SHELL: A SIMULATOR FOR THE SOFTWARE TEST VEHICLE OF THE INFOPLE-ETC(U)

JAN 82 T TO

UNCLASSIFIED

CISR-M010-8201-08

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

Technical Report #8

By
Tak To

January, 1982

Principal Investigator:
Professor Stuart E. Madnick

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

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited
SHELL: A Simulator for the Software Test Vehicles of the INFOPLEX Database Computer

Tak To

Sloan School of Management
50 Memorial Drive
Cambridge, MA 02139

Approved for public release; distribution unlimited.

database computer, database management system, Software Test Vehicle, distributed operating system

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 aforementioned 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.
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 infor-
mation 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 neces-
sary 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 con-
figuration; 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 **Introduction** .................................................... 1
1.1. **Infoflex** ....................................................... 1
1.2. **Scope of the Project** ........................................... 2
1.3. **Structure of this Paper** ........................................ 3

2.0 **The Hardware Environment of Infoflex** ....................... 4
2.1. **Overview** ....................................................... 4
2.2. **GC-LOS Protocol** ............................................... 7

3.0 **The Simulator** .................................................. 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 **The Local Operating System** .................................... 20
4.1. **Overview** ....................................................... 20
4.2. **System Service Requests** ....................................... 21
4.3. **The Local Operating System as a Network of Co-Routines** ... 22

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

| Figure 2.1 | Hardware Configuration of the Functional Hierarchy of INFOPLEX | 5 |
| Figure 2.2 | Protocol Between the Gateway Controller & the Local Operating System | 9 |
| Figure 6.1 | Functional Relationship of the Major Modules of SHELL | 32 |
| Figure 6.2 | Format of LOS | 34 |
| Figure 6.3 | Format of VP | 36 |
| Figure 6.4 | Format of SVR | 38 |
| Figure 6.5.1-2 | Data Structures Used at Various Stages of Message Transmission | 40 |
| Figure 6.6 | Format of the Dynamic Storage Area (DSA) | 44 |
| Figure 6.7 | Format of the Task Communication Area (TCA) | 45 |
| Figure 6.8.1-4 | State of the Stack During Context Switching | 47 |
| Figure 7.1 | Listing of Procedure EXECUTE | 54 |
| Figure 7.2 | Format of Argument for System Service Request 1 (Starting a New Virtual Process) | 60 |
| Figure 7.3.1-3 | Listing of Procedure TERM | 64 |
| Figure 7.4.1-3 | Sample Simulation Session | 69 |
| Figure 7.5 | Statistics of the Sample Simulation Session | 73 |
| Figure 7.6 | Chronology of Events in the Sample Simulation Session | 74 |

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 hierarchy or the functional hierarchy. 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

Global Bus

Local Bus

Gateway Controller

Processor

RAM/ROM

Processor

Memory

Processor

Memory

Processor

Memory

Processor

Memory
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 Request Queue (DRQ)**, a **Service Request 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.
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
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 goes 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 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 pended 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
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 RUN-NABLE.

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 VTIMEx.

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 explaination), 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

```
SHELL
  GCER*  AAHER*  SAHER*  GBER*
  |
  INITER  SCHEDULER  MAILMAN  CHECK_MAIL
  |
  VPER
  |
  RVSTK  EXECUTE
  |
  USER PROGRAM (*)
  |
  SEND*  SYNC*  WAIT*  FINISH*
  |
  SLEEP
  |
  VPSTART*  SVSTK

* calls STIMER
```

-32-
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.
<table>
<thead>
<tr>
<th>LEVEL</th>
<th>FIXED BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRQ</td>
<td>SIZE FIXED BIN</td>
</tr>
<tr>
<td>DBB</td>
<td>SIZE FIXED BIN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VPS-TABLE</th>
<th>MAXVP*</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDSA</td>
<td>FIXED BIN (31)</td>
</tr>
<tr>
<td>TDSA</td>
<td>FIXED BIN (31)</td>
</tr>
<tr>
<td>SAVESIZE</td>
<td>FIXED BIN (31)</td>
</tr>
<tr>
<td>SAFE</td>
<td>PTR</td>
</tr>
<tr>
<td>PROCNAME</td>
<td>CHAR(7) VAR</td>
</tr>
<tr>
<td>STATUS</td>
<td>CHAR(12) VAR</td>
</tr>
<tr>
<td>MAIL</td>
<td>PTR</td>
</tr>
<tr>
<td>WAIT BOX</td>
<td>FIXED BIN</td>
</tr>
<tr>
<td>WAIT MSG</td>
<td>PTR</td>
</tr>
<tr>
<td>VPID</td>
<td>FIXED BIN</td>
</tr>
<tr>
<td>LEVEL</td>
<td>FIXED BIN</td>
</tr>
<tr>
<td>VTIME</td>
<td>FIXED BIN (31)</td>
</tr>
</tbody>
</table>

* 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 used 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.
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).
\textbf{THISSVR.} (set by AAHER, SAHER)

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXT</td>
<td>PTR</td>
</tr>
<tr>
<td>STIME</td>
<td>FIXED BIN (31)</td>
</tr>
<tr>
<td>STIMEQ</td>
<td>FIXED BIN (31)</td>
</tr>
<tr>
<td>RTIME</td>
<td>FIXED BIN (31)</td>
</tr>
<tr>
<td>RRATE</td>
<td>FIXED BIN (15,7)</td>
</tr>
<tr>
<td>ID</td>
<td>FIXED BIN</td>
</tr>
</tbody>
</table>

\textbf{FIG. 6.4} FORMAT OF SVR (Server)

-38-
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.
FIG. 6.5.1 DATA STRUCTURES USED AT VARIOUS STAGES OF MESSAGE TRANSMISSION (part 1)
FIG 6.5.2
DATA STRUCTURES USED AT VARIOUS STAGES OF MESSAGE TRANSMISSION (part 2)

VP_MAIL

BOX

NEXT

BOXID

LIST

LIST

NEXT

THIS

(null)

PF_ADDR

.VPID

.BOXID

.PTR

AAHER (message)

(SEND)

SAHER (MAILMAN) (message)

SAHER (CHECK_MAIL) (message)

VP_WAIT_MSG
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 of 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])

Register 13 -

(Offsets in Hexadecimal)

<table>
<thead>
<tr>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>pointer to previous DSA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4C</th>
<th>4D</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEG#</td>
<td>Next Available Byte (NAB)</td>
</tr>
</tbody>
</table>

78
<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEG#</td>
<td>End of Segment</td>
</tr>
</tbody>
</table>

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)

VPER₂  EXECUTE₂  USER₂
VPER₂  EXECUTE₂  USER₁
VPER₁  EXECUTE₁

VPER calls GET₄ to mark the bottom of the stack.

