP 00070
                                     80 MAIL PTR, VP 00080
                                     80 WAIT, /* BOX AWAITING MAIL */VP 00090
                                     81 BOX FIXED BIN, /* MSG IN WAIT.BOX */VP 00100
                                     81 MSG PTR, /* INDEX IN THE VPST */VP 00110
                                     80 VPID FIXED BIN, /* LEVEL */VP 00120
                                     80 LEVEL FIXED BIN, VP 00130
                                     80 VTIME FIXED BIN (31)
                                     *****
                                     VPX00030
                                     VPX00040
                                     TER00060
                                     XINCLUDE SVRX;*****TER00070
800010  1  0  DCL THISSVR PTR EXTERNAL STATIC; /* -> CURRENT SVR, SET BY SAHER */SVR00010
                                     SVR00020
800030  1  0  DCL 1 SVR BASED (THISSVR), XINCLUDE SVR;*****SVR00030
                                     70 NEXT PTR, SVR00010
                                     70 STIME FIXED BIN (31), SVR00020
                                     70 STIMEQ FIXED BIN (31), /* ACCUMULATED STIME */SVR00030
                                     70 RTIME FIXED BIN (31), SVR00040
                                     70 RRATE FIXED BIN (15,7), /* REAL CPU TICK RATE */SVR00050
                                     70 ID FIXED BIN SVR00060
                                     *****
                                     SVR00030

```

FIG, 7.3.1 LISTING OF PROCEDURE TERM (part 1)

NUMBER LEV NT

```

*****
%INCLUDE PFSVC;*****
1100010 1 0 DCL PT_SVC PTR;
1100030 1 0 DCL 1 PF_SVC BASED (PT_SVC),
          2 SVC CHAR (7) VAR,
          2 PTR PTR;

*****
%INCLUDE MSG;*****
1200010 1 0 DCL MSGLEN FIXED BIN;
1200020 1 0 DCL MSGPT PTR;
1200030 1 0 DCL 1 MSG BASED (MSGPT),
          2 LEN FIXED BIN,
          2 STR CHAR (MSGLEN REFER (LEN));

*****
1200100 1 0 DCL (LEVEL,VPID,BOXID) FIXED BIN;
1200110 1 0 DCL MESSAGE CHAR(80) VAR;
1200120 1 0 DCL COMMAND CHAR(12);
1200130 1 0 DCL TYPE CHAR(8) VAR;
1200150 1 0 LOOP:
          PUT SKIP EDIT (' TERM: LEVEL = ',VP.LEVEL,' VP = ',VP.VPID,
          ' WAIT.BOX = ',VP.WAIT.BOX,' SVR = ',SVR.ID,' STIME = ',
          SVR.STIME) (A,F(2),A,F(2),A,F(2),A,F(2),A,F(2),A,F(11));
1200190 1 0 MSGPT = VP.WAIT.MSG;
1200200 1 0 IF (MSGPT = NULL) THEN DO;
1200210 1 1 PUT SKIP EDIT (' MSG: ',MSG.STR) (A,A(MSG.LEN));
1200220 1 1 FREE MSG;
1200230 1 1 VP.WAIT.MSG = NULL;
1200240 1 1 END;

1200260 1 0 WORK:
          PUT SKIP;
1200280 1 0 DISPLAY (' COMMAND?') REPLY (COMMAND);

1200300 1 0 IF (COMMAND = 'BUILD') THEN DO;
1200310 1 1 PUT SKIP LIST (' LEVEL> ');
1200320 1 1 GET LIST (LEVEL);
1200330 1 1 MSGLEN = 7;
1200340 1 1 ALLOCATE MSG;
1200350 1 1 MSG.STR = 'NEW VPI';
1200360 1 1 ALLOCATE PF_SVC;
1200370 1 1 PF_SVC.SVC = 'TERM';
1200380 1 1 PF_SVC.PTR = MSGPT;
1200390 1 1 CALL SEND (LEVEL,0,1,'S',15,PT_SVC);
1200400 1 1 GOTO WORK;
SVR00040
TER00070
TER00080
PFS00010
PFS00020
PFS00030
PFS00040
PFS00050
PFS00060
TER00080
TER00090
MSG00010
MSG00020
MSG00030
MSG00040
MSG00050
MSG00060
TER00090
TER00100
TER00110
TER00120
TER00130
TER00140
TER00150
TER00160
TER00170
TER00180
TER00190
TER00200
TER00210
TER00220
TER00230
TER00240
TER00250
TER00260
TER00270
TER00280
TER00290
TER00300
TER00310
TER00320
TER00330
TER00340
TER00350
TER00360
TER00370
TER00380
TER00390
TER00400

```

FIG. 7.3.2 LISTING OF PROCEDURE TERM (part 2)

NUMBER LEV NT

1200410	1	1	END;	TER00410
1200430	1	0	ELSE IF (COMMAND = 'SEND') THEN DO;	TER00420
1200440	1	1	PUT SKIP LIST (' LEVEL,VPID,BOXID,TYPE,MESSAGE> ');	TER00430
1200450	1	1	GET LIST (LEVEL,VPID,BOXID,TYPE,MESSAGE);	TER00440
1200460	1	1	MSGLEN = LENGTH (MESSAGE);	TER00450
1200470	1	1	ALLOCATE MSG;	TER00460
1200480	1	1	MSG.STR = MESSAGE;	TER00470
1200490	1	1	CALL SEND (LEVEL,VPID,BOXID,TYPE,MSGLEN,MSGPT);	TER00480
1200500	1	1	GOTO WORK;	TER00490
1200510	1	1	END;	TER00500
1200530	1	0	ELSE IF (COMMAND = 'WAIT') THEN DO;	TER00510
1200540	1	1	PUT SKIP LIST (' BOX> ');	TER00520
1200550	1	1	GET LIST (BOXID);	TER00530
1200560	1	1	PUT SKIP LIST (' WAITING');	TER00540
1200570	1	1	CALL WAIT (BOXID);	TER00550
1200580	1	1	GOTO LOOP;	TER00560
1200590	1	1	END;	TER00570
1200610	1	0	ELSE IF (COMMAND = 'SYNC') THEN DO;	TER00580
1200620	1	1	PUT SKIP LIST (' SYNCING');	TER00590
1200630	1	1	CALL SYNC;	TER00600
1200640	1	1	GOTO LOOP;	TER00610
1200650	1	1	END;	TER00620
1200670	1	0	ELSE IF (COMMAND = 'FINISH') THEN DO;	TER00630
1200680	1	1	PUT SKIP LIST (' FINISHING');	TER00640
1200690	1	1	CALL FINISH;	TER00650
1200700	1	1	RETURN;	TER00660
1200710	1	1	END;	TER00670
1200730	1	0	ELSE DO; /* UNKNOWN COMMAND */	TER00680
1200740	1	1	PUT LIST (' ??');	TER00690
1200750	1	1	GOTO WORK;	TER00700
1200760	1	1	END;	TER00710
1200780	1	0	END TERM;	TER00720
				TER00730
				TER00740
				TER00750
				TER00760
				TER00770
				TER00780

FIG. 7.3.3 LISTING OF PROCEDURE TERM (part 3)

If there is a message in the waiting box, it will print the message as well. TERM then asks for commands from the user and executes them. It recognizes five commands: BUILD, SEND, WAIT, SYNC and FINISH. After the command is performed, (it might take quite sometime), it will identify itself and ask for more work.

7.9.1. Command BUILD

BUILD creates a new process in the system with TERM as the top level procedure. It takes an argument, the level on which the new process is to be created. Note that at the beginning of simulation, there are already N (N = TERMINALS, a run time parameter) instances of TERM at level 0, being put there by the system initialization procedure. Repeated uses of BUILD can really proliferate the number of instances of TERM.

7.9.2. Command SEND

Command SEND takes five arguments: the destination level, VPID, and box number, the type of message (either 'S' or 'D') and the message itself (which must be within quotes). SEND will simply call (the utility procedure) SEND to send the message.

7.9.3. Command WAIT

Command WAIT takes one argument, the waiting box number. It will prompt 'WAITING..', and calls (the utility procedure) WAIT.

7.9.4. Commands SYNC and FINISH

Both these commands take no argument and are executed promptly by calling the corresponding utility procedure.

The network of instances of TERM is like a network of terminals or telex machines. The user can tailor the architecture of this network by building the desired number of processes at each level. Since the user can specify the action of each node in the network, any arbitrary traffic pattern can be generated and tested.

7.10. A Sample Simulation Session

A very simple simulation session is scripted in Figures 7.4.1-3. The configuration in this simulation consists of two levels, each with two processors.

Simulation starts with three processes at level 0: VP0.1, VP0.2, and VP0.3. VP0.1 builds a process at level 1 (VP1.1), and then goes into waiting for a message to come to box 1. Similarly, VP0.2 builds VP1.2 then waits for box 2; and VP0.3 builds VP1.3 and waits for box 3.

```

R: T=0.01/0.01 01:08:47
shell
LOAD SHELL ( CLEAR NODUP NOMAP )
FILEDEF STATS DISK STATS LISTING A1 ( BLOCK 800 )
R: T=0.37/0.65 01:08:59
start
EXECUTION BEGINS...

```

```

LEVEL 0:
NO. OF PROCESSORS, RRATE>
:
2,1

LEVEL 1:
NO. OF PROCESSORS, RRATE>
:
2,1

THRU_RATE,GB_GC,GC_GB,TERMINALS>
:
1,0,0,3

```

START SIMULATION...

```

TERM: LEVEL = 0 VP = 1 WAIT.BOX = 0 SVR = 2 STIME =      555
MSG: SET UP BY INITER

```

```

COMMAND?
build
LEVEL>
:
1

```

```

COMMAND?
wait
BOX>
:
1

```

WAITING

```

TERM: LEVEL = 0 VP = 2 WAIT.BOX = 0 SVR = 1 STIME =      1062
MSG: SET UP BY INITER

```

```

COMMAND?
build
LEVEL>
:
1

```

```

COMMAND?
wait
BOX>
:

```

FIG. 7.4 .1 SAMPLE SIMULATION SESSION (part 1)

2

WAITING

TERM: LEVEL = 1 VP = 1 WAIT.BOX = 0 SVR = 2 STIME = 58299
MSG: NEW VP!

COMMAND?

wait
BOX>
:
1

WAITING

TERM: LEVEL = 1 VP = 2 WAIT.BOX = 0 SVR = 1 STIME = 70248
MSG: NEW VP!

COMMAND?

wait
BOX>
:
2

WAITING

TERM: LEVEL = 0 VP = 3 WAIT.BOX = 0 SVR = 1 STIME = 95062
MSG: SET UP BY INITER

COMMAND?

build
LEVEL>
:
1

COMMAND?

wait
BOX>
:
3

WAITING

TERM: LEVEL = 1 VP = 3 WAIT.BOX = 0 SVR = 2 STIME = 151919
MSG: NEW VP!

COMMAND?

send
LEVEL,VPID,BOXID,TYPE,MESSAGE>
:
0,1,1,'S','from 1.3'

COMMAND?

finish

FINISHING

TERM: LEVEL = 0 VP = 1 WAIT.BOX = 1 SVR = 2 STIME = 209840
MSG: from 1.3

COMMAND?

FIG. 7.4.2 SAMPLE SIMULATION SESSION (part 2)

```

send
  LEVEL,VPID,BOXID,TYPE,MESSAGE>
:
1,1,1,'D','from 0.1'

COMMAND?
finish

FINISHING

```

```

TERM: LEVEL = 1 VP = 1 WAIT.BOX = 1 SVR = 1 STIME = 267747
MSG: from 0.1

```

```

COMMAND?
send
  LEVEL,VPID,BOXID,TYPE,MESSAGE>
:
1,2,2,'S','from 1.1'

COMMAND?
finish

FINISHING

```

```

TERM: LEVEL = 1 VP = 2 WAIT.BOX = 2 SVR = 2 STIME = 324723
MSG: from 1.1

```

```

COMMAND?
send
  LEVEL,VPID,BOXID,TYPE,MESSAGE>
:
0,2,2,'D','from 1.2'

COMMAND?
finish

FINISHING

```

```

TERM: LEVEL = 0 VP = 2 WAIT.BOX = 2 SVR = 1 STIME = 384779
MSG: from 1.2

```

```

COMMAND?
send
  LEVEL,VPID,BOXID,TYPE,MESSAGE>
:
0,3,3,'S','from 0.2'

COMMAND?
finish

FINISHING

```

```

TERM: LEVEL = 0 VP = 3 WAIT.BOX = 3 SVR = 2 STIME = 441675
MSG: from 0.2

```

```

COMMAND?
finish

FINISHING

```

```

R: T=0.64/1.76 01:11:57

```

FIG. 7.4.3 SAMPLE SIMULATION SESSION (part 3)

At level 1, VP1.1 waits for box 1 and VP1.2 waits for box 2. Then VP1.3 sends a message to VP0.1. After sending the message, VP1.3 terminates itself.

When VP0.1 wakes up, it sends a message to VP1.1 and then terminates. Similarly, VP1.1 sends to VP1.2; VP1.2 sends to VP0.2; and VP0.2 sends to VP0.3. When VP0.3 finally wakes up, it is the last process left in the system, and when it terminates, the simulation session ends as well.

In Figures 7.4.1-3, the user input is in lower case while the output of the simulator is in upper case. Figure 7.5 diagrams the chronology of the events in the simulation. The STATS file of the simulation is listed in Figure 7.6.

```

*** 81/05/19 01:09
*** PARAMETERS:
LEVEL 0: NP= 2 RRATE= 1.000:
LEVEL 1: NP= 2 RRATE= 1.000:
PARAMS.THRU_RATE= 1.000 PARAMS.DELAY_GB_GC=
0 PARAMS.DELAY_CC_GB=
0 PARAMS.TERMINALS=
31

*** SIMULATION:
GC ( 0): MSGOUT DELAY = 0 17
GB: STIME = 57802 MSGLEN = 0
GC ( 1): MSGIN DELAY = 0
GC ( 0): MSGOUT DELAY = 0
GB: STIME = 69750 MSGLEN = 0 17
GC ( 1): MSGIN DELAY = 0
GC ( 0): MSGOUT DELAY = 0
GB: STIME = 151372 MSGLEN = 0 17
GC ( 1): MSGIN DELAY = 0
GC ( 0): MSGOUT DELAY = 0
GB: STIME = 209459 MSGLEN = 0 10
GC ( 1): MSGIN DELAY = 0
GB: STIME = 267389 MSGLEN = 0 10
GC ( 0): MSGOUT DELAY = 0
GC ( 1): MSGIN DELAY = 0
GB: STIME = 384414 MSGLEN = 0 10
GC ( 0): MSGIN DELAY = 0

*** STATISTICS:
LEVEL 0:
SRQ_MAX = 10 DBB_MAX = 1
GC:
SVR_ID= 1 SVR.STIME= 384424 SVR.STIMEQ= 81:
AAM:
SVR_ID= 2 SVR.STIME= 210193 SVR.STIMEQ= 724:
SVR_ID= 1 SVR.STIME= 385128 SVR.STIMEQ= 704:
SAM:
SVR_ID= 1 SVR.STIME= 481150 SVR.STIMEQ= 266197:
SVR_ID= 2 SVR.STIME= 481592 SVR.STIMEQ= 220819:
LEVEL 1:
SRQ_MAX = 21 DBB_MAX = 1
GC:
SVR_ID= 1 SVR.STIME= 384424 SVR.STIMEQ= 81:
AAM:
SVR_ID= 2 SVR.STIME= 152346 SVR.STIMEQ= 1843:
SVR_ID= 1 SVR.STIME= 268088 SVR.STIMEQ= 1582:
SAM:
SVR_ID= 1 SVR.STIME= 400088 SVR.STIMEQ= 134592:
SVR_ID= 2 SVR.STIME= 400541 SVR.STIMEQ= 212533:

```

FIG. 7.5 STATISTICS OF SAMPLE SIMULATION SESSION

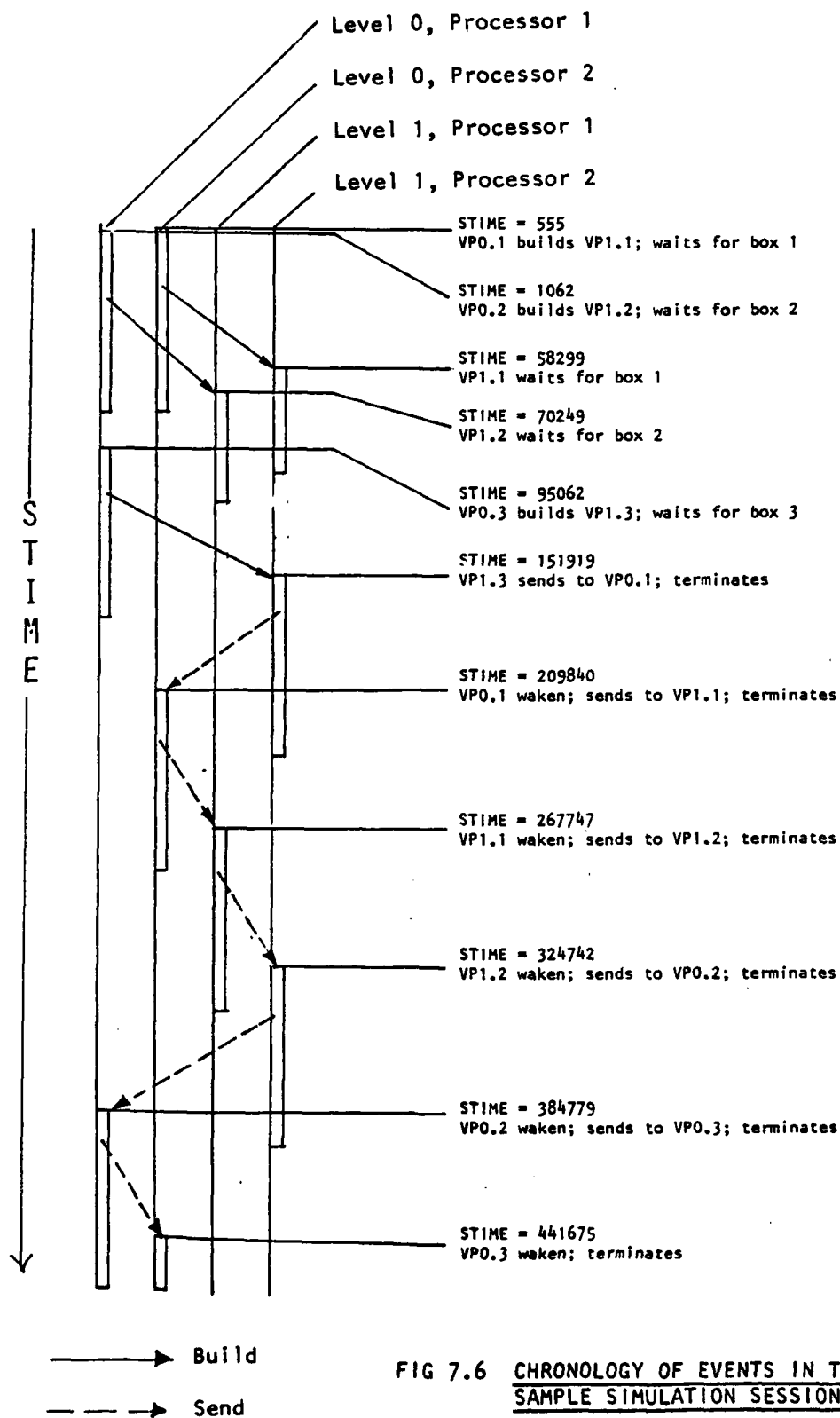


FIG 7.6 CHRONOLOGY OF EVENTS IN THE
 SAMPLE SIMULATION SESSION

8.0 CONCLUSION & OBSERVATIONS

SHELL is part of the first step in the gradual refinement of the design of INFOPLEX. As a simulator, it has two salient features.

The first is that SHELL offers a very easy interface with application programs. An ordinary PL/1 program can run under SHELL with minimal or no modification. The complete transparent context switching mechanism greatly eases the writing of application programs. Without this feature, an application would have to keep track of under which process it is working. Also, an application would not be able to exit (and return control to the simulator) from inside an inner block, since PL/1 does not allow control be transferred back into the inner block. This would severely handicap the programming style of the application program.

The second salient feature of SHELL is that it simulates a mixture of microscopic and macroscopic events. On the one hand, there are the short and simple events of data transaction among the machines; and on the other, the long and complex events of running application programs. This integrated approach offers a highly realistic simulation because one can study the interaction between the two types of events. For example, when one is studying the microscopic events of

traffic on the global bus, instead of having to use a statistical model to approximate the access pattern by the gateway controllers, one can run the procedures that generate the actual patterns. This can lead to insights which may otherwise be obscured.

It is hoped that through simulation using SHELL, a more optimal designs of INFOPLEX can be attained.

BIBLIOGRAPHY

- [Abraham] Abraham, M., "A Preliminary Design of the Control Structure of S.T.V.," unpublished lecture notes.
- [Hoare] Hoare, C.A.R., "Monitors: An Operating System Structure Concept," Comm. of the ACM, Vol. 17, No. 10 (1974).
- [Hsu 1] Hsu, Meichun, "A Preliminary Architectual Design For the Functional Hierarchy of the INFOPLEX Database Computer," Technical Report No. 5, Center for Information System Research, MIT, Cambridge.
- [Hsu 2] Hsu, Meichun, "Functional Hierarchy Environment," unpublished memo. 13, No. 2 (1979).
- [IBM] OS PL/1 Optimizing Compiler: Execution Logic, International Business Machine, INC Publication SY-0065.
- [Madnick] Madnick, S.E., "The INFOPLEX Database Computer: Concepts and directions," in Proceedings of IEEE Computer Conference, (1977), pp.168-176.
- [Wirth] Wirth, N., "Design and Implementation of Modula," Software - Practice and Experience, Vol. 7 (1977), pp.67-84.

APPENDIX A: LISTINGS OF PL/1 PROCEDURES

PL/1 Procedures:

AAHER
ADDQ
CONS
DEBUGR
EXECUTE
FINISH
GBER
GCER
QEVENT
SAHER
SEND
SHELL
SLEEP
STIMER
SVCER
SYNC
TERM
VPER
VPSTART
WAIT

SOURCE LISTING

NUMBER LEV NT

R