USER program calls SLEEP, which calls GET₄ to measure the top of the stack. It then prepare the save blocks.
Fig. 6.8.2  State of the stack during context switching (part 2)

- VPER₂
- EXECUTE₂
- USER₂
- SLEEP₂
- SVSTK

SVSTK saves data on the stack into the save blocks.

- VPER₃

SVSTK returns directly to VPER.

- VPER₄
- RTSTK

VPER calls RTSTK to restore data on to the stack.

- VPER₂
- EXECUTE₂
- USER₂
- SLEEP₂

RTSTK returns to SLEEP, as if SVSTK were returning.
FIG. 6.8.3  STATE OF THE STACK DURING CONTEXT SWITCHING (part 3)

VPER₂
EXECUTE₂
USER₂
SLEEP₃

SLEEP returns.

VPER₂
EXECUTE₂
USER₃

USER program continues execution.

VPER₂
EXECUTE₂
USER₄

USER program finishes execution.

VPER₂
EXECUTE₃

USER program returns to EXECUTE.
FIG. 6.8.4 STATE OF THE STACK DURING CONTEXT SwitchING (part 4)

Legend:

- Activation Record of a procedure on the stack (DSA, AUTOMATIC variables, et al). The subscripts distinguish the different states of the procedure.
- Assembly language procedure which does not use the stack.

EXECUTE returns.
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.
EXECUTE: PROC (PROCHAME);

SOURCE LISTING

NUMBER  LEV  NT

10     0     EXECUTE: PROC (PROCHAME);

30     1     DCL PROCHAME CHAR(*) VAR;

40     1     DCL TERM ENTRY;

80     1     IF PROCHAME = 'TERM' THEN DO; CALL TERM; RETURN; END;

80     1     PUT SKIP LIST ('ERROR: UNKNOWN PROCHAME, ' , PROCHAME,

100   1     ' (EXECUTE).'

120   1     END EXECUTE;

FIG 7.1  LISTING OF PROCEDURE EXECUTE
**Table 7.1** **LIST OF EXTERNAL IDENTIFIERS**

<table>
<thead>
<tr>
<th>ENTRIES:</th>
<th>DATA:</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAHER</td>
<td>DEBUG</td>
</tr>
<tr>
<td>ADDQ</td>
<td>PARAMS</td>
</tr>
<tr>
<td>CONS</td>
<td>THISLOS</td>
</tr>
<tr>
<td>DEBUGR</td>
<td>THISSVR</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>THISVP</td>
</tr>
<tr>
<td>FINISH</td>
<td>PLIXOPT</td>
</tr>
<tr>
<td>QEVENT</td>
<td></td>
</tr>
<tr>
<td>RTIHER</td>
<td></td>
</tr>
<tr>
<td>RTSTK</td>
<td></td>
</tr>
<tr>
<td>SAHER</td>
<td></td>
</tr>
<tr>
<td>SEND</td>
<td></td>
</tr>
<tr>
<td>SHELL</td>
<td></td>
</tr>
<tr>
<td>SLEEP</td>
<td></td>
</tr>
<tr>
<td>STIMER</td>
<td></td>
</tr>
<tr>
<td>SVCER</td>
<td></td>
</tr>
<tr>
<td>STSTK</td>
<td></td>
</tr>
<tr>
<td>SYNC</td>
<td></td>
</tr>
<tr>
<td>TERM</td>
<td></td>
</tr>
<tr>
<td>VPER</td>
<td></td>
</tr>
<tr>
<td>VPSTART</td>
<td></td>
</tr>
<tr>
<td>WAIT</td>
<td></td>
</tr>
</tbody>
</table>
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:

\[
\text{CALL SEND (level,vpid,boxid,type,len,ptr)}
\]

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. Messages  
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 AUTOMATIC 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.
Figure 7.3.1 LISTING OF PROCEDURE TERM (part 1)
PL/I OPTIMIZING COMPILER  TERM: PROC;

NUMBER LEV NT

*******
%INCLUDE PFSVC;******************************
110010 1 0 DCL P1_SVC PTR; PFS00010 PFS00020
110030 1 0 DCL 1 PF_SVC BASED (PT_SVC). 2 SVC CHAR (? VAR,
2 PTR PTR;

*******
120010 1 0 DCL MSGLEN FIXED BIN; MSG00010
120020 1 0 DCL MSGPT PTR; MSG00020
120030 1 0 DCL 1 MSG BASED (MSGPT); 2 LEN FIXED BIN,
2 STR CHAR (MSGLEN REFER (LEN)); MSG00030 MSG00050

120010 1 0 DCL (LEVEL,VPID,BOXID) FIXED BIN;
120011 1 0 DCL MESSAGE CHAR(80) VAR;
120012 1 0 DCL COMMAND CHAR(12);
120013 1 0 DCL TYPE CHAR(S) VAR;
120015 1 0 LOOP:

    PUT SKIP EDIT (' TERM: LEVEL = ',LEVEL,' VP = ',VPID,
                    ' WAIT. BOX = ',VPID,' BOXID,' SVR = ',SVR,' ID = ',SVR,' STIME = ',
                    STIME,' A,F(2),A,F(2),A,F(2),A,F(2),A,F(11));
120019 1 0 MSGPT = VP.WAIT.MSG;
120020 1 0 IF (MSGPT = NULL) THEN DO;
120021 1 1 PUT SKIP EDIT (' MSG: ',MSG.Str) (A,AMSG.LEN));
120022 1 1 FREE MSG;
120023 1 1 VP.WAIT.MSG = NULL;
120024 1 1 END;
120026 1 0 WORK:

    PUT SKIP:
120028 1 0 DISPLAY(' COMMAND?') REPLY (COMMAND);
120030 1 0 IF (COMMAND = 'BUILD') THEN DO;
120031 1 1 PUT SKIP LIST (' LEVEL: ');
120032 1 1 GET LIST (LEVEL);
120033 1 1 MSGLEN = 7;
120034 1 1 ALLOCATE MSG;
120035 1 1 MSG.STR = 'NEW VP1';
120036 1 1 ALLOCATE PF_SVC;
120037 1 1 PF_SVC.SVC = 'TERM';
120038 1 1 PF_SVC.PTR = MSGPT;
120039 1 1 CALL SEND (LEVEL,0,1,'5',19,PF_SVC);
120040 1 1 GOTO WORK;

FIG. 7.3.2 LISTING OF PROCEDURE TERM (part 2)
PL/I OPTIMIZING COMPILER  TERM: PROC;