```

10      0 AAHER: PROC (LOS,SVR,EVENT);
30      1 0 DCL 1 LOS,
      70 LEVEL FIXED BIN,
      70 SRQ,
      71 SIZE FIXED BIN,
      71 MAX FIXED BIN,
      70 DBB,
      71 SIZE FIXED BIN,
      71 MAX FIXED BIN,
      70 VPS,
      71 TABLE ( 20),
      80 BDSA FIXED BIN (31),
      80 TDSA FIXED BIN (31),
      80 SAVESIZE FIXED BIN (31),
      80 SAFE PTR,
      80 PROCNAME CHAR (7) VAR,
      80 STATUS CHAR (12) VAR,
      80 MAIL PTR,
      80 WAIT,
      81 BOX FIXED BIN,
      81 MSG PTR,
      80 VPID FIXED BIN,
      80 LEVEL FIXED BIN,
      80 VTIME FIXED BIN (31)
      /* ADDR (BOTTOM DSA)
      /* ADDR (TOP DSA)
      /* SIZE OF AREA TO BE SAVED
      /* -> 1ST SAVBLK
      /* NAME OF TOP LEVEL PROC
      /* CHAIN OF INCOMING MAIL
      /* BOX AWAITING MAIL
      /* MSG IN WAIT.BOX
      /* INDEX IN THE VPST
      /* LEVEL
      /* VP 00010
      /* VP 00020
      /* VP 00030
      /* VP 00040
      /* VP 00050
      /* VP 00060
      /* VP 00070
      /* VP 00080
      /* VP 00090
      /* VP 00100
      /* VP 00110
      /* VP 00120
      /* VP 00130
      AAH00030
      AAH00040
      SVR00010
      SVR00020
      /* SVR00030
      SVR00040
      /* SVR00050
      SVR00060
      AAH00040
      AAH00050
      /* EVE00010
      EVE00020
      EVE00030
      EVE00040
      EVE00050
      AAH00050
      AAH00060
      AAH00070
      DEB00010
      DEB00020
      DEB00030

300040 1 0 DCL 1 SVR,
      70 NEXT PTR,
      70 STIME FIXED BIN (31),
      70 STIMEQ FIXED BIN (31),
      70 RTIME FIXED BIN (31),
      70 RRATE FIXED BIN (15,7),
      70 ID FIXED BIN
      /* ACCUMULATED STIME
      /* REAL CPU TICK RATE

500050 1 0 DCL 1 EVENT,
      70 NEXT PTR,
      70 STIME FIXED BIN (31),
      70 TYPE CHAR (12) VAR,
      70 INDEX FIXED BIN (31),
      70 PTR PTR
      /* NEXT EVENT

700060 1 0 DCL (THISLOS,THISSVR) PTR EXTERNAL STATIC;
700070 1 0 DCL STM FIXED BIN (31);
800010 1 0 DCL 1 DEBUG EXTERNAL STATIC,
      2 SLEEPS BIT(1) INIT ('0'B),
      2 SAHERS BIT(1) INIT ('0'B),

```

NUMBER LEV NT

```

          2 SCHEDULERS BIT(1) INIT ('0'B),
          2 SHELLS BIT(1) INIT ('0'B),
          2 AAHERS BIT(1) INIT ('0'B);
900010  1  0  DCL PT_ADDR PTR;
900030  1  0  DCL 1 PF_ADDR BASED (PT_ADDR),
          2 VPID FIXED BIN,
          2 BOXID FIXED BIN,
          2 PTR PTR;
1000010 1  0  DCL PT_S PTR;
1000030 1  0  DCL 1 PF_S BASED (PT_S),
          2 LEN FIXED BIN,
          2 PTR PTR;
1100010 1  0  DCL STIMER ENTRY RETURNS (FIXED BIN (31));
1200010 1  0  DCL QEVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR,
          FIXED BIN (31), PTR);
1300010 1  0  DCL SVCER ENTRY (FIXED BIN, PTR);
1300150 1  0  DCL (ADDR) BUILTIN;
1300170 1  0  THISLOS = ADDR (LOS);
1300180 1  0  THISSVR = ADDR (SVR);
1300200 1  0  IF (EVENT.TYPE = 'D') THEN DO;
1300210 1  1  PT_ADDR = EVENT.PTR;
1300220 1  1  LOS.DBB.SIZE = LOS.DBB.SIZE - 1;
1300230 1  1  END;
1300240 1  0  ELSE DO; /* EVENT.TYPE = S */
1300250 1  1  PT_S = EVENT.PTR;
1300260 1  1  LOS.SRQ.SIZE = LOS.SRQ.SIZE - PF_S.LEN;
1300270 1  1  PT_ADDR = PF_S.PTR;
1300280 1  1  FREE PF_S;
1300290 1  1  END;
1300310 1  0  IF PF_ADDR.VPID = 0 THEN DO;
1300320 1  1  CALL SVCER (PF_ADDR.BOXID,PF_ADDR.PTR);
1300330 1  1  FREE PF_ADDR;
1300340 1  1  END;
1300350 1  0  ELSE DO;
1300360 1  1  STM = STIMER;
1300370 1  1  CALL QEVENT (LOS.LEVEL+3+3,STM,'LOGMSG',0,PT_ADDR);
1300380 1  1  END;
1300400 1  0  STM = STIMER;
1300410 1  0  IF (DEBUG.AAHERS) THEN DO;
1300420 1  1  PUT SKIP EDIT (' AAHER: TO = ',SVR.STIME,' TI = ',
          STM,' QT = ',SVR.STIMEQ)
          (A,F(11),A,F(11),A,F(11));
          DEB00040
          DEB00050
          DEB00060
          PFA00010
          PFA00020
          PFA00030
          PFA00040
          PFA00050
          PFA00060
          PFS00010
          PFS00020
          PFS00030
          PFS00040
          PFS00050
          AAH00110
          STI00010
          QEV00010
          QEV00020
          SVC00010
          AAH00150
          AAH00160
          AAH00170
          AAH00180
          AAH00190
          AAH00200
          AAH00210
          AAH00220
          AAH00230
          AAH00240
          AAH00250
          AAH00260
          AAH00270
          AAH00280
          AAH00290
          AAH00300
          AAH00310
          AAH00320
          AAH00330
          AAH00340
          AAH00350
          AAH00360
          AAH00370
          AAH00380
          AAH00390
          AAH00400
          AAH00410
          AAH00420
          AAH00430
          AAH00440

```

PL/I OPTIMIZING COMPILER

AAHER: PROC (LOS,SVR,EVENT);

NUMBER LEV NT

```

1300450 1 1      END;
1300460 1 0      SVR.STIMEQ = SVR.STIMEQ + STM - SVR.STIME;
1300470 1 0      SVR.STIME = STM;
1300490 1 0 END;

```

AAH00450
AAH00460
AAH00470
AAH00480
AAH00490

SOURCE LISTING

NUMBER LEV NT

```
10      0  ADDQ: PROC (Q,PO);                                ADD000010
                                                    ADD000020
                                                    ADD000030
                                                    ADD000040
                                                    ADD000050
                                                    ADD000060
60      1  0  DCL (Q,PO,PQ2,PO,PT) PTR;                      ADD000070
70      1  0  DCL (ADDR,NULL) BUILTIN;                      ADD000080
                                                    ADD000090
                                                    ADD000100
                                                    ADD000110
                                                    ADD000120
                                                    ADD000130
                                                    ADD000140
                                                    ADD000150
                                                    ADD000160
                                                    ADD000170
                                                    ADD000180
                                                    ADD000190
                                                    ADD000200
                                                    ADD000210
                                                    ADD000220
                                                    ADD000230
                                                    ADD000240
                                                    ADD000250
                                                    ADD000260
                                                    ADD000270

100     1  0  /* TB == FIRST PART OF SVR OR EVENT */
      DCL 1 TB BASED (PT),
            2 NEXT PTR,
            2 STIME FIXED BIN (31);

140     1  0  PQ = ADDR(0);
150     1  0  PQ2 = PQ->TB.NEXT;
160     1  0  DO WHILE (PQ2 /= NULL);
170     1  1  IF (PQ2->TB.STIME >= PO->TB.STIME) THEN GOTO FOUND;
180     1  1  PQ = PQ2;
190     1  1  PQ2 = PQ->TB.NEXT;
200     1  1  END;

220     1  0  FOUND:
      PO->TB.NEXT = PQ2;
240     1  0  PQ->TB.NEXT = PO;

260     1  0  END ADDQ;
```

SOURCE LISTING

NUMBER LEV NT

10	0	CONS: PROC (CAR, CDR) RETURNS (PTR);	CON00010
			CON00020
		%INCLUDE LIST;.....	CON00030
100010	1	0 DCL LISTPT PTR;	LIS00010
			LIS00020
100030	1	0 DCL 1 LIST BASED (LISTPT),	LIS00030
		2 NEXT PTR,	LIS00040
		2 THIS PTR;	LIS00050
			LIS00060
		*****	CON00030
200040	1	0 DCL (CAR, CDR) PTR;	CON00040
			CON00050
200060	1	0 ALLOCATE LIST;	CON00060
200070	1	0 LIST.NEXT = CDR;	CON00070
200080	1	0 LIST.THIS = CAR;	CON00080
200090	1	0 RETURN (LISTPT);	CON00090
			CON00100
200110	1	0 END CONS;	CON00110

SOURCE LISTING

NUMBER LEV NT

10	0	EXECUTE: PROC (PROCNAME);	EXE00010
			EXE00020
30	1 0	DCL PROCNAME CHAR(*) VAR;	EXE00030
40	1 0	DCL TERM ENTRY;	EXE00040
			EXE00050
60	1 0	IF PROCNAME = 'TERM' THEN DO; CALL TERM; RETURN; END;	EXE00060
			EXE00070
80	1 0	PUT SKIP LIST (' ERROR: UNKNOWN PROCNAME. ',PROCNAME,	EXE00080
		' (EXECUTE).');	EXE00090
100	1 0	STOP;	EXE00100
			EXE00110
120	1 0	END EXECUTE;	EXE00120

SOURCE LISTING

NUMBER LEV NT

```
10      0  DEBUGR: PROC;                                DEB00010
                                                    DEB00020
                                                    DEB00030
100010  1  0  %INCLUDE DEBUG;.....DEB00010
                DCL 1 DEBUG EXTERNAL STATIC,            DEB00020
                2 SLEEP$ BIT(1) INIT ('0'B),            DEB00030
                2 SAHER$ BIT(1) INIT ('0'B),            DEB00040
                2 SCHEDULERS BIT(1) INIT ('0'B),         DEB00050
                2 SHELL$ BIT(1) INIT ('0'B),             DEB00060
                2 AAHER$ BIT(1) INIT ('0'B);             DEB00070
                .....DEB00080
200050  1  0      DEBUG.SAHER$ = '1'B;                  DEB00090
200060  1  0      DEBUG.SCHEDULERS$ = '1'B;              DEB00100
200070  1  0      DEBUG.SHELL$ = '1'B;
200080  1  0      DEBUG.AAHER$ = '1'B;
200100  1  0  END DEBUGR;
```

SOURCE LISTING

NUMBER LEV NT

```

10      0  FINISH: PROC;                                FIN00010
                                                    FIN00020
                                                    FIN00030
100010  1  0  DCL THISVP PTR EXTERNAL STATIC;           /* -> CURRENT VP, SET BY VPER */VPX00010
                                                    VPX00020
100030  1  0  DCL 1 VP BASED (THISVP), %INCLUDE VP;*****VPX00030
      80 BDSA FIXED BIN (31),                          /* ADDR (BOTTOM DSA) */VP 00010
      80 TDSA FIXED BIN (31),                          /* ADDR (TOP DSA) */VP 00020
      80 SAVESIZE FIXED BIN (31),                      /* SIZE OF AREA TO BE SAVED */VP 00030
      80 SAFE PTR,                                     /* -> 1ST SAVBLK */VP 00040
      80 PROCNAME CHAR (7) VAR,                       /* NAME OF TOP LEVEL PROC */VP 00050
      80 STATUS CHAR (12) VAR,                        VP 00060
      80 MAIL PTR,                                     /* CHAIN OF INCOMING MAIL */VP 00070
      80 WAIT,                                         VP 00080
      81 BOX FIXED BIN,                               /* BOX AWAITING MAIL */VP 00090
      81 MSG PTR,                                     /* MSG IN WAIT.BOX */VP 00100
      80 VPID FIXED BIN,                             /* INDEX IN THE VPST */VP 00110
      80 LEVEL FIXED BIN,                             /* LEVEL */VP 00120
      80 VTIME FIXED BIN (31)                        VP 00130
      ..... ;                                         VPX00030
      .....                                         VPX00040
      .....                                         FIN00030
      .....                                         FIN00040
500010  1  0  %INCLUDE QEVENT;*****FIN00050
      DCL QEVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR,
      FIXED BIN (31), PTR);                         QEV00010
      .....                                         QEV00020
      .....                                         FIN00050
500060  1  0  DCL NULL BUILTIN;                       FIN00060
500070  1  0  DCL SLEEP ENTRY;                       FIN00070
      %INCLUDE STIMER;*****FIN00080
600010  1  0  DCL STIMER ENTRY RETURNS (FIXED BIN (31));
      .....                                         STI00010
      .....                                         FIN00080
      .....                                         FIN00090
600100  1  0  VP.WAIT.BOX = -1;                       FIN00100
600110  1  0  CALL QEVENT (VP.LEVEL*3+3,STIMER,'FINISH',VP.VPID,NULL);
      .....                                         FIN00110
600120  1  0  CALL SLEEP;                             FIN00120
      .....                                         FIN00130
600140  1  0  END;                                     FIN00140

```

SOURCE LISTING

NUMBER LEV NT

```

10      0  GBER: PROC (SVR,EVENT);                                GBE00010
30      1  0  DCL 1 SVR, XINCLUDE SVR;.....GBE00020
              70 NEXT PTR,                                       GBE00020
              70 STIME FIXED BIN (31),                             SVR00010
              70 STIMEQ FIXED BIN (31),                           SVR00020
              70 RTIME FIXED BIN (31),                             /* ACCUMULATED STIME */SVR00030
              70 RRATE FIXED BIN (15,7),                          /* REAL CPU TICK RATE */SVR00040
              70 ID FIXED BIN                                     SVR00050
              .....;                                             SVR00060
200040   1  0  DCL 1 EVENT, XINCLUDE EVENT;.....GBE00030
              70 NEXT PTR,                                       GBE00040
              70 STIME FIXED BIN (31),                             /* NEXT EVENT */EVE00010
              70 TYPE CHAR (12) VAR,                             EVE00020
              70 INDEX FIXED BIN (31),                             EVE00030
              70 PTR PTR                                           EVE00040
              .....;                                             EVE00050
400050   1  0  DCL STM FIXED BIN (31);                             GBE00060
              XINCLUDE PARAMS;.....GBE00070
500010   1  0  DCL 1 PARAMS EXTERNAL STATIC,                       PAR00010
              2 THRU_RATE FIXED BIN (31,7),                       PAR00020
              2 DELAY_GB_GC FIXED BIN (31),                       PAR00030
              2 DELAY_GC_GB FIXED BIN (31),                       PAR00040
              2 TERMINALS FIXED BIN;                               PAR00050
              .....;                                             PAR00060
600070   1  0  DCL STATS FILE EXTERNAL;                           GBE00070
              XINCLUDE PFLEVEL;.....GBE00080
700010   1  0  DCL PT_LEVEL PTR;                                   PFL00010
700030   1  0  DCL 1 PF_LEVEL BASED (PT_LEVEL),                   PFL00020
              2 LEVEL FIXED BIN,                                   PFL00030
              2 PTR PTR;                                           PFL00040
              .....;                                             PFL00050
800010   1  0  DCL PT_MSG PTR;                                     GBE00060
              XINCLUDE PFMSG;.....GBE00070
800030   1  0  DCL 1 PT_MSG BASED (PT_MSG),                       PFM00010
              2 LEN FIXED BIN,                                     PFM00020
              2 TYPE CHAR (12) VAR,                               PFM00030
              2 PTR PTR;                                           PFM00040
              .....;                                             PFM00050
              .....;                                             PFM00060
              .....;                                             PFM00070
              .....;                                             GBE00080
              .....;                                             GBE00090
              XINCLUDE OEVENT;.....GBE00100
              .....;                                             GBE00110

```

AD-A116 592

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
SHELL: A SIMULATOR FOR THE SOFTWARE TEST VEHICLE OF THE INFOPLE--ETC(U)
JAN 82 T TO N00039-81-C-0663
CISR-M010-0201-08 NL

UNCLASSIFIED

20 2

1-16-82

END

DATE

12-11-82

OF 4

NUMBER LEV NT

```
900010 1 0 DCL QEVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR,  
      FIXED BIN (31), PTR);  
      *****  
900130 1 0 STM = EVENT.STIME + PARAMS.DELAY_GB_GC;  
900140 1 0 PT_LEVEL = EVENT.PTR;  
900150 1 0 PT_MSG = PF_LEVEL.PTR;  
900160 1 0 PUT FILE (STATS) SKIP EDIT (' GB: STIME = ',SVR.STIME,  
      ' MSGLEN = ',PF_MSG.LEN) (A,F(11),A,F(11));  
900180 1 0 CALL QEVENT (PF_LEVEL.LEVEL+3+1,STM,'MSGIN',0,PT_MSG);  
900190 1 0 FREE PF_LEVEL;  
900200 1 0 SVR.STIME = EVENT.STIME;  
900220 1 0 END GBER;
```

```
QEV00010  
QEV00020  
GBE00110  
GBE00120  
GBE00130  
GBE00140  
GBE00150  
GBE00160  
GBE00170  
GBE00180  
GBE00190  
GBE00200  
GBE00210  
GBE00220
```

SOURCE LISTING

NUMBER LEV NT

```

10      0 GCER: PROC (LOS,SVR,EVENT);
30      1 0 DCL 1 LOS,
          70 LEVEL FIXED BIN,
          70 SRQ,
          71 SIZE FIXED BIN,
          71 MAX FIXED BIN,
          70 DBB,
          71 SIZE FIXED BIN,
          71 MAX FIXED BIN,
          70 VPS,
          71 TABLE (      20),
          80 BDSA FIXED BIN (31), /* ADDR (BOTTOM DSA)
          80 TDSA FIXED BIN (31), /* ADDR (TOP DSA)
          80 SAVESIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED
          80 SAFE PTR, /* -> 1ST SAVBLK
          80 PROCNAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC
          80 STATUS CHAR (12) VAR, /* CHAIN OF INCOMING MAIL
          80 MAIL PTR, /* BOX AWAITING MAIL
          80 WAIT, /* MSG IN WAIT.BOX
          81 BOX FIXED BIN, /* INDEX IN THE VPST
          81 MSG PTR, /* LEVEL
          80 VPID FIXED BIN,
          80 LEVEL FIXED BIN,
          80 VTIME FIXED BIN (31)
300040 1 0 DCL 1 SVR,
          70 NEXT PTR,
          70 STIME FIXED BIN (31), /* ACCUMULATED STIME
          70 STIMEQ FIXED BIN (31), /* REAL CPU TICK RATE
          70 RTIME FIXED BIN (31),
          70 RRATE FIXED BIN (15,7),
          70 ID FIXED BIN
500050 1 0 DCL 1 EVENT,
          70 NEXT PTR, /* NEXT EVENT
          70 STIME FIXED BIN (31),
          70 TYPE CHAR (12) VAR,
          70 INDEX FIXED BIN (31),
          70 PTR PTR
800010 1 0 DCL 1 PARAMS EXTERNAL STATIC,
          2 THRU_RATE FIXED BIN (31,7),
          2 DELAY_GB_GC FIXED BIN (31),
          2 DELAY_GC_GB FIXED BIN (31),
          2 TERMINALS FIXED BIN;

```

```

GCE00010
GCE00020
GCE00030
LOS00010
LOS00020
LOS00030
LOS00040
LOS00050
LOS00060
LOS00070
LOS00080
LOS00090
1
*/VP 00010
*/VP 00020
*/VP 00030
*/VP 00040
*/VP 00050
VP 00060
*/VP 00070
VP 00080
*/VP 00090
*/VP 00100
*/VP 00110
*/VP 00120
VP 00130
GCE00030
GCE00040
SVR00010
SVR00020
*/SVR00030
SVR00040
*/SVR00050
SVR00060
GCE00040
GCE00050
*/EVE00010
EVE00020
EVE00030
EVE00040
EVE00050
GCE00050
PAR00010
PAR00020
PAR00030
PAR00040
PAR00050

```

NUMBER LEV NT

800070	1	0	DCL STATS FILE EXTERNAL;	GCE00070
				GCE00080
900010	1	0	DCL QEVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR,	QEV00010
			FIXED BIN (31), PTR);	QEV00020
900100	1	0	DCL MAX BUILTIN;	GCE00100
				GCE00110
900120	1	0	PUT FILE (STATS) SKIP EDIT (' GC (' ,LOS.LEVEL,') : ' ,EVENT.TYPE,	GCE00120
			' DELAY = ' ,MAX(0,SVR.STIME-EVENT.STIME))	GCE00130
			(A,F(2),A,A(6),A,F(11));	GCE00140
				GCE00150
900160	1	0	IF (EVENT.TYPE = 'MSGIN') THEN CALL GCERIN;	GCE00160
900170	1	0	ELSE CALL GCEROUT;	GCE00170
				GCE00180
900190	1	0	RETURN;	GCE00190
				GCE00200
				GCE00210
900220	1	0	GCERIN: PROC;	GCE00220
				GCE00230
1000010	2	0	DCL PT_MSG PTR;	PFM00010
				PFM00020
1000030	2	0	DCL 1 PF_MSG BASED (PT_MSG),	PFM00030
			2 LEN FIXED BIN,	PFM00040
			2 TYPE CHAR (12) VAR,	PFM00050
			2 PTR PTR;	PFM00060
1000250	2	0	DCL STM FIXED BIN (31);	GCE00250
				GCE00260
1000270	2	0	PT_MSG = EVENT.PTR;	GCE00270
1000280	2	0	IF (PF_MSG.TYPE = 'S') THEN DO;	GCE00280
1000290	2	1	LOS.SRQ.SIZE = LOS.SRQ.SIZE + PF_MSG.LEN;	GCE00290
1000300	2	1	LOS.SRQ.MAX = MAX (LOS.SRQ.MAX, LOS.SRQ.SIZE);	GCE00300
1000310	2	1	END;	GCE00310
1000320	2	0	ELSE DO; /* PF_MSG.TYPE = 'D' */	GCE00320
1000330	2	1	LOS.DBB.SIZE = LOS.DBB.SIZE + 1;	GCE00330
1000340	2	1	LOS.DBB.MAX = MAX (LOS.DBB.MAX, LOS.DBB.SIZE);	GCE00340
1000350	2	1	END;	GCE00350
				GCE00360
1000370	2	0	STM = SVR.STIME + PARAMS.THRU_RATE * PF_MSG.LEN;	GCE00370
1000380	2	0	CALL QEVENT (LOS.LEVEL+3+2,STM,PF_MSG.TYPE,0,PF_MSG.PTR);	GCE00380
1000390	2	0	FREE PF_MSG;	GCE00390
1000400	2	0	SVR.STIMEQ = SVR.STIMEQ + STM - SVR.STIME;	GCE00400
1000410	2	0	SVR.STIME = STM;	GCE00410
				GCE00420
1000430	2	0	END GCERIN;	GCE00430
				GCE00440
				GCE00450
1000460	1	0	GCEROUT: PROC;	GCE00460
				GCE00470
1100010	2	0	DCL PT_LEVEL PTR;	PFL00010
				PFL00020

NUMBER LEV NT

R

1100030	2	0	DCL 1 PF_LEVEL BASED (PT_LEVEL), 2 LEVEL FIXED BIN, 2 PTR PTR;	PFL00030 PFL00040 PFL00050
1200010	2	0	DCL PT_MSG PTR;	PFM00010 PFM00020
1200030	2	0	DCL 1 PF_MSG BASED (PT_MSG), 2 LEN FIXED BIN, 2 TYPE CHAR (12) VAR, 2 PTR PTR;	PFM00030 PFM00040 PFM00050 PFM00060
1200500	2	0	DCL STM FIXED BIN (31);	GCE00500
1200520	2	0	STM = SVR.STIME + PARAMS.DELAY_GC_GB;	GCE00510
1200530	2	0	CALL QEVENT (0, STM, 'MSG', 0, EVENT.PTR);	GCE00520
1200540	2	0	PT_LEVEL = EVENT.PTR;	GCE00530
1200550	2	0	PT_MSG = PF_LEVEL.PTR;	GCE00540
1200560	2	0	STM = PARAMS.THRU_RATE + PF_MSG.LEN;	GCE00550
1200570	2	0	SVR.STIMEQ = SVR.STIMEQ + STM;	GCE00560
1200580	2	0	SVR.STIME = SVR.STIME + STM;	GCE00570 GCE00580 GCE00590
1200600	2	0	END GCEROUT;	GCE00600 GCE00610 GCE00620
1200630	1	0	END GCER;	GCE00630

SOURCE LISTING

NUMBER LEV NT

R

10	0	QEVENT: PROC (IQ,STIME,TYPE,INDEX,PTR);	QEV00010	
			QEV00020	
100010	1	0 DCL SQUEUE (0: 6) PTR EXTERNAL STATIC;	QS 00010	1
100020	1	0 DCL EQUEUE (0: 6) PTR EXTERNAL STATIC;	QS 00020	1
			QEV00050	
100060	1	0 DCL EPT PTR;	QEV00060	
100070	1	0 DCL 1 EVENT BASED (EPT),	QEV00070	
		70 NEXT PTR, /* NEXT EVENT	*/ EVE00010	
		70 STIME FIXED BIN (31),	EVE00020	
		70 TYPE CHAR (12) VAR,	EVE00030	
		70 INDEX FIXED BIN (31),	EVE00040	
		70 PTR PTR	EVE00050	
			QEV00070	
			QEV00080	
200090	1	0 DCL IQ FIXED BIN;	QEV00090	
200100	1	0 DCL STIME FIXED BIN (31);	QEV00100	
200110	1	0 DCL TYPE CHAR(*) VAR;	QEV00110	
200120	1	0 DCL INDEX FIXED BIN (31);	QEV00120	
200130	1	0 DCL PTR PTR;	QEV00130	
			QEV00140	
200150	1	0 DCL ADDQ ENTRY;	QEV00150	
200160	1	0 DCL NULL BUILTIN;	QEV00160	
			QEV00170	
200180	1	0 ALLOCATE EVENT;	QEV00180	
200190	1	0 EVENT.NEXT = NULL;	QEV00190	
200200	1	0 EVENT.STIME = STIME;	QEV00200	
200210	1	0 EVENT.TYPE = TYPE;	QEV00210	
200220	1	0 EVENT.INDEX = INDEX;	QEV00220	
200230	1	0 EVENT.PTR = PTR;	QEV00230	
200240	1	0 CALL ADDQ (EQUEUE(IQ),EPT);	QEV00240	
			QEV00250	
200260	1	0 END QEVENT;	QEV00260	

SOURCE LISTING

NUMBER LEV NT

```

10      0 SAHER: PROC (LOS,SVR,EVENT);
30      1 0 DCL 1 LOS,
          70 LEVEL FIXED BIN,
          70 SRQ,
          71 SIZE FIXED BIN,
          71 MAX FIXED BIN,
          70 DBB,
          71 SIZE FIXED BIN,
          71 MAX FIXED BIN,
          70 VPS,
          71 TABLE ( 20),
          80 BDSA FIXED BIN (31),
          80 TDSA FIXED BIN (31),
          80 SAVESIZE FIXED BIN (31),
          80 SAFE PTR,
          80 PROCNAME CHAR (7) VAR,
          80 STATUS CHAR (12) VAR,
          80 MAIL PTR,
          80 WAIT,
          81 BOX FIXED BIN,
          81 MSG PTR,
          80 VPID FIXED BIN,
          80 LEVEL FIXED BIN,
          80 VTIME FIXED BIN (31)
          /* ADDR (BOTTOM DSA)
          /* ADDR (TOP DSA)
          /* SIZE OF AREA TO BE SAVED
          /* -> 1ST SAVBLK
          /* NAME OF TOP LEVEL PROC
          /* CHAIN OF INCOMING MAIL
          /* BOX AWAITING MAIL
          /* MSG IN WAIT.BOX
          /* INDEX IN THE VPST
          /* LEVEL
          /* ACCUMULATED STIME
          /* REAL CPU TICK RATE
          /* NEXT EVENT
          /* VP 00010
          /* VP 00020
          /* VP 00030
          /* VP 00040
          /* VP 00050
          VP 00060
          /* VP 00070
          VP 00080
          /* VP 00090
          /* VP 00100
          /* VP 00110
          /* VP 00120
          VP 00130
          SAH00030
          SAH00040
          SVR00010
          SVR00020
          /*SVR00030
          SVR00040
          /*SVR00050
          SVR00060
          SAH00040
          SAH00050
          /* EVE00010
          EVE00020
          EVE00030
          EVE00040
          EVE00050
          SAH00050
          SAH00060
          SAH00070
          SAH00080
          PAR00010
          PAR00020
300040 1 0 DCL 1 SVR,
          70 NEXT PTR,
          70 STIME FIXED BIN (31),
          70 STIMEQ FIXED BIN (31),
          70 RTIME FIXED BIN (31),
          70 RRATE FIXED BIN (15,7),
          70 ID FIXED BIN
          500050 1 0 DCL 1 EVENT,
          70 NEXT PTR,
          70 STIME FIXED BIN (31),
          70 TYPE CHAR (12) VAR,
          70 INDEX FIXED BIN (31),
          70 PTR PTR
700070 1 0 DCL (THISLOS,THISSVR) PTR EXTERNAL STATIC;
700080 1 0 DCL STM FIXED BIN (31);
800010 1 0 DCL 1 PARAMS EXTERNAL STATIC,
          2 THRU_RATE FIXED BIN (31,7).

```

NUMBER LEV NT

```

          2 DELAY_GB_GC FIXED BIN (31);
          2 DELAY_GC_GB FIXED BIN (31);
          2 TERMINALS FIXED BIN;
900010 1 0 DCL 1 DEBUG EXTERNAL STATIC;
          2 SLEEPS BIT(1) INIT ('0'B);
          2 SAHER$ BIT(1) INIT ('0'B);
          2 SCHEDULERS BIT(1) INIT ('0'B);
          2 SHELL$ BIT(1) INIT ('0'B);
          2 SAHER$ BIT(1) INIT ('0'B);
          2 DELAY_GB_GC FIXED BIN (31);
          2 DELAY_GC_GB FIXED BIN (31);
          2 TERMINALS FIXED BIN;
900120 1 0 DCL (ADDR,NULL) BUILTIN;
1000010 1 0 DCL STIMER ENTRY RETURNS (FIXED BIN (31));
1000140 1 0 DCL VPER ENTRY;

1000160 1 0 THISLOS = ADDR (LOS);
1000170 1 0 THISSVR = ADDR (SVR);

1000190 1 0 IF (DEBUG.SAHER$) THEN DO;
1000200 1 1 PUT SKIP EDIT (' SAHER: LEVEL = ',LOS.LEVEL,
          ' S = (' ,SVR.ID.SVR.STIME,'),
          ' E = (' ,EVENT.TYPE,EVENT.STIME,'))
          (A,F(2), A,F(2),F(11),A, A,A(12),F(11),A);

1000240 1 1 END;

1000260 1 0 IF (EVENT.TYPE = 'LOGMSG') THEN DO;
1000270 1 1 CALL MAILMAN (LOS.VPS, EVENT.PTR);
1000280 1 1 END;
1000290 1 0 ELSE IF (EVENT.TYPE = 'SYNC') THEN DO;
1000300 1 1 IF (LOS.VPS.TABLE(EVENT.INDEX).STATUS = 'RUNNING') THEN
          LOS.VPS.TABLE(EVENT.INDEX).STATUS = 'RUNNABLE';
          /* DO NOT CHANGE NASCENT */
1000330 1 1 END;
1000340 1 0 ELSE IF (EVENT.TYPE = 'WAIT') THEN DO;
          /* WAIT.BOX WAS SET EARLIER */
1000360 1 1 LOS.VPS.TABLE(EVENT.INDEX).STATUS = 'BLOCKED';
1000370 1 1 END;
1000380 1 0 ELSE IF (EVENT.TYPE = 'FINISH') THEN DO;
          /* WAIT.BOX WAS SET EARLIER */
1000400 1 1 LOS.VPS.TABLE(EVENT.INDEX).STATUS = 'VOID';
1000410 1 1 END;
1000420 1 0 ELSE DO; /* EVENT.TYPE = INIT */
1000430 1 1 CALL INITER (LOS.VPS);
1000440 1 1 RETURN; /* DO NOT SCHEDULE */
1000450 1 1 END;

1000470 1 0 CALL CHECK_MAIL (LOS.VPS);
1000480 1 0 CALL SCHEDULER (LOS.VPS);

1000500 1 0 STM = STIMER;

```

```

PAR00030
PAR00040
PAR00050
DEB00010
DEB00020
DEB00030
DEB00040
DEB00050
DEB00060
SAH00110
SAH00120
STI00010
SAH00140
SAH00150
SAH00160
SAH00170
SAH00180
SAH00190
SAH00200
SAH00210
SAH00220
SAH00230
SAH00240
SAH00250
SAH00260
SAH00270
SAH00280
SAH00290
SAH00300
SAH00310
SAH00320
SAH00330
SAH00340
SAH00350
SAH00360
SAH00370
SAH00380
SAH00390
SAH00400
SAH00410
SAH00420
SAH00430
SAH00440
SAH00450
SAH00460
SAH00470
SAH00480
SAH00490
SAH00500

```

與

```

1000510 1 0 SVR.STIMEQ = SVR.STIMEQ + STM - SVR.STIME; SAH00510
1000520 1 0 SVR.STIME = STM; SAH00520
1000540 1 0 RETURN; SAH00530
SAH00540
SAH00550
SAH00560
1000570 1 0 MAILMAN: PROC (VPS, PT_ADDR); SAH00570
SAH00580
1000590 2 0 DCL 1 VPS, 2 TABLE ( 20), SAH00590
80 BDSA FIXED BIN (31), /* ADDR (BOTTOM DSA) */VP 00010
80 TOSA FIXED BIN (31), /* ADDR (TOP DSA) */VP 00020
80 SAVESIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED */VP 00030
80 SAFE PTR, /* -> 1ST SAVBLK */VP 00040
80 PROCNAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC */VP 00050
80 STATUS CHAR (12) VAR, VP 00060
80 MAIL PTR, /* CHAIN OF INCOMING MAIL */VP 00070
80 WAIT, VP 00080
81 BOX FIXED BIN, /* BOX AWAITING MAIL */VP 00090
81 MSG PTR, /* MSG IN WAIT.BOX */VP 00100
80 VPID FIXED BIN, /* INDEX IN THE VPST */VP 00110
80 LEVEL FIXED BIN, /* LEVEL */VP 00120
80 VTIME FIXED BIN (31) VP 00130
; SAH00590
1200010 2 0 DCL PT_ADDR PTR; PFA00010
PFA00020
1200030 2 0 DCL 1 PF_ADDR BASED (PT_ADDR), PFA00030
2 VPID FIXED BIN, PFA00040
2 BOXID FIXED BIN, PFA00050
2 PTR PTR; PFA00060
1300010 2 0 DCL BOXPT PTR; BOX00010
BOX00020
1300030 2 0 DCL 1 BOX BASED (BOXPT), BOX00030
2 NEXT PTR, BOX00040
2 ID FIXED BIN, BOX00050
2 LIST PTR; BOX00060
1300620 2 0 DCL VPID FIXED BIN; SAH00620
SAH00630
1400010 2 0 DCL CONS ENTRY (PTR, PTR) RETURNS (PTR); CON00010
SAH00650
1400660 2 0 VPID = PF_ADDR.VPID; SAH00660
1400670 2 0 IF (VPS.TABLE(VPID).STATUS = 'VOID') THEN DO; SAH00670
1400680 2 1 PUT SKIP EDIT ( 'WARNING: ILLEGAL ADDRESS ',LOS.LEVEL, SAH00680
'/',VPID) (A,F(2),A,F(2)); SAH00690
1400700 2 1 FREE PF_ADDR; SAH00700
/* CANNOT FREE VARIABLE FORMAT MSG */ SAH00710
1400720 2 1 RETURN; SAH00720
1400730 2 1 END; SAH00730
SAH00740
1400750 2 0 BOXPT = VPS.TABLE(VPID).MAIL; SAH00750

```

- 95 -

NUMBER LEV NT