NUMBER LEV NT

1200410 1 1 END:  TERO0410
1200430 1 0 ELSE IF (COMMAND = 'SEND') THEN DO;  TERO0430
1200440 1 1 PUT SKIP LIST (': LEVEL.VPID.BOXID.TYPE.MESSAGE> '):  TERO0440
1200450 1 1 GET LIST (LEVEL.VPID.BOXID.TYPE.MESSAGE):  TERO0450
1200460 1 1 MSGLEN = LENGTH (MESSAGE):  TERO0460
1200470 1 1 ALLOCATE MSG;  TERO0470
1200480 1 1 MSG.SIR = MESSAGE:  TERO0480
1200490 1 1 CALL SEND (LEVEL.VPID.BOXID.TYPE.MSGLEN.MSGPT):  TERO0490
1200500 1 1 GOTO WORK;  TERO0500
1200510 1 1 END;  TERO0510
1200530 1 0 ELSE IF (COMMAND = 'WAIT') THEN DO;  TERO0530
1200540 1 1 PUT SKIP LIST (' BOX> '):  TERO0540
1200550 1 1 GET LIST (BOXID):  TERO0550
1200560 1 1 PUT SKIP LIST (' WAITING'):  TERO0560
1200570 1 1 CALL WAIT (BOXID):  TERO0570
1200580 1 1 GOTO LOOP;  TERO0580
1200590 1 1 END;  TERO0590
1200610 1 0 ELSE IF (COMMAND = 'SYNC') THEN DO;  TERO0610
1200620 1 1 PUT SKIP LIST (" SYNCING"):  TERO0620
1200630 1 1 CALL SYNC:  TERO0630
1200640 1 1 GOTO LOOP;  TERO0640
1200650 1 1 END;  TERO0650
1200670 1 0 ELSE IF (COMMAND = 'FINISH') THEN DO;  TERO0670
1200680 1 1 PUT SKIP LIST (" FINISHING"):  TERO0680
1200690 1 1 CALL FINISH;  TERO0690
1200700 1 1 RETURN;  TERO0700
1200710 1 1 END;  TERO0710
1200730 1 0 ELSE DO: /* UNKNOWN COMMAND */  TERO0730
1200740 1 1 PUT LIST (" ??");  TERO0740
1200750 1 1 GOTO WORK;  TERO0750
1200760 1 1 END;  TERO0760
1200780 1 0 END TERM;

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 $N$ ($N = \text{TERMINALS}$, 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.
LOAD SHELL ( CLEAR NODUP NOMAP )
FILEDEF STATS DISK STATS LISTING A1 ( BLOCK 800 )

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 INITI

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 INITI

COMMAND?
build
LEVEL>
1
COMMAND?
wait
BOX>
1

FIG. 7.4.1 SAMPLE SIMULATION SESSION (part 1)
FIG. 7.4.2 SAMPLE SIMULATION SESSION (part 2)
FIG. 7.4.3 SAMPLE SIMULATION SESSION (part 3)
At level 1, VPl.1 waits for box 1 and VPl.2 waits for box 2. Then VPl.3 sends a message to VP0.1. After sending the message, VPl.3 terminates itself.

When VP0.1 wakes up, it sends a message to VPl.1 and then terminates. Similarly, VPl.1 sends to VPl.2; VPl.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.
### Simulation Parameters:

<table>
<thead>
<tr>
<th>Level</th>
<th>NP=</th>
<th>2 Rate=</th>
<th>1.000</th>
<th>0 Params.DELAY_GC_GB=</th>
<th>0 Params.TERM=</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Simulation:

- **GC (0): MSGOUT DELAY**
- **GBI STIME**
- **GC (1): MSGIN DELAY**
- **GC (2): MSGOUT DELAY**
- **GBI STIME**
- **GC (3): MSGIN DELAY**
- **GC (4): MSGOUT DELAY**
- **GBI STIME**
- **GC (5): MSGIN DELAY**
- **GBI STIME**
- **GC (6): MSGIN DELAY**

### Statistics:

#### Level 0:
- **SRC. MAX =**
- **10 DBR.MAX =**

#### Level 1:
- **SRC. ID=**
- **1 SRC.STIME=**
- **384424 SRC.STIME=**
- **81;**
- **SVR.STIME=**
- **210192 SVR.STIME=**
- **724;**

#### Level 1:
- **SRC. ID=**
- **1 SRC.STIME=**
- **385128 SRC.STIME=**
- **764;**

#### Level 1:
- **SRC. ID=**
- **1 SRC.STIME=**
- **481150 SRC.STIME=**
- **266197;**

#### Level 1:
- **SRC. ID=**
- **2 SRC.STIME=**
- **401592 SVR.STIME=**
- **220819;**
Level 0, Processor 1

Level 0, Processor 2

Level 1, Processor 1

Level 1, Processor 2

STIME = 555
VPO.1 builds VP1.1; waits for box 1

STIME = 1062
VPO.2 builds VP1.2; waits for box 2

STIME = 58299
VP1.1 waits for box 1

STIME = 70249
VP1.2 waits for box 2

STIME = 95062
VPO.3 builds VP1.3; waits for box 3

STIME = 151919
VP1.3 sends to VPO.1; terminates

STIME = 209840
VPO.1 waken; sends to VP1.1; terminates

STIME = 267747
VP1.1 waken; sends to VP1.2; terminates

STIME = 324742
VP1.2 waken; sends to VPO.2; terminates

STIME = 384779
VPO.2 waken; sends to VPO.3; terminates

STIME = 441675
VPO.3 waken; terminates

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


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

10  0  AAHER: PROC (LOS, SVR, EVENT);

30  1  DCL 1 LOS,
    70 LEVEL FIXED BIN,
    70 SQO,
    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 TOSA FIXED BIN (31),
    80 SAVESIZE FIXED BIN (31),
    80 SAVE PTR,
    80 PROGRAM CHAR (7) VAR,
    80 STATUS CHAR (12) VAR,
    80 MAIL PTR,
    80 WAIT,
    80 BOX FIXED BIN,
    80 MSG PTR,
    80 VPIO FIXED BIN,
    80 VTIME FIXED BIN (31)

300040  1  0  DCL 1 SVR,
    70 NEXT PTR,
    70 STIME FIXED BIN (31),
    70 STIMEO FIXED BIN (31),
    70 RTEME 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

700060  1  0  DCL (THISLOS, THISSVR) PTR EXTERNAL STATIC;
700070  1  0  DCL SM 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),
PL/I OPTIMIZING COMPILE

NUMBER LEV NT

2 SCHEDULERS BIT(1) INIT ("0'B),
2 SHELLS BIT(1) INIT ("0'B),
2 AHFERS BIT(1) INIT ("0'B);

900010 1 0 DCL PT_ADDR PTR;