```

1400760 2 0 DO WHILE (BOXPT ^= NULL);
1400770 2 1 IF BOX.ID = PF_ADDR.BOXID THEN GOTO FOUND;
1400780 2 1 BOXPT = BOX.NEXT;
1400790 2 1 END;

/* CREATE NEW BOX */
1400820 2 0 ALLOCATE BOX;
1400830 2 0 BOX.ID = PF_ADDR.BOXID;
1400840 2 0 BOX.LIST = NULL;
1400850 2 0 BOX.NEXT = VPS.TABLE(VPID).MAIL;
1400860 2 0 VPS.TABLE(VPID).MAIL = BOXPT;

1400880 2 0 FOUND:
1400890 2 0 BOX.LIST = CONS(PF_ADDR.PTR, BOX.LIST);
1400900 2 0 FREE PF_ADDR;

1400920 2 0 END MAILMAN;

1400950 1 0 CHECK_MAIL: PROC (VPS);
1400970 2 0 DCL 1 VPS, 2 TABLE ( 20),
      80 BDSA FIXED BIN (31), /* ADDR (BOTTOM DSA)
      80 TDSA FIXED BIN (31), /* ADDR (TOP DSA)
      80 SAVESIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED
      80 SAFE PTR, /* -> 1ST SAVBLK
      80 PROCNAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC
      80 STATUS CHAR (12) VAR, /* CHAIN OF INCOMING MAIL
      80 MAIL PTR, /* BOX AWAITING MAIL
      80 WAIT, /* MSG IN WAIT.BOX
      81 BOX FIXED BIN, /* INDEX IN THE VPST
      81 MSG PTR, /* LEVEL
      80 VPID FIXED BIN,
      80 LEVEL FIXED BIN,
      80 VTIME FIXED BIN (31);

1600010 2 0 DCL BOXPT PTR;
1600030 2 0 DCL 1 BOX BASED (BOXPT),
      2 NEXT PTR,
      2 ID FIXED BIN,
      2 LIST PTR;
1700010 2 0 DCL LISTPT PTR;
1700030 2 0 DCL 1 LIST BASED (LISTPT),
      2 NEXT PTR,
      2 THIS PTR;
1701010 2 0 DCL IV FIXED BIN;

```

```

SAH00760
SAH00770
SAH00780
SAH00790
SAH00800
SAH00810
SAH00820
SAH00830
SAH00840
SAH00850
SAH00860
SAH00870
SAH00880
SAH00890
SAH00900
SAH00910
SAH00920
SAH00930
SAH00940
SAH00950
SAH00960
SAH00970
*/VP 00010
*/VP 00020
*/VP 00030
*/VP 00040
*/VP 00050
VP 00060
*/VP 00070
VP 00080
*/VP 00090
*/VP 00100
*/VP 00110
*/VP 00120
VP 00130
SAH00970
SAH00980
BOX00010
BOX00020
BOX00030
BOX00040
BOX00050
BOX00060
LIS00010
LIS00020
LIS00030
LIS00040
LIS00050
SAH01010

```

NUMBER LEV NT

R

```

1701020 2 0 DCL PT PTR; SAH01020
1701040 2 0 DO IV = 1 TO 20; SAH01030
1701050 2 1 IF VPS.TABLE(IV).STATUS = 'BLOCKED' THEN DO; SAH01040 1
1701060 2 2 BOXPT = VPS.TABLE(IV).MAIL; SAH01050
1701070 2 2 DO WHILE (BOXPT ^= NULL); SAH01060
1701080 2 3 IF BOX.ID = VPS.TABLE(IV).WAIT.BOX THEN DO; SAH01070
1701090 2 4 IF BOX.LIST ^= NULL THEN DO; SAH01080
1701100 2 5 LISTPT = BOX.LIST; SAH01090
1701110 2 5 PT = LIST.THIS; SAH01100
1701120 2 5 BOX.LIST = LIST.NEXT; SAH01110
1701130 2 5 FREE LIST; SAH01120
1701140 2 5 VPS.TABLE(IV).WAIT.MSG = PT; SAH01130
1701150 2 5 VPS.TABLE(IV).STATUS = 'RUNNABLE'; SAH01140
1701160 2 5 GOTO NEXT_VP; SAH01150
1701170 2 5 END; SAH01160
1701180 2 4 END; SAH01170
1701190 2 3 BOXPT = BOX.NEXT; SAH01180
1701200 2 3 END; SAH01190
1701210 2 2 END; SAH01200
1701220 2 1 NEXT_VP: END; SAH01210
SAH01220
SAH01230
1701240 2 0 END CHECK_MAIL; SAH01240
SAH01250
SAH01260
1701270 1 0 SCHEDULER: PROC (VPS); SAH01270
SAH01280
1701290 2 0 DCL 1 VPS, 2 TABLE ( 20), SAH01290 1
80 BDSA FIXED BIN (31), /* ADDR (BOTTOM DSA) */VP 00010
80 TDSA FIXED BIN (31), /* ADDR (TOP DSA) */VP 00020
80 SAVESIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED */VP 00030
80 SAFE PTR, /* -> 1ST SAVBLK */VP 00040
80 PROCNAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC */VP 00050
80 STATUS CHAR (12) VAR, VP 00080
80 MAIL PTR, /* CHAIN OF INCOMING MAIL */VP 00070
80 WAIT, VP 00080
81 BOX FIXED BIN, /* BOX AWAITING MAIL */VP 00090
81 MSG PTR, /* MSG IN WAIT.BOX */VP 00100
80 VPID FIXED BIN, /* INDEX IN THE VPST */VP 00110
80 LEVEL FIXED BIN, /* LEVEL */VP 00120
80 VTIME FIXED BIN (31) VP 00130
; SAH01290
SAH01300
1801310 2 0 DCL (VTM,VTM0) FIXED BIN (31); SAH01310
1801320 2 0 DCL YOUNGEST FIXED BIN; SAH01320
1801330 2 0 DCL IV FIXED BIN; SAH01330
SAH01340
1801350 2 0 YOUNGEST = 0; SAH01350
1801360 2 0 VTM0 = 2147483647; SAH01360

```

NUMBER LEV NT

R

```

1801380 2 0      DO IV = 1 TO      20;
1801390 2 1      IF (VPS.TABLE(IV).STATUS = 'RUNNABLE') ;
                {VPS.TABLE(IV).STATUS = 'NASCENT'} THEN DO;
1801410 2 2          VTM = VPS.TABLE(IV).VTIME;
1801420 2 2          IF VTM < VTMO THEN DO;
1801430 2 3              YOUNGEST = IV;
1801440 2 3              VTMO = VTM;
1801450 2 3          END;
1801460 2 2      END;
1801470 2 1      END;

1801490 2 0      IF (DEBUG.SCHEDULERS) THEN DO;
1801500 2 1          PUT SKIP EDIT (' SCHEDULER: YOUNGEST = ',YOUNGEST) (A,F(3));
1801510 2 1      END;

1801530 2 0      IF (YOUNGEST > 0) THEN DO;
1801540 2 1          CALL VPER (VPS.TABLE(YOUNGEST));
1801550 2 1      END;

1801570 2 0      END SCHEDULER;

1801600 1 0      INITER: PROC (VPS);

1801620 2 0      DCL 1 VPS, 2 TABLE (      20),
                80 BDSA FIXED BIN (31), /* ADDR (BOTTOM DSA)
                80 TOSA FIXED BIN (31), /* ADDR (TOP DSA)
                80 SAVESIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED
                80 SAFE PTR, /* -> 1ST SAVBLK
                80 PROCNAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC
                80 STATUS CHAR (12) VAR,
                80 MAIL PTR, /* CHAIN OF INCOMING MAIL
                80 WAIT,
                81 BOX FIXED BIN, /* BOX AWAITING MAIL
                81 MSG PTR, /* MSG IN WAIT.BOX
                80 VPID FIXED BIN, /* INDEX IN THE VPST
                80 LEVEL FIXED BIN, /* LEVEL
                80 VTIME FIXED BIN (31)
                VP 00130
                SAH01620
                PAR00010
                PAR00020
                PAR00030
                PAR00040
                PAR00050
                SAH01640
                MSG00010
                MSG00020
                MSG00030

2000010 2 0      DCL 1 PARAMS EXTERNAL STATIC,
                2 THRU_RATE FIXED BIN (31,7),
                2 DELAY_GB_GC FIXED BIN (31),
                2 DELAY_GC_GB FIXED BIN (31),
                2 TERMINALS FIXED BIN;

2001640 2 0      DCL IV FIXED BIN;
2100010 2 0      DCL MSGLEN FIXED BIN;
2100020 2 0      DCL MSGPT PTR;
2100030 2 0      DCL 1 MSG BASED (MSGPT),

```

NUMBER LEV NT

```
                2 LEN FIXED BIN,  
                2 STR CHAR (MSGLEN REFER (LEN));  
2200010  2  0  DCL VPSTART ENTRY (/= VP =/, CHAR(=) VAR, PTR);  
  
2201680  2  0      MSGLEN = 16;  
2201690  2  0      DO IV = 1 TO PARAMS.TERMINALS;  
2201700  2  1          ALLOCATE MSG;  
2201710  2  1          MSG.STR = 'SET UP BY INITER';  
2201720  2  1          CALL VPSTART (VPS.TABLE(IV),'TERM',MSGPT);  
2201730  2  1      END;  
  
2201750  2  0  END INITER;  
  
2201780  1  0  END SAHER;
```

```
MSG00040  
MSG00050  
VPS00010  
SAH01670  
SAH01680  
SAH01690  
SAH01700  
SAH01710  
SAH01720  
SAH01730  
SAH01740  
SAH01750  
SAH01760  
SAH01770  
SAH01780
```

SOURCE LISTING

NUMBER LEV NT

```

10      0  SEND: PROC (LEVEL,VPID,BOXID,TYPE,LEN,MPT);          SEN00010
                                           SEN00020
30      1  0  DCL (LEVEL,VPID,BOXID,LEN) FIXED BIN;           SEN00030
40      1  0  DCL (TYPE) CHAR(*) VAR;                         SEN00040
50      1  0  DCL (MPT,PT) PTR;                                SEN00050
                                           SEN00060
100010  1  0  DCL PT_ADDR PTR;                                PFA00010
                                           PFA00020
100030  1  0  DCL 1 PF_ADDR BASED (PT_ADDR),                  PFA00030
                    2 VPID FIXED BIN,                        PFA00040
                    2 BOXID FIXED BIN,                       PFA00050
                    2 PTR PTR;                                PFA00060
                                           PFA00070
                                           SEN00060
                                           SEN00070
300010  1  0  DCL PT_S PTR;                                    PFS00010
                                           PFS00020
300030  1  0  DCL 1 PF_S BASED (PT_S),                        PFS00030
                    2 LEN FIXED BIN,                         PFS00040
                    2 PTR PTR;                                PFS00050
                                           PFS00060
                                           SEN00070
                                           SEN00080
400010  1  0  DCL PT_MSG PTR;                                  PFM00010
                                           PFM00020
400030  1  0  DCL 1 PF_MSG BASED (PT_MSG),                    PFM00030
                    2 LEN FIXED BIN,                         PFM00040
                    2 TYPE CHAR (12) VAR,                    PFM00050
                    2 PTR PTR;                                PFM00060
                                           PFM00070
                                           SEN00080
                                           SEN00090
500010  1  0  DCL PT_LEVEL PTR;                                PFL00010
                                           PFL00020
500030  1  0  DCL 1 PF_LEVEL BASED (PT_LEVEL),                PFL00030
                    2 LEVEL FIXED BIN,                       PFL00040
                    2 PTR PTR;                                PFL00050
                                           PFL00060
                                           SEN00090
                                           SEN00100
600010  1  0  DCL THISVP PTR EXTERNAL STATIC; /* -> CURRENT VP, SET BY VPER */VPX00010
                                           VPX00020
600030  1  0  DCL 1 VP BASED (THISVP), XINCLUDE VP;          VPX00030
                    80 BDSA FIXED BIN (31), /* ADDR (BOTTOM DSA) */VP 00010
                    80 TDSA FIXED BIN (31), /* ADDR (TOP DSA) */VP 00020
                    80 SAVESIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED */VP 00030

```

NUMBER LEV NT

-101-

```

      80 SAFE PTR,                /* -> 1ST SAVBLK          */VP 00040
      80 PROCNAME CHAR (7) VAR,   /* NAME OF TOP LEVEL PROC */VP 00050
      80 STATUS CHAR (12) VAR,    VP 00060
      80 MAIL PTR,                /* CHAIN OF INCOMING MAIL */VP 00070
      80 WAIT,                    VP 00080
      81 BOX FIXED BIN,           /* BOX AWAITING MAIL      */VP 00090
      81 MSG PTR,                 /* MSG IN WAIT.BOX        */VP 00100
      80 VPID FIXED BIN,          /* INDEX IN THE VPST      */VP 00110
      80 LEVEL FIXED BIN,         /* LEVEL                   */VP 00120
      80 VTIME FIXED BIN (31)     VP 00130
      .....                      VPX00030
      .....                      VPX00040
      .....                      SEN00100
      .....                      SEN00110
      .....                      SEN00120
900010 1 0 XINCLUDE QEVENT;.....SEN00120
      DCL QEVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR, QEV00010
      FIXED BIN (31), PTR); QEV00020
      ..... SEN00120
      XINCLUDE STIMER;.....SEN00130
1000010 1 0 DCL STIMER ENTRY RETURNS (FIXED BIN (31)); STI00010
      ..... SEN00130
      XINCLUDE SVCER;.....SEN00140
1100010 1 0 DCL SVCER ENTRY (FIXED BIN, PTR); SVC00010
      ..... SEN00140
1100150 1 0 DCL SYNC ENTRY; SEN00150
      ..... SEN00160
1100170 1 0 IF (LEVEL = VP.LEVEL) & (VPID = 0) THEN DO; SEN00170
1100180 1 1 CALL SYNC; SEN00180
1100190 1 1 CALL SVCER (BOXID,MPT); SEN00190
1100200 1 1 RETURN; SEN00200
1100210 1 1 END; SEN00210
      ..... SEN00220
      ..... SEN00230
1100240 1 0 ALLOCATE PF_ADDR; SEN00240
1100250 1 0 PF_ADDR.BOXID = BOXID; SEN00250
1100260 1 0 PF_ADDR.VPID = VPID; SEN00260
1100270 1 0 PF_ADDR.PTR = MPT; SEN00270
      ..... SEN00280
1100290 1 0 IF (LEVEL = VP.LEVEL) THEN DO; SEN00290
1100300 1 1 CALL QEVENT (VP.LEVEL*3+3,STIMER,'LOGMSG',0,PT_ADDR); SEN00300
1100310 1 1 END; SEN00310
      ..... SEN00320
      ..... SEN00330
1100330 1 0 ELSE DO; /* LEVEL ^= VP.LEVEL */ SEN00330
1100340 1 1 PT = PT_ADDR; SEN00340
1100350 1 1 IF (TYPE = 'S') THEN DO; SEN00350
1100360 1 2 ALLOCATE PF_S; SEN00360
1100370 1 2 PF_S.LEN = LEN; SEN00370
1100380 1 2 PF_S.PTR = PT_ADDR; SEN00380
1100390 1 2 PT = PT_S; SEN00390

```

NUMBER LEV NT

1100400	1	2	END;	SEN00400
1100410	1	1	ALLOCATE PF_MSG;	SEN00410
1100420	1	1	PF_MSG.LEN = LEN+2;	SEN00420
1100430	1	1	PF_MSG.TYPE = TYPE;	SEN00430
1100440	1	1	PF_MSG.PTR = PT;	SEN00440
1100450	1	1	ALLOCATE PF_LEVEL;	SEN00450
1100460	1	1	PF_LEVEL.LEVEL = LEVEL;	SEN00460
1100470	1	1	PF_LEVEL.PTR = PT_MSG;	SEN00470
1100480	1	1	CALL QEVENT (VP.LEVEL+3+1,STIMER,'MSGOUT',0,PT_LEVEL);	SEN00480
1100490	1	1	END;	SEN00490
1100510	1	0	END SEND;	SEN00500
				SEN00510

SOURCE LISTING

NUMBER LEV NT

R

```

10      0 SHELL: PROC OPTIONS (MAIN);
30      1 0 DCL PLIXOPT CHAR(40) VAR EXTERNAL STATIC
          INIT ('ISASIZE(-50K)');

100010  1 0 DCL SQUEUE (0:      6) PTR EXTERNAL STATIC;
100020  1 0 DCL EQUEUE (0:      6) PTR EXTERNAL STATIC;
100080  1 0 DCL 1 LOSES (0:      1) STATIC,
          70 LEVEL FIXED BIN,
          70 SRQ,
          71 SIZE FIXED BIN,
          71 MAX FIXED BIN,
          70 DBB,
          71 SIZE FIXED BIN,
          71 MAX FIXED BIN,
          70 VPS,
          71 TABLE (      20),
          80 BDSA FIXED BIN (31), /* ADDR (BOTTOM DSA) */
          80 TDSA FIXED BIN (31), /* ADDR (TOP DSA) */
          80 SAVESIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED */
          80 SAFE PTR, /* -> 1ST SAVBLK */
          80 PROCNAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC */
          80 STATUS CHAR (12) VAR, /* CHAIN OF INCOMING MAIL */
          80 MAIL PTR, /* BOX AWAITING MAIL */
          80 WAIT, /* MSG IN WAIT.BOX */
          81 BOX FIXED BIN, /* INDEX IN THE VPST */
          81 MSG PTR, /* LEVEL */
          80 VPID FIXED BIN,
          80 LEVEL FIXED BIN,
          80 VTIME FIXED BIN (31)

400090  1 0 DCL (EPT,SPT) PTR;
400100  1 0 DCL 1 EVENT BASED (EPT),
          70 NEXT PTR, /* NEXT EVENT */
          70 STIME FIXED BIN (31),
          70 TYPE CHAR (12) VAR,
          70 INDEX FIXED BIN (31),
          70 PTR PTR

500110  1 0 DCL 1 SVR BASED (SPT),
          70 NEXT PTR,
          70 STIME FIXED BIN (31),
          70 STIMEQ FIXED BIN (31), /* ACCUMULATED STIME */
          70 RTIME FIXED BIN (31), /* REAL CPU TICK RATE */
          70 RRATE FIXED BIN (15,7),
          70 ID FIXED BIN

```

```

SHE00010
SHE00020
SHE00030
SHE00040
SHE00050
QS 00010 1
QS 00020 1
SHE00080 1
LOS00010
LOS00020
LOS00030
LOS00040
LOS00050
LOS00060
LOS00070
LOS00080
LOS00090 1
*/VP 00010
*/VP 00020
*/VP 00030
*/VP 00040
*/VP 00050
VP 00060
*/VP 00070
VP 00080
*/VP 00090
*/VP 00100
*/VP 00110
*/VP 00120
VP 00130
SHE00080
SHE00090
SHE00100
*/EVE00010
EVE00020
EVE00030
EVE00040
EVE00050
SHE00100
SHE00110
SVR00010
SVR00020
*/SVR00030
SVR00040
*/SVR00050
SVR00060

```

NUMBER LEV NT

R

600120	1	0	DCL (I,LEVEL,EARLIEST) FIXED BIN;	SHE00110
600130	1	0	DCL (STM,STMO) FIXED BIN (31);	SHE00120
600140	1	0	DCL DAT CHAR(6);	SHE00130
600150	1	0	DCL TIM CHAR(9);	SHE00140
600160	1	0	DCL (YR,MO,DY,HR,MN) CHAR(2);	SHE00150
600170	1	0	DCL STATS FILE OUTPUT EXTERNAL;	SHE00160
700010	1	0	DCL 1 DEBUG EXTERNAL STATIC,	SHE00170
			2 SLEEPS BIT(1) INIT ('0'B),	DEB00010
			2 SAHERS BIT(1) INIT ('0'B),	DEB00020
			2 SCHEDULERS BIT(1) INIT ('0'B),	DEB00030
			2 SHELLS BIT(1) INIT ('0'B),	DEB00040
			2 AAHERS BIT(1) INIT ('0'B);	DEB00050
				DEB00060
				SHE00190
700200	1	0	DCL (ADDR,NULL,MAX,MOD,DATE,TIME,SUBSTR) BUILTIN;	SHE00200
700210	1	0	DCL (ADDQ,DEBUGR) ENTRY;	SHE00210
800010	1	0	DCL RTIMER ENTRY RETURNS (FIXED BIN (31));	RT100010
900010	1	0	DCL QEVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR,	QEV00010
			FIXED BIN (31), PTR);	QEV00020
900240	1	0	DCL (GBER,GCER,SAHER,AAHER) ENTRY;	SHE00240
				SHE00250
				SHE00260
900270	1	0	OPEN FILE (STATS) TITLE ('STATS');	SHE00270
900280	1	0	DAT = DATE;	SHE00280
900290	1	0	TIM = TIME;	SHE00290
900300	1	0	YR = SUBSTR(DAT,1,2);	SHE00300
900310	1	0	MO = SUBSTR(DAT,3,2);	SHE00310
900320	1	0	DY = SUBSTR(DAT,5,2);	SHE00320
900330	1	0	HR = SUBSTR(TIM,1,2);	SHE00330
900340	1	0	MN = SUBSTR(TIM,3,2);	SHE00340
900350	1	0	PUT FILE (STATS) SKIP EDIT (' *** 'YR','/',MO,'/',DY,' ',	SHE00350
			HR,' ',MN)	SHE00360
			(A,A(2),A,A(2),A,A(2),A,A(2),A,A(2));	SHE00370
900380	1	0	PUT FILE (STATS) SKIP EDIT (' *** PARAMETERS: ')(A);	SHE00380
900390	1	0	PUT FILE (STATS) SKIP (2);	SHE00390
900400	1	0	CALL DEBUGR;	SHE00400
900410	1	0	CALL SET_LOS;	SHE00410
900420	1	0	CALL SET_O;	SHE00420
900430	1	0	CALL SET_P;	SHE00430
900440	1	0	PUT SKIP (3) LIST (' START SIMULATION...');	SHE00440
900450	1	0	PUT SKIP (3);	SHE00450
900460	1	0	PUT FILE (STATS) SKIP (3) EDIT (' *** SIMULATION:')(A);	SHE00460
900470	1	0	PUT FILE (STATS) SKIP (2);	SHE00470
				SHE00480
900490	1	0	LOOP:	SHE00490
				SHE00500
			EARLIEST = -1;	SHE00510
900520	1	0	STMO = 2147483647;	SHE00520
				SHE00530

PL/I OPTIMIZING COMPILER

SHELL: PROC OPTIONS (MAIN);

NUMBER LEV NT

```

900540 1 0      DO I = 0 TO 6;
900550 1 1      IF (EQUEUE(I) ^= NULL) THEN DO;
900560 1 2          EPT = EQUEUE(I);
900570 1 2          SPT = SQUEUE(I);
900580 1 2          STM = MAX (EVENT.STIME,SVR.STIME);
900590 1 2          IF (STM < STMO) THEN DO;
900600 1 3              EARLIEST = I;
900610 1 3              STMO = STM;
900620 1 3          END;
900630 1 2      END;
900640 1 1      END;

900660 1 0      IF (DEBUG.SHELL$) THEN DO;
900670 1 1          PUT SKIP EDIT (' SHELL: EARLIEST = ',EARLIEST) (A,F(2));
900680 1 1      END;
900690 1 0      IF (EARLIEST = -1) THEN GOTO EXIT;

900710 1 0      EPT = EQUEUE(EARLIEST);
900720 1 0      SPT = SQUEUE(EARLIEST);
900730 1 0      EQUEUE(EARLIEST) = EVENT.NEXT;
900740 1 0      LEVEL = (EARLIEST-1)/3;
900750 1 0      SVR.STIME = STMO;
900760 1 0      SVR.RTIME = RTIMER;

900780 1 0      IF (EARLIEST = 0) THEN DO;
900790 1 1          CALL GBER (SVR,EVENT);
900800 1 1      END;
900810 1 0      ELSE IF (MOD(EARLIEST,3) = 1) THEN DO; /* GC */
900820 1 1          CALL GCER (LOSES(LEVEL),SVR,EVENT);
900830 1 1      END;
900840 1 0      ELSE IF (MOD(EARLIEST,3) = 2) THEN DO; /* AAK */
900850 1 1          CALL AAKER (LOSES(LEVEL),SVR,EVENT);
900860 1 1      END;
900870 1 0      ELSE DO; /* SAH */
900880 1 1          CALL SAHER (LOSES(LEVEL),SVR,EVENT);
900890 1 1      END;

900910 1 0      FREE EVENT;
900920 1 0      IF (EARLIEST ^= 0) & (MOD(EARLIEST,3) ^= 1) THEN DO;
900930 1 1          SQUEUE(EARLIEST) = SVR.NEXT;
900940 1 1          SVR.NEXT = NULL;
900950 1 1          CALL ADDQ (SQUEUE(EARLIEST),SPT);
900960 1 1      END;

900980 1 0      GOTO LOOP;

901000 1 0      EXIT;

      PUT FILE (STATS) SKIP (3) EDIT (' *** STATISTICS!') (A);

```

```

SHE00540 1
SHE00550
SHE00560
SHE00570
SHE00580
SHE00590
SHE00600
SHE00610
SHE00620
SHE00630
SHE00640
SHE00650
SHE00660
SHE00670
SHE00680
SHE00690
SHE00700
SHE00710
SHE00720
SHE00730
SHE00740
SHE00750
SHE00760
SHE00770
SHE00780
SHE00790
SHE00800
SHE00810
SHE00820
SHE00830
SHE00840
SHE00850
SHE00860
SHE00870
SHE00880
SHE00890
SHE00900
SHE00910
SHE00920
SHE00930
SHE00940
SHE00950
SHE00960
SHE00970
SHE00980
SHE00990
SHE01000
SHE01010
SHE01020

```

PL/I OPTIMIZING COMPILER

SHELL: PROC OPTIONS (MAIN);

NUMBER LEV NT

```

901030 1 0 PUT FILE (STATS) SKIP (2);
901040 1 0 CALL REPORT;
901050 1 0 CLOSE FILE (STATS);
901060 1 0 RETURN;

```

```

901090 1 0 SET_LOS: PROC;

```

```

901110 2 0 DCL (L,IV) FIXED BIN;

```

```

901130 2 0 DO L = 0 TO 1;
901140 2 1 LOSES(L).LEVEL = L;
901150 2 1 LOSES(L).SRQ.SIZE = 0;
901160 2 1 LOSES(L).SRQ.MAX = 0;
901170 2 1 LOSES(L).DBB.SIZE = 0;
901180 2 1 LOSES(L).DBB.MAX = 0;
901190 2 1 DO IV = 1 TO 20;
901200 2 2 LOSES(L).VPS.TABLE(IV).LEVEL = L;
901210 2 2 LOSES(L).VPS.TABLE(IV).VPID = IV;
901220 2 2 LOSES(L).VPS.TABLE(IV).STATUS = 'VOID';
901230 2 2 END;
901240 2 1 END;

```

```

901260 2 0 END SET_LOS;

```

```

901290 1 0 SET_Q: PROC;

```

```

901310 2 0 DCL (IQ,L,NP,IP) FIXED BIN;
901320 2 0 DCL (RRATE) FIXED BIN (15,7);

```

```

901340 2 0 DO IQ = 0 TO 6;
901350 2 1 SQUEUE(IQ) = NULL;
901360 2 1 SQUEUE(IQ) = NULL;
901370 2 1 END;

```

```

901390 2 0 DO L = 0 TO 1;
901400 2 1 PUT SKIP EDIT (' LEVEL ',L,' ') (A,F(2),A);
901410 2 1 PUT FILE (STATS) SKIP EDIT (' LEVEL ',L,' ') (A,F(2),A);
901420 2 1 PUT SKIP LIST (' NO. OF PROCESSORS, RRATE> ');
901430 2 1 GET LIST (NP,RRATE);
901440 2 1 PUT FILE (STATS) DATA (NP,RRATE);
901450 2 1 DO IP = 1 TO NP;
901460 2 2 CALL NEW_SVR(RRATE,IP,SQUEUE(L*3+2));
901470 2 2 CALL NEW_SVR(RRATE,IP,SQUEUE(L*3+3));
901480 2 2 END;
901490 2 1 CALL NEW_SVR(RRATE,1,SQUEUE(L*3+1));
901500 2 1 END; /* L */

```

```

SHE01030
SHE01040
SHE01050
SHE01060
SHE01070
SHE01080
SHE01090
SHE01100
SHE01110
SHE01120
SHE01130 1
SHE01140
SHE01150
SHE01160
SHE01170
SHE01180
SHE01190 1
SHE01200
SHE01210
SHE01220
SHE01230
SHE01240
SHE01250
SHE01260
SHE01270
SHE01280
SHE01290
SHE01300
SHE01310
SHE01320
SHE01330
SHE01340 1
SHE01350
SHE01360
SHE01370
SHE01380
SHE01390 1
SHE01400
SHE01410
SHE01420
SHE01430
SHE01440
SHE01450
SHE01460
SHE01470
SHE01480
SHE01490
SHE01500
SHE01510

```

NUMBER LEV NT

```

901520 2 0 CALL NEW_SVR(1,1,SQUEUE(0));
901530 2 0 CALL QEVENT (3,0,'INIT',0,NULL);
901540 2 0 RETURN;
          /* END */

```

```

901580 2 0 NEW_SVR: PROC (RRATE,ID,WHERE);

```

```

901600 3 0 DCL RRATE FIXED BIN (15,7);
901610 3 0 DCL ID FIXED BIN;
901620 3 0 DCL WHERE PTR;
901630 3 0 DCL SVRPT PTR;
901640 3 0 DCL 1 SVR BASED (SVRPT),
          70 NEXT PTR,
          70 STIME FIXED BIN (31),
          70 STIMEQ FIXED BIN (31),
          70 RTIME FIXED BIN (31),
          70 RRATE FIXED BIN (15,7),
          70 ID FIXED BIN
          ;
          /* ACCUMULATED STIME
          /* REAL CPU TICK RATE

```

```

1001660 3 0 ALLOCATE SVR;
1001670 3 0 SVR.STIME = 0;
1001680 3 0 SVR.STIMEQ = 0;
1001690 3 0 SVR.RRATE = RRATE;
1001700 3 0 SVR.ID = ID;
1001710 3 0 SVR.NEXT = WHERE;
1001720 3 0 WHERE = SVRPT;

```

```

1001740 3 0 END NEW_SVR;

```

```

1001770 2 0 END SET_Q;

```

```

1001800 1 0 SET_P: PROC;

```

```

1100010 2 0 DCL 1 PARAMS EXTERNAL STATIC,
          2 THRU_RATE FIXED BIN (31,7),
          2 DELAY_GB_GC FIXED BIN (31),
          2 DELAY_GC_GB FIXED BIN (31),
          2 TERMINALS FIXED BIN;

```

```

1101840 2 0 PUT SKIP LIST (' THRU_RATE,GB_GC,GC_GB,TERMINALS> ');
1101850 2 0 GET LIST (PARAMS.THRU_RATE,PARAMS.DELAY_GB_GC,
          PARAMS.DELAY_GC_GB,PARAMS.TERMINALS);
1101870 2 0 PUT FILE (STATS) SKIP EDIT (' ') (A);
1101880 2 0 PUT FILE (STATS) DATA (PARAMS);

```

```

SHE01520
SHE01530
SHE01540
SHE01550
SHE01560
SHE01570
SHE01580
SHE01590
SHE01600
SHE01610
SHE01620
SHE01630
SHE01640
SVR00010
SVR00020
/*SVR00030
SVR00040
/*SVR00050
SVR00060
SHE01640
SHE01650
SHE01660
SHE01670
SHE01680
SHE01690
SHE01700
SHE01710
SHE01720
SHE01730
SHE01740
SHE01750
SHE01760
SHE01770
SHE01780
SHE01790
SHE01800
SHE01810
PAR00010
PAR00020
PAR00030
PAR00040
PAR00050
SHE01830
SHE01840
SHE01850
SHE01860
SHE01870
SHE01880
SHE01890

```

PL/I OPTIMIZING COMPILER

SHELL: PROC OPTIONS (MAIN);

NUMBER LEV NT

1101900 2 0 END SET_P;

1101930 1 0 REPORT: PROC;

1101950 2 0 DCL SVRPT PTR;

1101960 2 0 DCL 1 SVR BASED (SVRPT);

70 NEXT PTR,

70 STIME FIXED BIN (31),

70 STIMEQ FIXED BIN (31),

70 RTIME FIXED BIN (31),

70 RRATE FIXED BIN (15,7),

70 ID FIXED BIN

/* ACCUMULATED STIME

/* REAL CPU TICK RATE

1201970 2 0 DCL (I,L) FIXED BIN;

1201980 2 0 DCL TITLE(3) CHAR(3) STATIC INIT (' GC','AAH','SAH');

1202000 2 0 DO L = 0 TO 1;

1202020 2 1 PUT FILE (STATS) SKIP EDIT (' LEVEL ',L,' ') (A,F(2),A);

1202030 2 1 PUT FILE (STATS) SKIP EDIT ('

' SRQ.MAX =',LOSES(L).SRQ.MAX,

' DBB.MAX =',LOSES(L).DBB.MAX)

(A,A,F(11),A,F(11));

1202080 2 1 DO I = 1 TO 3;

1202090 2 2 PUT FILE (STATS) SKIP EDIT (' ',TITLE(I),' ') (A,A(3),A);

1202100 2 2 SVRPT = SQUEUE(L*3+1);

1202110 2 2 DO WHILE (SVRPT ^= NULL);

1202120 2 3 PUT FILE (STATS) SKIP EDIT (' ') (A);

1202130 2 3 PUT FILE (STATS) DATA (SVR.ID,SVR.STIME,SVR.STIMEQ);

1202140 2 3 SVRPT = SVR.NEXT;

1202150 2 3 END;

1202160 2 2 END;

1202180 2 1 END; /* L */

1202200 2 0 END REPORT;

1202220 1 0 END SHELL;

SHE01900

SHE01910

SHE01920

SHE01930

SHE01940

SHE01950

SHE01960

SVR00010

SVR00020

*/SVR00030

SVR00040

*/SVR00050

SVR00060

SHE01960

SHE01970

SHE01980

SHE01990

SHE02000

SHE02010

SHE02020

SHE02030

SHE02040

SHE02050

SHE02060

SHE02070

SHE02080

SHE02090

SHE02100

SHE02110

SHE02120

SHE02130

SHE02140

SHE02150

SHE02160

SHE02170

SHE02180

SHE02190

SHE02200

SHE02210

SHE02220

SOURCE LISTING

NUMBER LEV NT

R

```

10      0 SLEEP: PROC;                                SLE00010
30      1 0 DCL (THIS,NEXT) PTR;                        SLE00020
40      1 0 DCL (NULL,ADDR,MIN) BUILTIN;               SLE00030
50      1 0 DCL (NSIZE,OSIZE,NEW_NBLK,OLD_NBLK,DIFF_NBLK) FIXED BIN (31); SLE00040
60      1 0 DCL SAVE_ADDR FIXED BIN (31);              SLE00050
100010  1 0 DCL THISVP PTR EXTERNAL STATIC;             SLE00060
/* -> CURRENT VP, SET BY VPER */VPX00010
100030  1 0 DCL : VP BASED (THISVP),                    VPX00020
      80 BDSA FIXED BIN (31),                          /* ADDR (BOTTOM DSA) */VP 00010
      80 TDSA FIXED BIN (31),                          /* ADDR (TOP DSA) */VP 00020
      80 SAVESIZE FIXED BIN (31),                      /* SIZE OF AREA TO BE SAVED */VP 00030
      80 SAFE PTR,                                     /* -> 1ST SAVBLK */VP 00040
      80 PROCNAME CHAR (7) VAR,                       /* NAME OF TOP LEVEL PROC */VP 00050
      80 STATUS CHAR (12) VAR,                        /* VP 00060
      80 MAIL PTR,                                     /* CHAIN OF INCOMING MAIL */VP 00070
      80 WAIT,                                         /* VP 00080
      81 BOX FIXED BIN,                               /* BOX AWAITING MAIL */VP 00090
      81 MSG PTR,                                     /* MSG IN WAIT.BOX */VP 00100
      80 VPID FIXED BIN,                             /* INDEX IN THE VPST */VP 00110
      80 LEVEL FIXED BIN,                             /* LEVEL */VP 00120
      80 VTIME FIXED BIN (31)                         /* VP 00130
      ;                                                VPX00030
400070  1 0 DCL NEWSAVBLK PTR;                        SAV00030
400090  1 0 DCL 1 SAVBLK BASED (NEWSAVBLK),            SAV00080
      2 NEXT PTR,                                     /* NEXT SAVED ITEM */SAV00090
      2 ADDR FIXED BIN (31),                          /* BEG ADDR OF SAVED ITEM */SAV00100
      2 SIZE FIXED BIN (31),                          /* SIZE OF SAVED ITEM */SAV00110
      2 AREA ( 250) FIXED BIN (31);                  SAV00120
500010  1 0 DCL 1 DEBUG EXTERNAL STATIC,              SAV00130
      2 SLEEPS BIT(1) INIT ('0'B),                   SAV00140
      2 SAHERS BIT(1) INIT ('0'B),                   DEB00010
      2 SCHEDULERS BIT(1) INIT ('0'B),               DEB00020
      2 SHELLS BIT(1) INIT ('0'B),                   DEB00030
      2 AAHERS BIT(1) INIT ('0'B);                   DEB00040
600010  1 0 DCL DSA FIXED BIN (31,0),                 DEB00050
      NAB FIXED BIN (31,0),                           DEB00060
      SEG FIXED BIN (31,0),                           SLE00100
      EOS FIXED BIN (31,0);                           GET00010
                                                    GET00020
                                                    GET00030
                                                    GET00040
                                                    GET00050
                                                    GET00060

```

NUMBER LEV NT

R

```

500070 1 0 DCL GET4 ENTRY (FIXED BIN (31), FIXED BIN (31),
      FIXED BIN (31), FIXED BIN (31))
      OPTIONS (ASSEMBLER, INTER);
700010 1 0 DCL SVSTK ENTRY (PTR, FIXED BIN (31))
      OPTIONS (ASSEMBLER, INTER);

      /* ARG1: PTR -> SAVBLK
      ARG2: ADDR OF RETURN DSA (BOTTOM DSA)
      */

      /* VP.STATUS = 'RUNNABLE' DEBUG */

700160 1 0 CALL GET4 (DSA,NAB,SEG,EDS);
700170 1 0 IF SEG = 255 THEN DO;
700180 1 1 PUT SKIP LIST (' ERROR: ILLEGAL SEGNO. (SLEEP)');
700190 1 1 PUT DATA (SEG);
700200 1 1 STDP;
700210 1 1 END;