900020 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;

1000020 1 0 DCL I (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 Q_EVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR,
FIXED BIN (31), PTR);
1300010 1 0 DCL SVC_ENTRY (FIXED BIN, PTR);
1300100 1 0 DCL (ADDR) BUILTIN;

1200170 1 0 THISLOS = ADDR (LOS);
1200180 1 0 THISSVR = ADDR (SVR);

1200220 1 0 IF (EVENT.TYPE = "0") THEN DO;
1300210 1 0 PV_ADDR = EVENT.PTR;
1300220 1 0 LOS.DBB.SIZE = LOS.DBB.SIZE - 1;
1300230 1 0 ELSE DO; /* EVENT.TYPE = S */
1300240 1 0 Pt_S = EVENT.PTR;
1300250 1 0 LOS.SRO.SIZE = LOS.SRO.SIZE - PF_S.LEN;
1300260 1 0 PT_ADDR = PF_S.PTR;
1300270 1 1 FREE PF_S;
1300280 1 0 END;

1300310 1 0 IF PF_ADDR.VPID = 0 THEN DO;
1300320 1 1 CALL SVC_ENTRY (PF_ADDR,PV_ADDR,PF_ADDR.PTR);
1300330 1 1 FREE PF_ADDR;
1300340 1 0 END;
1300350 1 0 ELSE DO;
1300360 1 1 STU = STIMER;
1300370 1 1 CALL QEVENT (LOS.LEVEL+3+3,STU,'LOOSG',0,PT_ADDR);
1300380 1 1 FREE PF_S;
1300390 1 0 END;

1300400 1 0 STU = STIMER;
1300410 1 0 IF (DEBUG.AHFFERS) THEN DO;
1300420 1 1 PUT SKIP EDIT ('AHER: TO = ',SVR.STIMER,' Ti = ',
STU,' QT = ',SVR.STIMEQ)
(A,F(I2),A,F(I2),A,F(I2));
PL/I OPTIMIZING COMPILER

AAHER: PROC (LOS,SVR,EVENT):

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>LEV</th>
<th>HT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1300450</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1300460</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1300470</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1300490</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

END;

SVR.STIMEQ = SVR.STIMEQ + STM - SVR.STIME;

SVR.STIME = STM;

END:
**PL/I OPTIMIZING COMPILER**

**ADD0: PROC (Q,PO):**

**SOURCE LISTING**

**NUMBER LISTING**

```
10 0 ADD0: PROC (Q,PO);
   /* Q -> HEAD OF QUEUE
      PO -> OBJECT TO BE ADDED */
50 1 0 DCL (Q.PQ.PO.PT) PTR;
70 1 0 DCL (ADDR.NULL) BUILTIN;
100 1 0 DCL 1 TB BASED (PT),
       2 NEXT PTR,
       2 STIME FIXED BIN (31);
140 1 0 DCL I:
150 1 0 PQ = PQ.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.PO.POS
```

**PAGE 2**
CONS: PROC (CAR, CDR) RETURNS (PTR);

%INCLUDE LIST0;

DCL LISTPT PTR;

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

******

DCL (CAR, CDR) PTR;

ALLOCATE LIST;

LIST.NEXT = CDR;

LIST.THIS = CAR;

RETURN (LISTPT);

END CONS;
PL/I OPTIMIZING COMPILER

EXECUTE: PROC (PROCNAM());

SOURCE LISTING

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>LEV</th>
<th>NT</th>
<th>CODE</th>
<th>EXECUTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>EXECUTE: PROC (PROCNAM());</td>
<td>EXEC00010</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>0</td>
<td>DCL PROCNAME CHAR(*) VAR;</td>
<td>EXEC00020</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>0</td>
<td>DCL TERM ENTRY;</td>
<td>EXEC00030</td>
</tr>
<tr>
<td>60</td>
<td>1</td>
<td>0</td>
<td>IF PROCNAME = 'TERM' THEN DO; CALL TERM: RETURN; END;</td>
<td>EXEC00040</td>
</tr>
<tr>
<td>80</td>
<td>1</td>
<td>0</td>
<td>PUT SKIP LIST ('ERROR: UNKNOWN PROCNAME. ' , PROCNAM, (EXECUTE).');</td>
<td>EXEC00050</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>0</td>
<td>STOP;</td>
<td>EXEC00060</td>
</tr>
<tr>
<td>120</td>
<td>1</td>
<td>0</td>
<td>END EXECUTE;</td>
<td>EXEC00070</td>
</tr>
</tbody>
</table>


SOURCE LISTING

NUMBER LEV NT

10 0 DEBUG: PROC;

%INCLUDE DEBUG;******************************************************

100010 1 0 DCL I DEBUG EXTERNAL STATIC,
  2 SLEEPS BIT(1) INIT ('0'B),
  2 SAHERS BIT(1) INIT ('0'B),
  2 SCHEDULERS BIT(1) INIT ('1'B),
  2 SHELLS BIT(1) INIT ('0'B),
  2 AAHERS BIT(1) INIT ('0'B);

******************************************************

200050 1 0 DEBUG.SAHERS = '1'B;
200050 1 0 DEBUG.SCHEDULERS = '1'B;
200050 1 0 DEBUG.SHELLS = '1'B;
200050 1 0 DEBUG.AAHERS = '1'B;

200100 1 0 END DEBUG;
PL/I OPTIMIZING COMPILER

SOURCE LISTING

NUMBER
LEVT

10   0  FINISH: PROC;

100010  1  0  DCL THISVP PTR EXTERNAL STATIC;  /* -> CURRENT VP, SET BY VPER */  VPI00010

100030  1  0  DCL 1 VP BASED (THISVP),  %INCLUDE VP100000 000010

00  D0SA  FIXED  BIN  (31),  /* ADDR (BOTTOM DSA) */  VPI00010
00  TOSA  FIXED  BIN  (31),  /* ADDR (TOP DSA) */  VPI00020
00  SAVE  FIXED  BIN  (31),  /* SIZE OF AREA TO BE SAVES */  VPI00030
00  SAVB   FIXED  BIN  (31),  /* -> 1ST SAVB1K */  VPI00040
00  PROCP   CHAR  (7)  VAR,  /* NAME OF TOP LEVEL PROC */  VPI00050
00  STATUS  CHAR  (12)  VAR,  /* CHAIN OF INCOMING MAIL */  VPI00070
00  MAIL   PTR,  /* CURRENT VP, SET BY VPER */  VPI00080
00  WAIT,  /* CHAIN OF INCOMING MAIL */  VPI00090
01  BOX  FIXED  BIN,  /* BOX WAITING MAIL */  VPI00100
01  MSG  PTR,  /* MSG IN WAIT.BOX */  VPI00110
00  VPID  FIXED  BIN,  /* INDEX IN THE VPST */  VPI00120
00  LEVEL  FIXED  BIN,  /* LEVEL */  VPI00130
00  VTIME  FIXED  BIN  (31)  /* VTIME */  VPI00140