700240 1 0 NSIZE = NAB - VP.BDSA;
700250 1 0 OSIZE = VP.SAVESIZE;
700260 1 0 NEW_NBLK = (NSIZE + 1000 - 1) / 1000;
700270 1 0 OLD_NBLK = (OSIZE + 1000 - 1) / 1000;
700280 1 0 DIFF_NBLK = NEW_NBLK - OLD_NBLK;
700290 1 0 IF (DEBUG.SLEEPS) THEN DO;
700300 1 1 PUT SKIP LIST (' SLEEP: ');
700310 1 1 PUT DATA (NSIZE,OSIZE,NEW_NBLK,OLD_NBLK,DIFF_NBLK);
700320 1 1 END;

700340 1 0 IF DIFF_NBLK > 0 THEN DO;
700350 1 1 NEXT = VP.SAFE;
700360 1 1 DO WHILE (DIFF_NBLK > 0);
700370 1 2 ALLOCATE SAVBLK SET (THIS);
700380 1 2 THIS -> SAVBLK.NEXT = NEXT;
700390 1 2 NEXT = THIS;
700400 1 2 DIFF_NBLK = DIFF_NBLK - 1;
700410 1 2 END;
700420 1 1 VP.SAFE = THIS;
700430 1 1 END;
700440 1 0 ELSE IF DIFF_NBLK < 0 THEN DO;
700450 1 1 THIS = VP.SAFE;
700460 1 1 DO WHILE (DIFF_NBLK < 0);
700470 1 2 NEXT = THIS -> SAVBLK.NEXT;
700480 1 2 FREE THIS -> SAVBLK;
700490 1 2 THIS = NEXT;
700500 1 2 DIFF_NBLK = DIFF_NBLK + 1;
700510 1 2 END;
700520 1 1 VP.SAFE = THIS;

```

```

GET00070
GET00080
GET00090
SVS00010
SVS00020
SVS00030
SVS00040
SVS00050
SVS00060
SLE00130
SLE00140
SLE00150
SLE00160
SLE00170
SLE00180
SLE00190
SLE00200
SLE00210
SLE00220
SLE00230
SLE00240
SLE00250
SLE00260 1
SLE00270 1
SLE00280
SLE00290
SLE00300
SLE00310
SLE00320
SLE00330
SLE00340
SLE00350
SLE00360
SLE00370
SLE00380
SLE00390
SLE00400
SLE00410
SLE00420
SLE00430
SLE00440
SLE00450
SLE00460
SLE00470
SLE00480
SLE00490
SLE00500
SLE00510
SLE00520

```

NUMBER LEV NT

R

700530	1	1	END;	SLE00530
700550	1	0	VP.SAVESIZE = NSIZE;	SLE00540
700560	1	0	VP.TDSA = DSA;	SLE00550
				SLE00560
				SLE00570
700580	1	0	THIS = VP.SAFE;	SLE00580
700590	1	0	SAVE_ADDR = VP.BDSA;	SLE00590
700600	1	0	DO WHILE (THIS ^= NULL);	SLE00600
700610	1	1	THIS -> SAVBLK.SIZE = MIN(1000,NSIZE);	SLE00610 1
700620	1	1	THIS -> SAVBLK.ADDR = SAVE_ADDR;	SLE00620
700630	1	1	NSIZE = NSIZE - 1000;	SLE00630 1
700640	1	1	SAVE_ADDR = SAVE_ADDR + 1000;	SLE00640 1
700650	1	1	THIS = THIS -> SAVBLK.NEXT;	SLE00650
700660	1	1	END;	SLE00660
				SLE00670
700680	1	0	CALL SVSTK (VP.SAFE,VP.BDSA);	SLE00680
			/* CONTROL SHOULD COME HERE AFTER RTSTK */	SLE00690
				SLE00700
700720	1	0	END SLEEP;	SLE00710
				SLE00720

SOURCE LISTING

NUMBER LEV NT

```

10      0 STIMER: PROC RETURNS (FIXED BIN(31));                                ST100010
                                                ST100020
XINCLUDE SVRX;*****ST100030
100010  1 0 DCL THISSVR PTR EXTERNAL STATIC; /* -> CURRENT SVR, SET BY SAHER */ SVR00010
                                                SVR00020
100030  1 0 DCL 1 SVR BASED (THISSVR), XINCLUDE SVR;*****SVR00030
              70 NEXT PTR,                                                SVR00010
              70 STIME FIXED BIN (31),                                     SVR00020
              70 STIMEQ FIXED BIN (31),                                     /* ACCUMULATED STIME */SVR00030
              70 RTIME FIXED BIN (31),                                     SVR00040
              70 RRATE FIXED BIN (15,7),                                  /* REAL CPU TICK RATE */SVR00050
              70 ID FIXED BIN                                             SVR00060
              ***** ;                                                  SVR00030
              *****                                                    SVR00040
400040  1 0 DCL RT FIXED BIN (31);                                           ST100030
XINCLUDE RTIMER;*****ST100040
500010  1 0 DCL RTIMER ENTRY RETURNS (FIXED BIN (31));                      ST100050
              *****                                                    RT100010
              *****                                                    ST100050
500070  1 0      RT = RTIMER - SVR.RTIME;                                    ST100060
500080  1 0      RT = RT + SVR.RRATE;                                        ST100070
500090  1 0      RETURN (RT + SVR.STIME);                                    ST100080
                                                ST100090
500110  1 0 END;                                                            ST100100
                                                ST100110

```

SOURCE LISTING

NUMBER LEV NT

R

```

10      0  SVCER: PROC (SVC,PT);                                SVC00010
30      1  0  DCL SVC FIXED BIN;                                SVC00020
40      1  0  DCL PT PTR;                                       SVC00030
100010  1  0  DCL THISVP PTR EXTERNAL STATIC; /* -> CURRENT VP, SET BY VPER */VPX00010
100030  1  0  DCL 1 VP BASED (THISVP),                          VPX00020
      80 BDSA FIXED BIN (31), /* ADDR (BOTTOM DSA) */VP 00010
      80 TOSA FIXED BIN (31), /* ADDR (TOP DSA) */VP 00020
      80 SAVESIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED */VP 00030
      80 SAFE PTR, /* -> 1ST SAVBLK */VP 00040
      80 PROCNAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC */VP 00050
      80 STATUS CHAR (12) VAR, /* CHAIN OF INCOMING MAIL */VP 00060
      80 MAIL PTR, /* CHAIN OF INCOMING MAIL */VP 00070
      80 WAIT, /* CHAIN OF INCOMING MAIL */VP 00080
      81 BOX FIXED BIN, /* BOX AWAITING MAIL */VP 00090
      81 MSG PTR, /* MSG IN WAIT.BOX */VP 00100
      80 VPID FIXED BIN, /* INDEX IN THE VPST */VP 00110
      80 LEVEL FIXED BIN, /* LEVEL */VP 00120
      80 VTIME FIXED BIN (31) /* LEVEL */VP 00130
      ; /* -> CURRENT LOS, ST BY SAHER */LOS00010
400010  1  0  DCL THISLOS PTR EXTERNAL STATIC; /* -> CURRENT LOS, ST BY SAHER */LOS00020
400030  1  0  DCL 1 LOS BASED (THISLOS),                          LOS00030
      70 LEVEL FIXED BIN, /* LEVEL */LOS00010
      70 SRO, /* LEVEL */LOS00020
      71 SIZE FIXED BIN, /* LEVEL */LOS00030
      71 MAX FIXED BIN, /* LEVEL */LOS00040
      70 DBB, /* LEVEL */LOS00050
      71 SIZE FIXED BIN, /* LEVEL */LOS00060
      71 MAX FIXED BIN, /* LEVEL */LOS00070
      70 VPS, /* LEVEL */LOS00080
      71 TABLE ( 20), /* LEVEL */LOS00090
      80 BDSA FIXED BIN (31), /* ADDR (BOTTOM DSA) */VP 00010
      80 TOSA FIXED BIN (31), /* ADDR (TOP DSA) */VP 00020
      80 SAVESIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED */VP 00030
      80 SAFE PTR, /* -> 1ST SAVBLK */VP 00040
      80 PROCNAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC */VP 00050
      80 STATUS CHAR (12) VAR, /* CHAIN OF INCOMING MAIL */VP 00060
      80 MAIL PTR, /* CHAIN OF INCOMING MAIL */VP 00070
      80 WAIT, /* CHAIN OF INCOMING MAIL */VP 00080
      81 BOX FIXED BIN, /* BOX AWAITING MAIL */VP 00090
      81 MSG PTR, /* MSG IN WAIT.BOX */VP 00100
      80 VPID FIXED BIN, /* INDEX IN THE VPST */VP 00110
      80 LEVEL FIXED BIN, /* LEVEL */VP 00120

```

NUMBER LEV NT

```

      80 VTIME FIXED BIN (31)
      800010 1 0 DCL THISSVR PTR EXTERNAL STATIC; /* -> CURRENT SVR, SET BY SAHER */
      800030 1 0 DCL 1 SVR BASED (THISSVR),
        70 NEXT PTR,
        70 STIME FIXED BIN (31),
        70 STIMEQ FIXED BIN (31), /* ACCUMULATED $TIME
        70 RTIME FIXED BIN (31), /* REAL CPU TICK RATE
        70 RRATE FIXED BIN (15,7),
        70 ID FIXED BIN
      1000100 1 0 DCL SVCS (1) LABEL INIT (START);
      1000120 1 0 DCL IV FIXED BIN;
      1100010 1 0 DCL PT_SVC PTR;
      1100030 1 0 DCL 1 PF_SVC BASED (PT_SVC),
        2 SVC CHAR (7) VAR,
        2 PTR PTR;
      1200010 1 0 DCL VPSTART ENTRY (/* VP */, CHAR(*) VAR, PTR);
      1200160 1 0 IF (SVC <= 0) ; (SVC > 1) THEN DO;
      1200170 1 1 PUT SKIP LIST (' ERROR: ILLEGAL SVC. ');
      1200180 1 1 PUT DATA (SVC);
      1200190 1 1 STOP;
      1200200 1 1 END;
      1200220 1 0 GOTO SVCS(SVC);
      1200240 1 0 START: /* SVC = 1 */
        DO IV = 1 TO 20;
      1200270 1 1 IF LDS.VPS.TABLE(IV).STATUS = 'VOID' THEN GOTO START;
      1200280 1 1 END;
      1200290 1 0 PUT SKIP LIST (' ERROR: EXCEED MAX. VP. (SVCER)');
      1200300 1 0 STOP;
      1200320 1 0 START1:
        PT_SVC = PT;
      1200350 1 0 CALL VPSTART (LDS.VPS.TABLE(IV), PF_SVC.SVC, PF_SVC.PTR);
      1200360 1 0 FREE PF_SVC;
      1200370 1 0 RETURN;
      1200390 1 0 END;

```

```

VP 00130
LOS00030
SVR00010
SVR00020
SVR00030
SVR00010
SVR00020
*/SVR00030
SVR00040
*/SVR00050
SVR00060
SVR00030
SVC00090
SVC00100
SVC00110
SVC00120
PFS00010
PFS00020
PFS00030
PFS00040
PFS00050
VPS00010
SVC00150
SVC00160
SVC00170
SVC00180
SVC00190
SVC00200
SVC00210
SVC00220
SVC00230
SVC00240
SVC00250
SVC00260
SVC00270
SVC00280
SVC00290
SVC00300
SVC00310
SVC00320
SVC00330
SVC00340
SVC00350
SVC00360
SVC00370
SVC00380
SVC00390

```

-114-

SOURCE LISTING

NUMBER LEV NT

```

10      0 SYNC: PROC;
                                     SYN00010
                                     SYN00020
                                     SYN00030
100010  1 0 DCL THISVP PTR EXTERNAL STATIC; /* -> CURRENT VP, SET BY VPER */VPX00010
                                     VPX00020
100030  1 0 DCL 1 VP BASED (THISVP), %INCLUDE VP;*****VPX00030
      80 BDSA FIXED BIN (31), /* ADDR (BOTTOM DSA) */VP 00010
      80 TOSA FIXED BIN (31), /* ADDR (TOP DSA) */VP 00020
      80 SAVESIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED */VP 00030
      80 SAFE PTR, /* -> 1ST SAVBLK */VP 00040
      80 PROCNAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC */VP 00050
      80 STATUS CHAR (12) VAR, VP 00060
      80 MAIL PTR, /* CHAIN OF INCOMING MAIL */VP 00070
      80 WAIT, VP 00080
      81 BOX FIXED BIN, /* BOX AWAITING MAIL */VP 00090
      81 MSG PTR, /* MSG IN WAIT.BOX */VP 00100
      80 VPID FIXED BIN, /* INDEX IN THE VPST */VP 00110
      80 LEVEL FIXED BIN, /* LEVEL */VP 00120
      80 VTIME FIXED BIN (31) VP 00130
      ***** ; VPX00030
      ***** VPX00040
      ***** SYN00030
      ***** SYN00040
500010  1 0 %INCLUDE QEVENT;*****SYN00050
      DCL QEVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR,
      FIXED BIN (31), PTR); QEVD00010
      ***** QEVD00020
500060  1 0 DCL NULL BUILTIN; SYN00050
500070  1 0 DCL SLEEP ENTRY; SYN00060
      %INCLUDE STIMER;*****SYN00070
600010  1 0 DCL STIMER ENTRY RETURNS (FIXED BIN (31)); STI00010
      ***** SYN00080
      ***** SYN00090
600100  1 0 CALL QEVENT (VP.LEVEL+3+3,STIMER,'SYNC',VP.VPID,NULL); SYN00100
600110  1 0 CALL SLEEP; SYN00110
      ***** SYN00120
600130  1 0 END; SYN00130

```

SOURCE LISTING

NUMBER LEV NT

```

10      0  TERM: PROC;                                TER00010
                                                    TER00020
%INCLUDE USERS;.....TER00030
%INCLUDE STIMER;.....USE00010
200010  1  0  DCL STIMER ENTRY RETURNS (FIXED BIN (31)); ST100010
%INCLUDE STIMER;.....USE00010
300020  1  0  DCL SEND ENTRY (FIXED BIN, FIXED BIN, FIXED BIN, CHAR(*) VAR, USE00020
                        FIXED BIN, PTR);                USE00030
300040  1  0  DCL WAIT ENTRY (FIXED BIN);                USE00040
300050  1  0  DCL SYNC ENTRY;                          USE00050
300060  1  0  DCL FINISH ENTRY;                        USE00060
                                                    USE00070
%INCLUDE STIMER;.....TER00030
400040  1  0  DCL (NULL,LENGTH) BUILTIN;                TER00040
                                                    TER00050
%INCLUDE VPX;.....TER00060
500010  1  0  DCL THISVP PTR EXTERNAL STATIC; /* -> CURRENT VP, SET BY VPER */VPX00010
                                                    VPX00020
500030  1  0  DCL 1 VP BASED (THISVP), %INCLUDE VP;.....VPX00030
                        80 BDSA FIXED BIN (31),          /* ADDR (BOTTOM DSA) */VP 00010
                        80 TDSA FIXED BIN (31),          /* ADDR (TOP DSA) */VP 00020
                        80 SAVESIZE FIXED BIN (31),      /* SIZE OF AREA TO BE SAVED */VP 00030
                        80 SAFE PTR,                    /* -> 1ST SAVBLK */VP 00040
                        80 PROCNAME CHAR (7) VAR,        /* NAME OF TOP LEVEL PROC */VP 00050
                        80 STATUS CHAR (12) VAR,         VP 00060
                        80 MAIL PTR,                    /* CHAIN OF INCOMING MAIL */VP 00070
                        80 WAIT,                        VP 00080
                        81 BOX FIXED BIN,                /* BOX AWAITING MAIL */VP 00090
                        81 MSG PTR,                      /* MSG IN WAIT.BOX */VP 00100
                        80 VPID FIXED BIN,               /* INDEX IN THE VPST */VP 00110
                        80 LEVEL FIXED BIN,              /* LEVEL */VP 00120
                        80 VTIME FIXED BIN (31)          VP 00130
%INCLUDE VPX;.....VPX00030
%INCLUDE VPX;.....VPX00040
%INCLUDE VPX;.....TER00060
%INCLUDE SVRX;.....TER00070
800010  1  0  DCL THISSVR PTR EXTERNAL STATIC; /* -> CURRENT SVR, SET BY SAHER */SVR00010
                                                    SVR00020
800030  1  0  DCL 1 SVR BASED (THISSVR), %INCLUDE SVR;.....SVR00030
                        70 NEXT PTR,                    SVR00010
                        70 STIME FIXED BIN (31),         SVR00020
                        70 STIMEQ FIXED BIN (31),        /* ACCUMULATED STIME */SVR00030
                        70 RTIME FIXED BIN (31),         SVR00040
                        70 RRATE FIXED BIN (15,7),       /* REAL CPU TICK RATE */SVR00050
                        70 ID FIXED BIN                  SVR00060
%INCLUDE SVRX;.....SVR00030

```

NUMBER LEV NT

```

*****
%INCLUDE PFSVC;*****
1100010 1 0 DCL PT_SVC PTR;
1100030 1 0 DCL 1 PF_SVC BASED (PT_SVC),
          2 SVC CHAR (7) VAR,
          2 PTR PTR;
*****
%INCLUDE MSG;*****
1200010 1 0 DCL MSGLEN FIXED BIN;
1200020 1 0 DCL MSGPT PTR;
1200030 1 0 DCL 1 MSG BASED (MSGPT),
          2 LEN FIXED BIN,
          2 STR CHAR (MSGLEN REFER (LEN));
*****
1200100 1 0 DCL (LEVEL,VPID,BOXID) FIXED BIN;
1200110 1 0 DCL MESSAGE CHAP(80) VAR;
1200120 1 0 DCL COMMAND CHAR(12);
1200130 1 0 DCL TYPE CHAR(8) VAR;
1200150 1 0 LOOP:
          PUT SKIP EDIT (' TERM: LEVEL = ',VP.LEVEL,' VP = ',VP.VPID,
          ' WAIT.BOX = ',VP.WAIT.BOX,' SVR = ',SVR.ID,' STIME = ',
          SVR.STIME) (A,F(2),A,F(2),A,F(2),A,F(2),A,F(11));
1200190 1 0 MSGPT = VP.WAIT.MSG;
1200200 1 0 IF (MSGPT ^= NULL) THEN DO;
1200210 1 1 PUT SKIP EDIT (' MSG: ',MSG.STR) (A,A(MSG.LEN));
1200220 1 1 FREE MSG;
1200230 1 1 VP.WAIT.MSG = NULL;
1200240 1 1 END;
1200260 1 0 WORK:
          PUT SKIP;
1200280 1 0 DISPLAY (' COMMAND?') REPLY (COMMAND);
1200300 1 0 IF (COMMAND = 'BUILD') THEN DO;
1200310 1 1 PUT SKIP LIST (' LEVEL> ');
1200320 1 1 GET LIST (LEVEL);
1200330 1 1 MSGLEN = 7;
1200340 1 1 ALLOCATE MSG;
1200350 1 1 MSG.STR = 'NEW VP!';
1200360 1 1 ALLOCATE PF_SVC;
1200370 1 1 PF_SVC.SVC = 'TERM';
1200380 1 1 PF_SVC.PTR = MSGPT;
1200390 1 1 CALL SEND (LEVEL,0,1,'S',15,PT_SVC);
1200400 1 1 GOTO WORK;
SVR00040
TER00070
TER00080
PFS00010
PFS00020
PFS00030
PFS00040
PFS00050
PFS00060
TER00080
TER00090
MSG00010
MSG00020
MSG00030
MSG00040
MSG00050
MSG00060
TER00090
TER00100
TER00110
TER00120
TER00130
TER00140
TER00150
TER00160
TER00170
TER00180
TER00190
TER00200
TER00210
TER00220
TER00230
TER00240
TER00250
TER00260
TER00270
TER00280
TER00290
TER00300
TER00310
TER00320
TER00330
TER00340
TER00350
TER00360
TER00370
TER00380
TER00390
TER00400

```

PL/I OPTIMIZING COMPILER

TERM: PROC;

NUMBER LEV NT

```

1200410 1 1      END;
1200430 1 0      ELSE IF (COMMAND = 'SEND') THEN DO;
1200440 1 1          PUT SKIP LIST (' LEVEL,VPID,BOXID,TYPE,MESSAGE ');
1200450 1 1          GET LIST (LEVEL,VPID,BOXID,TYPE,MESSAGE);
1200460 1 1          MSGLEN = LENGTH (MESSAGE);
1200470 1 1          ALLOCATE MSG;
1200480 1 1          MSG.STR = MESSAGE;
1200490 1 1          CALL SEND (LEVEL,VPID,BOXID,TYPE,MSGLEN,MSGPT);
1200500 1 1          GOTO WORK;
1200510 1 1      END;

1200530 1 0      ELSE IF (COMMAND = 'WAIT') THEN DO;
1200540 1 1          PUT SKIP LIST (' BOX> ');
1200550 1 1          GET LIST (BOXID);
1200560 1 1          PUT SKIP LIST (' WAITING');
1200570 1 1          CALL WAIT (BOXID);
1200580 1 1          GOTO LOOP;
1200590 1 1      END;

1200610 1 0      ELSE IF (COMMAND = 'SYNC') THEN DO;
1200620 1 1          PUT SKIP LIST (' SYNCING');
1200630 1 1          CALL SYNC;
1200640 1 1          GOTO LOOP;
1200650 1 1      END;

1200670 1 0      ELSE IF (COMMAND = 'FINISH') THEN DO;
1200680 1 1          PUT SKIP LIST (' FINISHING');
1200690 1 1          CALL FINISH;
1200700 1 1          RETURN;
1200710 1 1      END;

1200730 1 0      ELSE DO; /* UNKNOWN COMMAND */
1200740 1 1          PUT LIST (' ??');
1200750 1 1          GOTO WORK;
1200760 1 1      END;

1200780 1 0      END TERM;

```

```

TER00410
TER00420
TER00430
TER00440
TER00450
TER00460
TER00470
TER00480
TER00490
TER00500
TER00510
TER00520
TER00530
TER00540
TER00550
TER00560
TER00570
TER00580
TER00590
TER00600
TER00610
TER00620
TER00630
TER00640
TER00650
TER00660
TER00670
TER00680
TER00690
TER00700
TER00710
TER00720
TER00730
TER00740
TER00750
TER00760
TER00770
TER00780

```

SOURCE LISTING

NUMBER LEV NT

```

10      0 VPER: PROC (VP);                                VPE00010
30      1 0 DCL (THIS, NEXT) PTR;                          VPE00020
40      1 0 DCL THISVP PTR EXTERNAL STATIC;               VPE00030
50      1 0 DCL 1 VP,                                       VPE00040
      80 BDSA FIXED BIN (31), /* ADDR (BOTTOM DSA)          VPE00050
      80 TDSA FIXED BIN (31), /* ADDR (TOP DSA)             */VP 00010
      80 SAVESIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED */VP 00020
      80 SAFE PTR, /* -> 1ST SAVBLK */VP 00030
      80 PROCNAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC */VP 00040
      80 STATUS CHAR (12) VAR, /* CHAIN OF INCOMING MAIL */VP 00050
      80 MAIL PTR, /* CHAIN OF INCOMING MAIL */VP 00060
      80 WAIT, /* BOX AWAITING MAIL */VP 00070
      81 BOX FIXED BIN, /* MSG IN WAIT.BOX */VP 00080
      81 MSG PTR, /* INDEX IN THE VPST */VP 00090
      80 VPID FIXED BIN, /* LEVEL */VP 00100
      80 LEVEL FIXED BIN, /* LEVEL */VP 00110
      80 VTIME FIXED BIN (31) /* LEVEL */VP 00120
      VP 00130
      VPE00050
-119- 300010 1 0 DCL THISSVR PTR EXTERNAL STATIC; /* -> CURRENT SVR, SET BY SAHER */ SVR00010
      300030 1 0 DCL 1 SVR BASED (THISSVR), SVR00020
      70 NEXT PTR, SVR00030
      70 STIME FIXED BIN (31), /* ACCUMULATED STIME */SVR00010
      70 STIMEQ FIXED BIN (31), /* ACCUMULATED STIME */SVR00020
      70 RTIME FIXED BIN (31), /* REAL CPU TICK RATE */SVR00030
      70 RRATE FIXED BIN (15,7), /* REAL CPU TICK RATE */SVR00040
      70 ID FIXED BIN SVR00050
      SVR00060
      SVR00030
      BOX00010
      BOX00020
      BOX00030
      BOX00040
      BOX00050
      BOX00060
      SAV00030
      SAV00060
      SAV00070
      SAV00080
      SAV00090
      700070 1 0 DCL NEWSAVBLK PTR; /* NEXT SAVED ITEM */SAV00100
      700090 1 0 DCL 1 SAVBLK BASED (NEWSAVBLK), /* BEG ADDR OF SAVED ITEM */SAV00110
      2 NEXT PTR, /* SIZE OF SAVED ITEM */SAV00120
      2 ADDR FIXED BIN (31), /* SIZE OF SAVED ITEM */SAV00130
      2 SIZE FIXED BIN (31), /* SIZE OF SAVED ITEM */SAV00140
      2 AREA ( 250) FIXED BIN (31); SAV00090
      VPE00090

```

NUMBER LEV NT

```

900010 1 0 DCL EXECUTE ENTRY (CHAR(*) VAR);
1000010 1 0 DCL DSA FIXED BIN (31,0),
          NAB FIXED BIN (31,0),
          SEG FIXED BIN (31,0),
          EOS FIXED BIN (31,0);

1000070 1 0 DCL GET4 ENTRY (FIXED BIN (31), FIXED BIN (31),
          FIXED BIN (31), FIXED BIN (31))
          OPTIONS (ASSEMBLER, INTER);
1100010 1 0 DCL RTSTK ENTRY (PTR, FIXED BIN (31))
          OPTIONS (ASSEMBLER, INTER);

          /* ARG1: PTR -> SAVBLK
          ARG2: ADDR OF RETURN DSA (TOP DSA)
          */
1100130 1 0 DCL (ADDR,NULL) BUILTIN;
1200010 1 0 DCL STIMER ENTRY RETURNS (FIXED BIN (31));

1200170 1 0 THISVP = ADDR(VP);
1200190 1 0 IF VP.STATUS = 'NASCENT' THEN DO; /* STARTING A VP */
1200210 1 1 CALL GET4 (DSA,NAB,SEG,EOS);
1200220 1 1 IF SEG = 255 THEN DO;
1200230 1 2 PUT SKIP LIST (' ERROR: ILLEGAL SEGNO (VPER)');
1200240 1 2 PUT DATA (SEG);
1200250 1 2 STOP;
1200260 1 2 END;

1200280 1 1 VP.BOSA = DSA;
1200290 1 1 VP.STATUS = 'RUNNING'; /* SET TO RUNNING */
1200300 1 1 CALL EXECUTE (VP.PROCNAME);

          /*... SVSTK & RTSTK SHOULD RETURN HERE TOO ...*/
1200340 1 1 VP.VTIME = VP.VTIME + (STIMER - SVR.STIME);
1200350 1 1 IF VP.WAIT.BOX = -1 THEN DO; /* FINISHED */
1200360 1 2 THIS = VP.SAFE;
1200370 1 2 DO WHILE (THIS = NULL);
1200380 1 3 NEXT = THIS -> SAVBLK.NEXT;
1200390 1 3 FREE THIS -> SAVBLK;
1200400 1 3 THIS = NEXT;
1200410 1 3 END;
1200420 1 2 VP.SAFE = NULL;
1200430 1 2 THIS = VP.MAIL;
1200440 1 2 DO WHILE (THIS = NULL);
1200450 1 3 NEXT = THIS -> BOX.NEXT;

```

```

EXE00010
GET00010
GET00020
GET00030
GET00040
GET00050
GET00060
GET00070
GET00080
GET00090
RTS00010
RTS00020
RTS00030
RTS00040
RTS00050
RTS00060
VPE00130
STI00010
VPE00150
VPE00160
VPE00170
VPE00180
VPE00190
VPE00200
VPE00210
VPE00220
VPE00230
VPE00240
VPE00250
VPE00260
VPE00270
VPE00280
VPE00290
VPE00300
VPE00310
VPE00320
VPE00330
VPE00340
VPE00350
VPE00360
VPE00370
VPE00380
VPE00390
VPE00400
VPE00410
VPE00420
VPE00430
VPE00440
VPE00450

```

NUMBER LEV NT

R

```

1200460 1 3      FREE THIS -> BOX;
1200470 1 3      THIS = NEXT;
1200480 1 3      END;
1200490 1 2      VP.MAIL = NULL;
1200500 1 2      END; /* FREEING */

1200520 1 1      RETURN;
1200530 1 1      END;

1200550 1 0      ELSE DO; /* STATUS = RUNNABLE */

1200570 1 1          CALL GET4 (DSA,NAB,SEG,EOS);
1200580 1 1          IF SEG ^= 255 THEN DO;
1200590 1 2              PUT SKIP LIST (' ERROR: ILLEGAL SEGNO (VPER)');
1200600 1 2              PUT DATA (SEG);
1200610 1 2              STOP;
1200620 1 2          END;
1200630 1 1          IF (EOS-DSA) < (VP.SAVESIZE) THEN DO;
1200640 1 2              PUT SKIP LIST
                  (' ERROR: NOT ENOUGH SPACE TO RESUME VP (VPER)');
1200660 1 2              PUT DATA (VP.VPID,VP.SAVESIZE,DSA,EOS);
1200670 1 2              STOP;
1200680 1 2          END;
1200690 1 1          IF DSA ^= VP.BDSA THEN DO;
1200700 1 2              PUT SKIP LIST (' ERROR: BDSA DISCREPENCY (VPER)');
1200710 1 2              PUT DATA (VP.VPID,VP.BDSA,DSA);
1200720 1 2              STOP;
1200730 1 2          END;

1200750 1 1          VP.STATUS = 'RUNNING';
1200760 1 1          CALL RTSTK (VP.SAFE,VP.TDSA);

1200790 1 1          /* CONTROL SHOULD NEVER REACH HERE */
1200800 1 1          PUT SKIP LIST (' ERROR: ILLEGAL RETURN FROM RTSTK (VPER)');
1200820 1 1          STOP;
1200840 1 1          END; /* RESUMING A VP */

1200840 1 0      END VPER;

```

```

VPE00460
VPE00470
VPE00480
VPE00490
VPE00500
VPE00510
VPE00520
VPE00530
VPE00540
VPE00550
VPE00560
VPE00570
VPE00580
VPE00590
VPE00600
VPE00610
VPE00620
VPE00630
VPE00640
VPE00650
VPE00660
VPE00670
VPE00680
VPE00690
VPE00700
VPE00710
VPE00720
VPE00730
VPE00740
VPE00750
VPE00760
VPE00770
VPE00780
VPE00790
VPE00800
VPE00810
VPE00820
VPE00830
VPE00840

```

SOURCE LISTING

NUMBER LEV NT

```

10      0 VPSTART: PROC (VP, PROCNAME, ARGPT);                                VPS00010
                                           VPS00020
30      1 0 DCL 1 VP, %INCLUDE VP;.....VPS00030
          80 BDSA FIXED BIN (31),      /* ADDR (BOTTOM DSA)      */VP 00010
          80 TDSA FIXED BIN (31),      /* ADDR (TOP DSA)       */VP 00020
          80 SAVESIZE FIXED BIN (31),  /* SIZE OF AREA TO BE SAVED */VP 00030
          80 SAFE PTR,                  /* -> 1ST SAVBLK        */VP 00040
          80 PROCNAME CHAR (7) VAR,    /* NAME OF TOP LEVEL PROC */VP 00050
          80 STATUS CHAR (12) VAR,     /* VP 00060
          80 MAIL PTR,                  /* CHAIN OF INCOMING MAIL */VP 00070
          80 WAIT,                      /* VP 00080
          81 BOX FIXED BIN,             /* BOX AWAITING MAIL     */VP 00090
          81 MSG PTR,                   /* MSG IN WAIT.BOX      */VP 00100
          80 VPID FIXED BIN,            /* INDEX IN THE VPST    */VP 00110
          80 LEVEL FIXED BIN,           /* LEVEL                 */VP 00120
          80 VTIME FIXED BIN (31)      /* VP 00130
          .....                        VPS00030
200040  1 0 DCL PROCNAME CHAR(*) VAR;    VPS00040
200050  1 0 DCL ARGPT PTR;                VPS00050
                                           VPS00060
200070  1 0 DCL NULL BUILTIN;             VPS00070
          %INCLUDE QEVENT;.....VPS00080
300010  1 0 DCL QEVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR,
          FIXED BIN (31), PTR);          QEVO00010
          .....QEVO00020
          .....VPS00080
          %INCLUDE STIMER;.....VPS00090
400010  1 0 DCL STIMER ENTRY RETURNS (FIXED BIN (31));
          .....STI00010
          .....VPS00090
          .....VPS00100
400110  1 0 VP.STATUS = 'NASCENT';        VPS00110
400120  1 0 VP.PROCNAME = PROCNAME;        VPS00120
400130  1 0 VP.SAFE = NULL;                VPS00130
400140  1 0 VP.WAIT.BOX = 0;               VPS00140
400150  1 0 VP.WAIT.MSG = ARGPT;           VPS00150
400160  1 0 VP.SAVESIZE = 0;               VPS00160
400170  1 0 VP.MAIL = NULL;                VPS00170
400180  1 0 VP.VTIME = 0;                  VPS00180
400190  1 0 CALL QEVENT (VP.LEVEL*3+3,STIMER,'SYNC',VP.VPID,NULL);
          /* KICK START */                VPS00190
                                           VPS00200
                                           VPS00210
400220  1 0 END VPSTART;                  VPS00220

```

SOURCE LISTING

NUMBER LEV NT

```

10      0  WAIT: PROC (BOXID);                                WA100010
                                                    WA100020
100010  1  0  %INCLUDE QEVENT;.....                          WA100030
DCL QEVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR,    QEV00010
                FIXED BIN (31), PTR);                          QEV00020
                .....                                       WA100030
%INCLUDE STIMER;.....                                       WA100040
200010  1  0  DCL STIMER ENTRY RETURNS (FIXED BIN (31));      STI00010
                .....                                       WA100040
200050  1  0  DCL SLEEP ENTRY;                                WA100050
200060  1  0  DCL (NULL) BUILTIN;                             WA100060
                                                    WA100070
%INCLUDE VPX;.....                                       WA100080
300010  1  0  DCL THISVP PTR EXTERNAL STATIC;                /* -> CURRENT VP, SET BY VPER */VPX00010
                                                    VPX00020
300030  1  0  DCL 1 VP BASED (THISVP), %INCLUDE VP;.....     VPX00030
                80 BDSA FIXED BIN (31), /* ADDR (BOTTOM DSA) */VP 00010
                80 TOSA FIXED BIN (31), /* ADDR (TOP DSA) */VP 00020
                80 SAVESIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED */VP 00030
                80 SAFE PTR, /* -> 1ST SAVBLK */VP 00040
                80 PROCNAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC */VP 00050
                80 STATUS CHAR (12) VAR, VP 00060
                80 MAIL PTR, /* CHAIN OF INCOMING MAIL */VP 00070
                80 WAIT, VP 00080
                81 BOX FIXED BIN, /* BOX AWAITING MAIL */VP 00090
                81 MSG PTR, /* MSG IN WAIT.BOX */VP 00100
                80 VPID FIXED BIN, /* INDEX IN THE VPST */VP 00110
                80 LEVEL FIXED BIN, /* LEVEL */VP 00120
                80 VTIME FIXED BIN (31) VP 00130
                ..... VPX00030
                ; VPX00040
                ..... WA100080
500090  1  0  DCL BOXID FIXED BIN; WA100090
                ..... WA100100
500110  1  0  VP.WAIT.BOX = BOXID; WA100110
500120  1  0  CALL QEVENT (VP.LEVEL*3+3,STIMER,'WAIT',VP.VPID,NULL); WA100120
500130  1  0  CALL SLEEP; WA100130
                ..... WA100140
500150  1  0  END WAIT; WA100150

```

APPENDIX B: LISTINGS OF ASSEMBLY LANGUAGE ROUTINES

ASSEMBLY LANGUAGE ROUTINES:

GET4
RTIMER
RTSTK
SVSTK

ASM 0201 21.34 05/18/81

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT	
				1 *	GET4 GETS THE VALUE OF THE CURRENT DSA, NAB, SEGNO & EOS	GET00010
				2 *		GET00020
				3 *	DCL GET ENTRY OPTIONS (ASSEMBLE, INTER);	GET00030
				4 *	CALL GET4 (DSA, NAB, SEGNO, EOS);	GET00040
				5 *		GET00050
000000				6 GT	CSECT	GET00060
				7	ENTRY GET4	GET00070
				8	DC C'GET4 '	GET00080
000000 C7C5E3F440				9	DC AL1(5)	GET00090
000005 05				10 GET4	STM 14,12,12(13) SAVE REGISTERS	GET00100
000006 90EC D00C	0000C			11	L 6,0(1)	GET00110
00000A 5861 0000	00000			12	ST 13,0(6) ARG1 <= R13	GET00120
00000E 50D6 0000	00000			13	L 6,4(1)	GET00130
000012 5861 0004	00004			14	L 7,76(13) R7 <= (SEGNO, NAB)	GET00140
000016 587D 004C	0004C			15	LA 7,0(7) R7 <= NAB	GET00150
00001A 4177 0000	00000			16	ST 7,0(6) ARG2 <= NAB	GET00160
00001E 5076 0000	00000			17	L 6,8(1)	GET00170
000022 5861 0008	00008			18	L 7,76(13) R7 <= (SEGNO, NAB)	GET00180
000026 587D 004C	0004C			19	SRL 7,24 R7 <= SEGNO	GET00190
00002A 8870 0018	00018			20	ST 7,0(6) ARG3 <= SEGNO	GET00200
00002E 5076 0000	00000			21	L 6,12(1)	GET00210
000032 5861 000C	0000C			22	L 7,12(12) R7 <= (SEGNO, EOS)	GET00220
000036 587C 000C	0000C			23	LA 7,0(7) R7 <= EOS	GET00230
00003A 4177 0000	00000			24	ST 7,0(6) ARG4 <= EOS	GET00240
00003E 5076 0000	00000			25	LM 14,12,12(13) SAVE REGISTOR	GET00250
000042 98EC D00C	0000C			26	BR 14	GET00260
000046 07FE				27	END	GET00270

ASM 0201 21.34 05/18/81

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT
000000				1 RTM	CSECT
				2	ENTRY RTIMER
				3	DC C' RTIMER'
				4	DC AL1(6)
				5 RTIMER	STM 14,12,12(13)
				6	BALR 11,0
				7	USING *,11
				8	LA 4,BLK
				9	DC X'8340000C'
				10	L 4,28(4)
				11	N 4,=X'7FFFFFFF'
				12	L 1,0(1)
				13	ST 4,0(1)
				14	LM 14,12,12(13)
				15	BR 14
				16 *	40
				17 BLK	DS
				18	END
				19	=X'7FFFFFFF'
000000	40D9E3C9D4C5D9				
000007	06	0000C			
000008	90EC D00C				
00000C	0580				
00000E	4140 8022	0000E			
000012	8340000C				
000016	5844 001C	00030			
00001A	5440 8042				
00001E	5811 0000	0001C			
000022	5041 0000	00050			
000026	98EC D00C	00000			
00002A	07FE	0000C			
000030					
000050	7FFFFFFF				

DIAG R4,X'0C', GET BLOCK
LOWER 32 BITS OF TOTAL CPU TIME

RT100010
RT100020
RT100030
RT100040
RT100050
RT100060
RT100070
RT100080
RT100090
RT100100
RT100110
RT100120
RT100130
RT100140
RT100150
RT100160
RT100170
RT100180

LOC OBJECT CODE ADDR1 ADDR2 STMT SOURCE STATEMENT

ASM 0201 21.34 05/18/81