%INCLUDE QEVENT:  /* NAME OF TOP LEVEL PROC */  VPI00010

500010  1  0  DCL QEVENT ENTRY (FIXED BIN,  /* ADDR */  VPI00010

ENTRY,  /* ADDR */  VPI00020

FIXED  BIN  (31));  /* ADDR */  VPI00030

500060  1  0  DCL NULL BUILTIN;  /* ADDR */  VPI00040

500070  1  0  DCL SLEEP ENTRY;  /* ADDR */  VPI00050

600010  1  0  DCL STIMER ENTRY RETURNS (FIXED BIN (31));  /* ADDR */  VPI00060

600060  1  0  DCL NULL BUILTIN;  /* ADDR */  VPI00070

600070  1  0  DCL SLEEP ENTRY;  /* ADDR */  VPI00080

600100  1  0  CALL QEVENT (VP.LEVEL+3,STIMER,'FINISH',VP.VPID,NULL);  /* ADDR */  VPI00090

600110  1  0  CALL SLEEP;  /* ADDR */  VPI00100

600120  1  0  END;  /* ADDR */  VPI00110

800140  1  0  END;

PAGE 2
PL/I OPTIMIZING COMPILER

GBER: PROC (SVR.EVENT);

SOURCE LISTING

NUMBER LEV NT

10 0 GBER: PROC (SVR.EVENT);

30 1 0 DCL 1 SVR, %INCLUDE SVR;*******************************
0 NEXT PTR,
0 STIME FIXED BIN (31), /* ACCUMULATED STIME */
0 STIMEO FIXED BIN (31),
0 RTIME FIXED BIN (15,7), /* REAL CPU TICK RATE */
0 ID FIXED BIN

200040 1 0 DCL 1 EVENT, %INCLUDE EVENT;*******************************
0 NEXT PTR, /* NEXT EVENT */
0 STIME FIXED BIN (31),
0 TYPE CHAR (12) VAR,
0 INDEX FIXED BIN (31),
0 PTR PTR

400050 1 0 DCL 1 SIM FIXED BIN (31): %INCLUDE SIMS*******************************

500060 1 0 DCL 1 PARMS EXTERNAL STATIC,
0 THRU_RATE FIXED BIN (31,7),
0 DELAY_GR_GC FIXED BIN (31),
0 DELAY_GC_DB FIXED BIN (31),
0 TERMINALS FIXED BIN;

600070 1 0 DCL STATS FILE EXTERNAL;
0 %INCLUDE PFLEVEL:*******************************

700080 1 0 DCL PT_LEVEL PTR;

700090 1 0 DCL 1 PF_LEVEL BASED (PT_LEVEL),
0 LEVEL FIXED BIN,
0 PTR PTR;

8000A0 1 0 DCL 1 PTR_MSG BASED (PT_MSG),
0 LEN FIXED BIN,
0 TYPE CHAR (12) VAR,
0 PTR PTR;

800110 %INCLUDE QEVENT;*******************************

GBO0030
GBO0020
GBO00030
SVO00010
GBO00020
SVO00020
SVO00040
SVO000050
SVO00060
GBO00030
GBO00040
EVO00010
EVO00020
EVO00030
EVO00040
EVO00050
GBO00040
GBO00050
GBO00060
GBO00070
GBO00080
PFLD0010
PFLD0020
PFLD0030
PFLD0040
PFLD0050
PFLD0060
GBO00080
GBO00090
GBO00090
GBO00100
GBO00020
GBO00030
GBO00040
GBO00050
GBO00060
GBO00070
GBO00090
DCL QEVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR, FIXED BIN (31), PTR);

********************

STM = EVENT.STIME + PARAMS.DELAY_GB_GC;

PT_LEVEL = EVENT.PTR;

PUT FILE (STATS) SKIP EDIT (' GB: STIME = ', SVR.STIME, ' MSGLEN = ', PF_MSG.LEN) (A,F(11),A,F(11));

CALL QEVENT (PF_LEVEL.LEVEL+3+1, STM, 'MSGIN', S, PT_MSG);

FREE PF_LEVEL;

SVR.STIME = EVENT.STIME;

END GBER;
PL/I OPTIMIZING COMPILER

GCER: PROC (LOS,SVR.EVENT);

SOURCE LISTING

NUMBER
LEV NT

10 0 GCER: PROC (LOS,SVR.EVENT);

30 1 0 DCL 1 LOS,
70 LEVEL FIXED BIN,
70 SNO,
71 SIZE FIXED BIN,
71 MAX FIXED BIN,
70 DBR,
71 SIZE FIXED BIN,
71 MAX FIXED BIN.
70 VPS,
71 TABLE (20),
80 BOSA 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, /* CHAIN OF INCOMING MAIL */
80Mail PTR,
80 WAIT, /* BOX WAITING 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) /* CPU TICK RATE */

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

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

800010 1 0 DCL 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;
PL/I OPTIMIZING COMPILER

GCER: PROC (LOS,SVR,EVENT):

NUMBER LEV NT

1100030 2 0 DCL 1 PF_LEVEL BASED (PT_LEVEL),
  2 LEVEL FIXED BIN,
  2 PTR PTR;

1200010 2 0 DCL PT_MSG PTR;

1200030 2 0 DCL 1 PF_MSG BASED (PT_MSG),
  2 LEN FIXED BIN,
  2 TYPE CHAR (12) VAR,
  2 PTR PTR;

1200500 2 0 DCL STM FIXED BIN (31);

1200520 2 0 STM = SVR.STIME + PARAMS.DELAY_GC_GB;

1200530 2 0 CALL QEVENT (0,STM,'MSG',0,EVENT.PTR);

1200540 2 0 PT_LEVEL = EVENT.PTR;

1200550 2 0 PT_MSG = PF_LEVEL.PTR;

1200560 2 0 STM = PARAMS.THRU_RATE * PF_.M.LEN;

1200570 2 0 SVR.STIMEQ = SVR.STIMEQ + STM;

1200580 2 0 SVR.STIME = SVR.STIME + STM;

1200600 2 0 END GCEROUT;

1200630 1 0 END GCER;