```

1 * RTSTK RESTORES THE STACK:
2 *
3 * DCL RTSTK ENTRY OPTIONS (ASSEMBLE, INTER);
4 * CALL RTSTK (PTR, DSA);
5 *
6 * WHERE
7 *
8 * PTR -> SAVBLK: 0(SAVBLK) -> NEXT SAVBLK
9 *              4(SAVBLK) -> 1ST LOCATION TO SAVE/RESTORE
10 *             8(SAVBLK) -> SIZE OF SAVBLK
11 *             12(SAVBLK)... SAVE AREA
12 *
13 * DSA -> RETURN DSA (WHERE RTSTK SHOULD RETURN)
14 *
15 RT CSECT
16 ENTRY RTSTK
17 DC C'RTSTK'
18 DC AL1(5)
19 *
20 RTSTK STM 14,12,12(13) SAVE REGISTERS
21 BALR 11,0 ESTABLISH BASE
22 USING -,11
23 LR 6,13 R6 -> OLD DSA
24 LA 13,SAFE R13 -> NEW DSA
25 ST 6,4(13) CHAIN BACK
26 MVI 0(13),X'80' SET FLAGS
27 MVI 1(13),X'00'
28 MVC 72(12,13),72(6) COPY A(LSW), ETC.
29 *
30 L 6,0(1) R6 -> 1ST ARG -> 1ST SAVBLK
31 L 6,0(6) R8 -> 1ST SAVBLK
32 *
33 AGAIN L 7,4(6) R7 -> SAVBLK.ADDR
34 LA 8,12(6) R8 -> SAVBLK.AREA
35 L 10,8(6) R10 -> SAVBLK.SIZE
36 S 10,=F'4' WE ARE COUNTING FROM SIZE-4 TO 0
37 *
38 LOOP L 9,0(10,8) GET WORD
39 ST 9,0(10,7) RESTORE WORD
40 S 10,=F'4' DEC COUNT
41 BNM LOOP AGAIN IF MORE
42 *
43 L 6,0(6) R6 -> SAVBLK.NEXT
44 CL 6,=X'FF000000' NULL?
45 BNE AGAIN
46 *
47 L 6,4(1) R6 -> 2ND ARG -> TOP DSA
48 L 13,0(6) R13 -> TOP DSA
49 LM 14,12,12(13) RETURN THERE
50 BR 14
51 *
52 SAFE DS 128F
53 END
54 =F'4'
55 =X'FF070000'

```

000000
000000 D9E3E2E3D2
000005 05
000006 90EC D00C 0000C
00000A 05B0 0000C
00000C 186D 0000C
00000E 41D0 B05C 0006B
000012 506D 0004 00004
000016 928D D000 00000
00001A 9200 D001 00001
00001E D208 D048 8048 00048 00048
000024 5861 0000 00000
000028 5866 0000 00000
00002C 5876 0004 00004
000030 4186 000C 0000C
000034 58A6 0008 00008
000038 58A0 825C 00268
00003C 589A 8000 00000
000040 509A 7000 00000
000044 58A0 825C 00268
000048 478D B030 0003C
00004C 5866 0000 00000
000050 5560 8260 0026C
000054 4770 B020 0002C
000058 5861 0004 00004
00005C 58D6 0000 00000
000060 98EC D00C 0000C
000064 07FE
000068
000268 00000004
00026C FF000000

RTS00010
RTS00020
RTS00030
RTS00040
RTS00050
RTS00060
RTS00070
RTS00080
RTS00090
RTS00100
RTS00110
RTS00120
RTS00130
RTS00140
RTS00150
RTS00160
RTS00170
RTS00180
RTS00190
RTS00200
RTS00210
RTS00220
RTS00230
RTS00240
RTS00250
RTS00260
RTS00270
RTS00280
RTS00290
RTS00300
RTS00310
RTS00320
RTS00330
RTS00340
RTS00350
RTS00360
RTS00370
RTS00380
RTS00390
RTS00400
RTS00410
RTS00420
RTS00430
RTS00440
RTS00450
RTS00460
RTS00470
RTS00480
RTS00490
RTS00500
RTS00510
RTS00520
RTS00530

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT	ASM 0201 21.34 05/18/81
				1 *	SVSTK SAVES THE STACK:	SVS00010
				2 *		SVS00020
				3 *	DCL SVSTK ENTRY OPTIONS (ASSEMBLE, INTER);	SVS00030
				4 *	CALL SVSTK (PTR, DSA);	SVS00040
				5 *		SVS00050
				6 *	WHERE	SVS00060
				7 *		SVS00070
				8 *	PTR -> SAVBLK: 0(SAVBLK) -> NEXT SAVBLK	SVS00080
				9 *	4(SAVBLK) -> 1ST LOCATION TO SAVE/RESTORE	SVS00090
				10 *	8(SAVBLK) -> SIZE OF SAVBLK	SVS00100
				11 *	12(SAVBLK)... SAVE AREA	SVS00110
				12 *		SVS00120
				13 *	DSA -> RETURN DSA (WHERE SVSTK SHOULD RETURN)	SVS00130
				14 *		SVS00140
000000				15 SV	CSECT	SVS00150
				16	ENTRY SVSTK	SVS00160
000000	E2E5E2E3D2			17	DC C'SVSTK'	SVS00170
000005	05			18	DC AL(5)	SVS00180
				19 *		SVS00190
000006	90EC D00C	0000C		20 SVSTK	STM 14,12,12(13) SAVE REGISTERS	SVS00200
00000A	05B0			21	BALR 11,0 ESTABLISH BASE	SVS00210
		0000C		22	USING *,11	SVS00220
00000C	186D			23	LR 6,13 R6 -> OLD DSA	SVS00230
00000E	41D0 B05C	00068		24	LA 13,SAFE R13 -> NEW DSA	SVS00240
000012	506D 0004	00004		25	ST 6,4(13) CHAIN BACK	SVS00250
000016	9280 D000	00000		26	MVI 0(13),X'80' SET FLAGS	SVS00260
00001A	9200 D001	00001		27	MVI 1(13),X'00'	SVS00270
00001E	D20B D04B	6D4B D004B D004B		28	MVC 72(12,13),72(6) COPY A(LSW), ETC.	SVS00280
				29 *		SVS00290
000024	5861 D000	00000		30	L 6,0(1) R6 -> 1ST ARG -> 1ST SAVBLK	SVS00300
000028	5866 D000	00000		31	L 6,0(6) R6 -> 1ST SAVBLK	SVS00310
				32 *		SVS00320
00002C	5876 0004	00004		33 AGAIN	LA 7,4(6) R7 -> SAVBLK.ADDR	SVS00330
000030	4186 000C	0000C		34	LA 8,12(6) R8 -> SAVBLK.AREA	SVS00340
000034	58A6 000B	0000B		35	L 10,8(6) R10 -> SAVBLK.SIZE	SVS00350
000038	58A0 B25C	00268		36	S 10,-F'4' WE ARE COUNTING FROM SIZE-4 TO 0	SVS00360
				37 *		SVS00370
00003C	589A 7000	00000		38 LOOP	L 9,0(10,7) GET WORD	SVS00380
000040	509A 9000	00000		39	ST 9,0(10,8) RESTORE WORD	SVS00390
000044	58A0 B25C	00268		40	S 10,-F'4' DEC COUNT	SVS00400
000048	47B0 B030	0003C		41	BNM LOOP AGAIN IF MORE	SVS00410
				42 *		SVS00420
00004C	5866 D000	00000		43	L 6,0(6) R6 -> SAVBLK.NEXT	SVS00430
000050	5560 B260	0026C		44	CL 6,-X'FF000000' NULL?	SVS00440
000054	4770 B020	0002C		45	BNE AGAIN	SVS00450
				46 *		SVS00460
000058	5861 0004	00004		47	L 6,4(1) R6 -> 2ND ARG -> TOP DSA	SVS00470
00005C	58D6 0000	00000		48	L 13,0(6) R13 -> TOP DSA	SVS00480
000060	98EC D00C	0000C		49	LM 14,12,12(13) RETURN THERE	SVS00490
000064	07FE			50	BR 14	SVS00500
				51 *		SVS00510
000068				52 SAFE	DS 128F	SVS00520
				53	END	SVS00530
000268	00000004			54	=F'4'	
00026C	FF000000			55	=X'FF000000'	

APPENDIX C: LISTINGS OF MACRO FILES (DATA DECLARATIONS)

MACRO FILES:

BOX
CONFIG
DEBUG
EVENT
LIST
LOS
LOSX
MSG
PARAMS
PFADDR
PFLEVEL
PFMSG
PFS
PFSVC
QUEUES
SAVBLK
SVR
SVRX
VP
VPX

FILE: BOX PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL BOXPT PTR;

DCL 1 BOX BASED (BOXPT),
2 NEXT PTR,
2 ID FIXED BIN,
2 LIST PTR;

BOX00010
BOX00020
BOX00030
BOX00040
BOX00050
BOX00060
BOX00070

FILE: CONFIG PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

XDECLARE MAXLEVEL FIXED;
XDECLARE MAXQ FIXED;
XMAXLEVEL = 1;
XMAXQ = 6;

CON00010
CON00020
CON00030
CON00040

FILE: DEBUG PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL 1 DEBUG EXTERNAL STATIC,
2 SLEEPS BIT(1) INIT ('0'B),
2 SAHERS BIT(1) INIT ('0'B),
2 SCHEDULERS BIT(1) INIT ('0'B),
2 SHELLS BIT(1) INIT ('0'B),
2 AAHERS BIT(1) INIT ('0'B);

DEB00010
DEB00020
DEB00030
DEB00040
DEB00050
DEB00060

FILE: EVENT PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

70 NEXT PTR,
70 SYTIME FIXED BIN (31),
70 TYPE CHAR (12) VAR,
70 INDEX FIXED BIN (31),
70 PTR PTR

/* NEXT EVENT

*/ EVE00010
EVE00020
EVE00030
EVE00040
EVE00050

FILE: LIST PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL LISTPT PTR;

LIS00010
LIS00020
LIS00030
LIS00040
LIS00050
LIS00060

DCL 1 LIST BASED (LISTPT),
2 NEXT PTR,
2 THIS PTR;

FILE: LOS

PLI

A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

70 LEVEL FIXED BIN,	LOS00010
70 SRQ,	LOS00020
71 SIZE FIXED BIN,	LOS00030
71 MAX FIXED BIN,	LOS00040
70 DBB,	LOS00050
71 SIZE FIXED BIN,	LOS00060
71 MAX FIXED BIN,	LOS00070
70 VPS, %DECLARE MAXVP FIXED: %MAXVP = 20;	LOS00080
71 TABLE (MAXVP), %INCLUDE VP;	LOS00090

FILE: LOSX PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL THISLOS PTR EXTERNAL STATIC: /* -> CURRENT LOS, ST BY SAHER */LOS00010
DCL 1 LOS BASED (THISLOS), %INCLUDE LOS; 1 LOS00020
LOS00030
LOS00040

FILE: MSG PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL MSGLEN FIXED BIN;
DCL MSGPT PTR;
DCL 1 MSG BASED (MSGPT),
 2 LEN FIXED BIN,
 2 STR CHAR (MSGLEN REFER (LEN));

MSG00010
MSG00020
MSG00030
MSG00040
MSG00050
MSG00060

FILE: PARAMS PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL 1 PARAMS EXTERNAL STATIC,
2 THRU_RATE FIXED BIN (31,7),
2 DELAY_GB_GC FIXED BIN (31),
2 DELAY_GC_GB FIXED BIN (31),
2 TERMINALS FIXED BIN;

PAR00010
PAR00020
PAR00030
PAR00040
PAR00050
PAR00060

FILE: PFADDR PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL PT_ADDR PTR:

PFA00010
PFA00020
PFA00030
PFA00040
PFA00050
PFA00060
PFA00070

DCL 1 PF_ADDR BASED (PT_ADDR),
2 VPID FIXED BIN,
2 BOXID FIXED BIN,
2 PTR PTR;

FILE: PFLEVEL PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL PY_LEVEL PTR;

DCL 1 PF_LEVEL BASED (PY_LEVEL),
2 LEVEL FIXED BIN,
2 PTR PTR;

PFL00010
PFL00020
PFL00030
PFL00040
PFL00050
PFL00060

FILE: PFMSG PL1 A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL PT_MSG PTR:

DCL 1 PF_MSG BASED (PT_MSG),
2 LEN FIXED BIN,
2 TYPE CHAR (12) VAR,
2 PTR PTR:

PFM00010
PFM00020
PFM00030
PFM00040
PFM00050
PFM00060
PFM00070

FILE: PFS PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL PT_S PTR;

DCL 1 PF_S BASED (PT_S),
2 LEN FIXED BIN,
2 PTR PTR;

PFS00010
PFS00020
PFS00030
PFS00040
PFS00050
PFS00060

FILE: PFSVC PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL PT_SVC PTR;

DCL 1 PF_SVC BASED (PT_SVC),
2 SVC CHAR (7) VAR,
2 PTR PTR;

PFS00010
PFS00020
PFS00030
PFS00040
PFS00050
PFS00060

PAGE 001

CONVERSATIONAL MONITOR SYSTEM

FILE: QUEUES PLI A

DCL SQUEUE (0:MAXQ) PTR EXTERNAL STATIC;
DCL EQUEUE (0:MAXQ) PTR EXTERNAL STATIC;

QS 00010
QS 00020
QS 00030

FILE: SAVBLK PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

XDECLARE MAXSAVBYTE FIXED;
XDECLARE MAXSAVWORD FIXED;

XMAXSAVBYTE = 1000;
XMAXSAVWORD = 250;

DCL NEWSAVBLK PTR;

DCL 1 SAVBLK BASED (NEWSAVBLK),

2 NEXT PTR,

2 ADDR FIXED BIN (31),

2 SIZE FIXED BIN (31),

2 AREA (MAXSAVWORD) FIXED BIN (31);

XDEACTIVATE MAXSAVWORD;

SAV00010
SAV00020
SAV00030
SAV00040
SAV00050
SAV00060
SAV00070
SAV00080
SAV00090
•/SAV00100
•/SAV00110
•/SAV00120
SAV00130
SAV00140
SAV00150
SAV00160

/* NEXT SAVED ITEM
/* BEG ADDR OF SAVED ITEM
/* SIZE OF SAVED ITEM

FILE: SVR

PLI

A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

70 NEXT PTR.
70 STIME FIXED BIN (31).
70 STIMEQ FIXED BIN (31).
70 RTIME FIXED BIN (31).
70 RRATE FIXED BIN (15,7).
70 ID FIXED BIN

/* ACCUMULATED STIME

/* REAL CPU TICK RATE

SVR00010
SVR00020
•/SVR00030
SVR00040
•/SVR00050
SVR00060

FILE: SVRX PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL THISVR PTR EXTERNAL STATIC: /* -> CURRENT SVR, SET BY SAHER */ SVR00010
DCL 1 SVR BASED (THISVR), XINCLUDE SVR; ; SVR00020
SVR00030
SVR00040

FILE: VP

PL1 A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

80 BDSA FIXED BIN (31).	/* ADDR (BOTTOM DSA)	*/VP 00010
80 TDSA FIXED BIN (31).	/* ADDR (TOP DSA)	*/VP 00020
80 SAVESIZE FIXED BIN (31).	/* SIZE OF AREA TO BE SAVED	*/VP 00030
80 SAFE PTR.	/* -> 1ST SAVBLK	*/VP 00040
80 PROCNAME CHAR (7) VAR.	/* NAME OF TOP LEVEL PROC	*/VP 00050
80 STATUS CHAR (12) VAR.		VP 00060
80 MAIL PTR.	/* CHAIN OF INCOMING MAIL	*/VP 00070
80 WAIT.		VP 00080
81 BOX FIXED BIN.	/* BOX AWAITING MAIL	*/VP 00090
81 MSG PTR.	/* MSG IN WAIT.BOX	*/VP 00100
80 VPID FIXED BIN.	/* INDEX IN THE VPST	*/VP 00110
80 LEVEL FIXED BIN.	/* LEVEL	*/VP 00120
80 VTIME FIXED BIN (31)		VP 00130

FILE: VPX PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL THISVP PTR EXTERNAL STATIC; /* -> CURRENT VP, SET BY VPER */VPX00010
DCL 1 VP BASED (THISVP), XINCLUDE VP: ; VPX00020
 VPX00030
 VPX00040

APPENDIX: LISTINGS OF MACRO FILES (ENTRY DECLARATIONS)

MACRO FILES:

CONS
EXECUTE
GET4
QEVENT
RTIMER
RTSTK
STIMER
SVCER
SVSTK
USERS
VPSTART

FILE: CONS PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL CONS ENTRY (PTR, PTR) RETURNS (PTR);

CON00010

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

FILE: EXECUTE PL1 A
DCL EXECUTE ENTRY (CHAR(*) VAR):

EXE00010

FILE: GET4 PL1 A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL DSA FIXED BIN (31,0).
NAB FIXED BIN (31,0).
SEG FIXED BIN (31,0).
EOS FIXED BIN (31,0);

GET00010
GET00020
GET00030
GET00040
GET00050
GET00060
GET00070
GET00080
GET00090

DCL GET4 ENTRY (FIXED BIN (31), FIXED BIN (31),
FIXED BIN (31), FIXED BIN (31))
OPTIONS (ASSEMBLER, INTER);

FILE: QEVENT PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL QEVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR,
FIXED BIN (31), PTR);

QEV00010
QEV00020

FILE: RTIMER PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

RT100010

DCL RTIMER ENTRY RETURNS (FIXED BIN (31));

FILE: RTSTK PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL RTSTK ENTRY (PTR, FIXED BIN (31))
 OPTIONS (ASSEMBLER, INTER);

/* ARG1: PTR -> SAVBLK
 ARG2: ADDR OF RETURN DSA (TOP DSA)
*/

RTS00010
RTS00020
RTS00030
RTS00040
RTS00050
RTS00060
RTS00070

FILE: STIMER PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL STIMER ENTRY RETURNS (FIXED BIN (31));

ST100010

FILE: SVCER PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL SVCER ENTRY (FIXED BIN, PTR):

SVC00010

FILE: SVSTK PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL SVSTK ENTRY (PTR, FIXED BIN (31))
 OPTIONS (ASSEMBLER, INTER):

/* ARG1: PTR -> SAVBLK
 ARG2: ADDR OF RETURN DSA (BOTTOM DSA)
*/

SVS00010
SVS00020
SVS00030
SVS00040
SVS00050
SVS00060
SVS00070

FILE: USERS PL1 A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

XINCLUDE SYIMER;
DCL SEND ENTRY (FIXED BIN, FIXED BIN, FIXED BIN, CHAR(*) VAR,
 FIXED BIN, PTR);
DCL WAIT ENTRY (FIXED BIN);
DCL SYNC ENTRY;
DCL FINISH ENTRY;

USE00010
USE00020
USE00030
USE00040
USE00050
USE00060
USE00070

FILE: VPSTART PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL VPSTART ENTRY (/ * VP */ , CHAR(*) VAR, PTR);

VPS00010

END

DATE
FILMED

8 82

DTIC