PAGE 4
### PL/I Optimizing Compiler

**QEVENT**: PROC (IQ.STIME.TYPEINDEX.PTR)1

#### SOURCE LISTING

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>LEVEL</th>
<th>NT</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>QEVENT: PROC (IQ.STIME.TYPEINDEX.PTR);</td>
</tr>
<tr>
<td>100010</td>
<td>1</td>
<td>0</td>
<td>DCL SQUEUE (0: 6) PTR EXTERNAL STATIC;</td>
</tr>
<tr>
<td>100020</td>
<td>1</td>
<td>0</td>
<td>DCL EQUEUE (0: 6) PTR EXTERNAL STATIC;</td>
</tr>
<tr>
<td>100080</td>
<td>1</td>
<td>0</td>
<td>DCL EPT PTR;</td>
</tr>
<tr>
<td>100070</td>
<td>1</td>
<td>0</td>
<td>DCL 1 EVENT BASED (EPT), 70 NEXT PTR, 70 STIME FIXED BIN (31), 70 TYPE CHAR (12) VAR, 70 INDEX FIXED BIN (31), 70 PTR PTR</td>
</tr>
<tr>
<td>200090</td>
<td>1</td>
<td>0</td>
<td>DCL IQ FIXED BIN;</td>
</tr>
<tr>
<td>200100</td>
<td>1</td>
<td>0</td>
<td>DCL STIME FIXED BIN (31);</td>
</tr>
<tr>
<td>200110</td>
<td>1</td>
<td>0</td>
<td>DCL TYPE CHAR(+VAR);</td>
</tr>
<tr>
<td>200120</td>
<td>1</td>
<td>0</td>
<td>DCL INDEX FIXED BIN (31);</td>
</tr>
<tr>
<td>200130</td>
<td>1</td>
<td>0</td>
<td>DCL PTR PTR;</td>
</tr>
<tr>
<td>200150</td>
<td>1</td>
<td>0</td>
<td>DCL ADDQ ENTRY;</td>
</tr>
<tr>
<td>200160</td>
<td>1</td>
<td>0</td>
<td>DCL NULL BUILTIN;</td>
</tr>
<tr>
<td>200180</td>
<td>1</td>
<td>0</td>
<td>ALLOCATE EVENT;</td>
</tr>
<tr>
<td>200190</td>
<td>1</td>
<td>0</td>
<td>EVENT.NEXT = NULL;</td>
</tr>
<tr>
<td>200200</td>
<td>1</td>
<td>0</td>
<td>EVENT.STIME = STIME;</td>
</tr>
<tr>
<td>200210</td>
<td>1</td>
<td>0</td>
<td>EVENT.TYPE = TYPE;</td>
</tr>
<tr>
<td>200220</td>
<td>1</td>
<td>0</td>
<td>EVENT.INDEX = INDEX;</td>
</tr>
<tr>
<td>200230</td>
<td>1</td>
<td>0</td>
<td>EVENT.PTR = PTR;</td>
</tr>
<tr>
<td>200240</td>
<td>1</td>
<td>0</td>
<td>CALL ADDQ (EQUEUE(IQ).EPT);</td>
</tr>
<tr>
<td>200260</td>
<td>1</td>
<td>0</td>
<td>END QEVENT;</td>
</tr>
</tbody>
</table>
PL/I OPTIMIZING COMPILER

SAHER: PROC (LOS,SVR,EVENT);  

SOURCE LISTING

NUMBER LEVEL  

10  0 SAHER: PROC (LOS,SVR,EVENT);  

30  1 0 DCL 1 LOS,  

70 LEVEL FIXED BIN,  

70 SQO,  

71 SIZE FIXED BIN,  

71 MAX FIXED BIN,  

70 DBR,  

71 SIZE FIXED BIN,  

71 MAX FIXED BIN,  

70 VPS,  

71 TABLE ( 20),  

80 BOSA 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 PRONAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC */VP 00050  

80 STATUS CHAR (12) VAR, /* NAME OF TOP LEVEL PROC */VP 00060  

80 MAIL PTR, /* CHAIN OF INCOMING MAIL */VP 00070  

80 WAIT, /* BOX FIXED BIN, /* BOX WAITING MAIL */VP 00080  

81 MSG PTR, /* MSG IN WAIT.OOX */VP 00090  

80 VPID FIXED BIN, /* INDEX IN THE VPST */VP 00100  

80 LEVEL FIXED BIN, /* LEVEL */VP 00110  

80 ATIME FIXED BIN (31), /* REAL CPU TICK RATE */VP 00120  

80 RRATE FIXED BIN (15.7). /* REAL CPU TICK RATE */VP 00130  

800040 1 0 DCL 1 SVR,  

70 NEXT PTR, /* NEXT PTR */VP 00010  

70 STIME FIXED BIN (31), /* ACCUMULATED STIME */SVR00020  

70 STIME FIXED BIN (31), /* ACCUMULATED STIME */SVR00030  

70 RTIME FIXED BIN (31), /* ACCUMULATED STIME */SVR00040  

70 RRATE FIXED BIN (15.7), /* REAL CPU TICK RATE */SVR00050  

70 ID FIXED BIN, /* REAL CPU TICK RATE */SVR00060  

700050 1 0 DCL 1 EVENT, /* NEXT EVENT */EVE00010  

70 NEXT PTR, /* NEXT EVENT */EVE00020  

70 STIME FIXED BIN (31), /* NEXT EVENT */EVE00030  

70 TYPE CHAR (12) VAR, /* NEXT EVENT */EVE00040  

70 ID FIXED BIN (31), /* NEXT EVENT */EVE00050  

70 PTR PTR /* NEXT EVENT */EVE00060  

700070 1 0 DCL (THIS,THIS,SVR) PTR EXTERNAL STATIC;  

700080 1 0 DCL SVR FIXED BIN (31);  

800010 1 0 DCL 1 PASS EXTERNAL STATIC,  

2 THRU_RATE FIXED BIN (31.7);  

800030
PL/I OPTIMIZING COMPILER

SAHER: PROC (LOS, SVR, EVENT);

NUMBER LEVFNT

2 DELAY .GB_OC FIXED BIN (31).
2 DELAY .GB_OC FIXED BIN (31).
2 TERMINALS FIXED BIN;
900010 10 DCL I DEBUG EXTERNAL STATIC,
2 SLEEP (1) INIT ('0'8),
2 SAHER (1) INIT ('0'8),
2 SCHEDULER (1) INIT ('0'8),
2 SHELTS (1) INIT ('0'8),
2 AHERR (1) INIT ('0'8);
900020 10 DCL (ADDR, NULL) BUILTIN;
1000010 10 DCL SLEPER ENTRY RETURNS (FIXED BIN (31));
1000020 10 DCL TERMINAL ENTRY;
1000030 10 THISLOS = ADDR (LOS);
1000040 10 THISVR = ADDR (SVR);
1000050 10 IF (DEBUG .SAHERS) THEN DO;
1000060 10 PUT SKIP EDIT ('SAHER: LEVEL = ', LOS. LEVEL,
1000070 10 'E = (\', EVENT .TYPE', EVENT .STIME, ')')
1000080 10 'A, A (12), P (11), A11';
1000090 10 END;
10000100 10 IF (EVENT .TYPE = 'LOGMSG') THEN DO;
10000200 10 CALL MAILMAN (LOS. VPS, EVENT .PTR);
10000300 10 IF (LOS. VPS. TABL(EVENT .INDEX). STATUS = 'RUNNING') THEN
10000400 10 /* DO NOT CHANGE MASSECT */
10000500 10 END;
10000600 10 ELSE IF (EVENT .TYPE = 'SYNC') THEN DO;
10000700 10 IF (LOS. VPS. TABL(EVENT .INDEX). STATUS = 'RUNNING') THEN
10000800 10 /* WAIT. BOX WAS SET EARLIER */
10000900 10 END;
10000100 10 ELSE IF (EVENT .TYPE = 'WAIT') THEN DO;
10000200 10 LOS. VPS. TABL(EVENT .INDEX). STATUS = 'BLOCKED';
10000300 10 /* WAIT. BOX WAS SET EARLIER */
10000400 10 END;
10000500 10 ELSE IF (EVENT .TYPE = 'FINISH') THEN DO;
10000600 10 /* WAIT. BOX WAS SET EARLIER */
10000700 10 END;
10000800 10 CALL ENTER (LOS. VPS);
10000900 10 /* DO NOT SCHEDULE */
10000100 10 CALL CHECK_MAIL (LOS. VPS);
10000200 10 CALL SCHEDULER (LOS. VPS);
10000300 10 STM = STIME;
PL/I OPTIMIZING COMPILER

SAHER: PROC (LOS.SVR, EVENT);

NUMBER LEV NT

1000510 1 0 SRV.STIME = SRV.STIME + STM - SRV.STIME;
1000520 1 0 SRV.STIME = STM;
1000540 1 0 RETURN;

1000570 1 0 MAILMAN: PROC (VPS, PT_ADDR);

1000590 2 0 DCL 1 VPS, 2 TABLE ( 20 ),
80 BOSA 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,
80 PROCHAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC
80 STATUS CHAR (12) VAR,
80 MAIL PTR,
80 BOX FIXED BIN, /* BOX WAITING MAIL
81 MSG PTR, /* MSG IN WAIT BOX
80 VPID FIXED BIN, /* INDEX IN THE VPST
80 VTIME FIXED BIN, /* LEVEL

1200010 2 0 DCL PT_ADDR PTR;
1200030 2 0 DCL 1 PF_ADDR BASED (PT_ADDR),
2 VPID FIXED BIN,
2 BOXID FIXED BIN,
2 PTR PTR;
1300010 2 0 DCL BOXPT PTR;
1300030 2 0 DCL 1 BOX BASED (BOXPT),
2 NEXT PTR,
2 ID FIXED BIN,
2 LIST PTR;
1300070 2 0 DCL VPID FIXED BIN;
1400010 2 0 DCL CONS ENTRY (PTR, PTR) RETURNS (PTR);
1400060 2 0 VPID = PF_ADDR.VPID;
1400070 2 0 IF (VPS.TABLE(VPID).STATUS = 'VOID') THEN DO;
1400080 2 1 PUT SKIP EDIT ('WARNING: ILLEGAL ADDRESS ',LOS.LEVEL, 
1400090 2 1 '/' ,VPID) (A,F(2),A,F(2));
1400070 2 1 FREE PF_ADDR;
1400070 2 1 /* CANNOT FREE VARIABLE FORMAT MSG */
1400070 2 1 RETURN;
1400070 2 1 END;
1400070 2 1 BOXPT = VPS.TABLE(VPID).MAIL;

PGM SW排出
PL/I OPTIMIZING COMPILER

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;

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 BOXPT FIXED BIN (31).
80 YOSA FIXED BIN (31).
80 SAVESIZE FIXED BIN (31).
80 SAFE PTR.
80 PROCNAME VAR.
80 STATUS VAR.
80 MAIL PTR.
80 WAIT.
80 BOX FIXED BIN.
80 MSG PTR.
80 VPID FIXED BIN.
80 LEVEL FIXED BIN.
80 VTIME FIXED BIN (31).

1600010  2 0  DCL BOXPT PTR;
1600020  2 0  DCL 1 BOX BASED (BOXPT),
1600030  2 0  2 NEXT PTR,
1600040  2 0  2 ID FIXED BIN,
1600050  2 0  2 LIST PTR;
1700010  2 0  DCL LISTPTR PTR;
1700020  2 0  DCL 1 LIST BASED (LISTPTR),
1700030  2 0  2 NEXT PTR,
1700040  2 0  2 THIS PTR;
1701010  2 0  DCL IV FIXED BIN;

PAGE 8
PL/I OPTIMIZING COMPIlER

SCHEDULER: PROC (LOS, SVR, EVENT);

1701020 2 0 DCL PT PTR;
1701040 2 0 DO IV = 1 TO 20;
1701050 2 1 IF VPS.TABLE(IV).STATUS = 'BLOCKED' THEN DO;
1701060 2 2 BOXPT = VPS.TABLE(IV).MAIL;
1701070 2 2 DO WHILE (BOXPT = NULL);
1701080 2 3 IF BOX.ID = VPS.TABLE(IV).WAIT.BOX THEN DO;
1701090 2 4 IF BOX.LIST = NULL THEN DO;
1701100 2 5 LISTPT = BOX.LIST;
1701110 2 5 PT = LIST.THIS;
1701120 2 5 BOX.LIST = LIST.NEXT;
1701130 2 5 FREE LIST;
1701140 2 5 VPS.TABLE(IV).WAIT.MSG = PT;
1701150 2 5 VPS.TABLE(IV).STATUS = 'RUNNABLE';
1701160 2 5 GOTO NEXT_VP;
1701170 2 5 END;
1701180 2 4 END;
1701190 2 3 BOXPT = BOX.NEXT;
1701200 2 3 END;
1701210 2 2 END;
1701220 2 1 NEXT_VP: END;
1701240 2 0 END CHECK_MAIL;

1701270 1 0 SCHEDULER: PROC (VPS);
1701290 2 0 DCL I VPS. 2 TABLE[20],
  80 BDSA FIXED BIN (31), /* ADDR (BOTTOMDSA)
  80 TDSA FIXED BIN (31), /* ADDR (TOPDSA)
  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 WAITING MAIL
  80 VTIME FIXED BIN, /* MSG IN WAIT.BOX
  80 VTIME FIXED BIN, /* INDEX IN THE VPST
  80 LEVEL FIXED BIN, /* LEVEL
  80 VTIME FIXED BIN (31)

 1801310 2 0 DCL (VTM, VTM0) FIXED BIN (31);
1801320 2 0 DCL YOUNGEST FIXED BIN;
1801330 2 0 DCL IV FIXED BIN;
1801350 2 0 YOUNGEST = 0;
1801360 2 0 VTM0 = 2147483647;
DO IV = 1 TO 20;
  IF (VPS.TABLE(IV).STATUS = ‘RUNNABLE’) THEN DO;
    IF VIM < VIMO THEN DO;
      YOUNGEST = IV;
    END;
  END;
IF (DEBUG.SCHEDULERS) THEN GO;
  PUT SKIP EDIT (I SCHEDULER; YOUNGEST = ‘.*.YOUNGEST’) (A.F(3))
END;
IF (YOUNGEST > 0) THEN DO;
  CALL VPER (VPS.TABLE(YOUNGEST));
END;

DCL 1 VPS, 2 TABLE (20),
  80 DSA FIXED BIN (31), /* ADDR (BOTTOM DSA)
  80 TOPD FIXED BIN (31), /* ADDR (TOP DSA)
  80 SAVESIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED
  80 SAFE PTR, /* -> 1ST SAVBLK
  80 PROCHKE CHAR (7) VAR, /* NAME OF TOP LEVEL PROC
  80 STATUS CHAR (12) VAR,
  80 MAIL PTR, /* CHAIN OF INCOMING MAIL
  80 BOX FIXED BIN,
  80 MSG FIXED BIN, /* BOX AWAITING MAIL
  80 MSG PTR, /* MSG IN WAIT.BOX
  80 VID fixed BIN,
  80 LEVEL FIXED BIN,
  80 VTIME FIXED BIN (31)

DCL 1 PARAMS EXTERNAL STATIC,
  2 THRU_RATE FIXED BIN (31,7),
  2 DELAY_GB_CC FIXED BIN (31),
  2 DELAY_CC_GB FIXED BIN (31),
  2 TERMINALS FIXED BIN;
DCL IV FIXED BIN;
DCL MSGLEN FIXED BIN;
DCL MSGPT PTR;
DCL MSG BASED (MSGPT),
PL/I OPTIMIZING COMPILER

SAHER: PROC (LOS.SVR.EVENT):

NUMBER LEV NT

   2 LEN FIXED BIN.
   2 STR CHAR (MSGLEN REFER (LEK));
   200010 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
VP500010
SAH01670
SAH01680
SAH01690
SAH01700
SAH01710
SAH01720
SAH01730
SAH01740
SAH01750
SAH01760
SAH01770
SAH01780
SOURCE LISTING

NUMBER LEVEL

10 0 SEND: PROC (LEVEL, VPID, BOXID, TYPE, LEN, MPT);

30 1 0 DCL (LEVEL, VPID, BOXID, LEN) FIXED BIN;

40 1 0 DCL (TYPE) CHAR(*) VAR;

50 1 0 DCL (MPT, PT) PTR;

%INCLUDE PFADDR;

100 0 1 0 DCL PT-ADDR PTR; PFA0010

%INCLUDE PF;

300 0 1 0 DCL PT-S PTR; PFS0010

%INCLUDE PFMSG;

400 0 1 0 DCL PT-MSG PTR; PFMO010

%INCLUDE PFLEVEL;

500 0 1 0 DCL THISVP PTR EXTERNAL STATIC; /* - CURRENT VP, SET BY VPER */ VPX0010

%INCLUDE VPX;

600 0 1 0 DCL THISVP PTR EXTERNAL STATIC; /* - CURRENT VP, SET BY VPER */ VPX0010

%INCLUDE VP;

800 0 1 0 DCL TYPE CHAR(*) VAR;

900 0 1 0 DCL LEN FIXED BIN, 2 PTR PTR;

%INCLUDE PFA;

1000 0 1 0 DCL PT-ADDR PTR; PFA0010

%INCLUDE PF;

3000 0 1 0 DCL PT-S PTR; PFS0010

%INCLUDE PFMSG;

4000 0 1 0 DCL PT-MSG PTR; PFMO010

%INCLUDE PFLEVEL;

5000 0 1 0 DCL THISVP PTR EXTERNAL STATIC; /* - CURRENT VP, SET BY VPER */ VPX0010

%INCLUDE VPX;

6000 0 1 0 DCL VP BASED (THSVIP), XINCLUDE VP; /* INCLUDE */ VPX0010

8000 0 1 0 DCL TYPE CHAR(*) VAR,
OPTIMIZING COMPILER

SEND: PROC (LEVEL, VPID, BOXID, TYPE, LEN, MPT)

NUMBER LEV NT

- SAFE PTR,
- PROCNAME CHAR (7) VAR,
- STATUS CHAR (13) VAR,
- MAIL PTR,
- WAIT,
- BOX FIXED BIN,
- MSG PTR,
- VPID FIXED BIN,
- LEVEL FIXED BIN,
- TIME FIXED BIN (31)

*******************************************************************************

**INCLUDE DEVENT;**************************************************

90010 1 0 DCL DEVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR,
QEVENT ENTRY (FIXED BIN (31), PTR);
QEVENT ENTRY;

***************

**INCLUDE STIMER;**************************************************

100010 1 0 DCL STIMER ENTRY (FIXED BIN (31));
STIMER ENTRY;

***************

**INCLUDE SVCER;**************************************************

110010 1 0 DCL SVCER ENTRY (FIXED BIN, PTR);

***************

110010 1 0 DCL SVCER ENTRY;

1100170 1 0 IF (LEVEL = VP.LEVEL) & (VPID = 0) THEN DO;
1100190 1 1 CALL SVCER (BOXID, MPT);
1100200 1 1 RETURN;
1100210 1 1 END;

1100240 1 0 ALLOCATE PF_ADDR;
1100250 1 0 PF_ADDR, BOXID = BOXID;
1100260 1 0 PF_ADDR, VPID = VPID;
1100270 1 0 PF_ADDR, MPT = MPT;

1100290 1 0 IF (LEVEL = VP.LEVEL) THEN DO;
1100300 1 1 CALL DEVENT (VP.LEVEL+3, STIMER, 'LOGMSG', 0, PT_ADDR);
1100310 1 1 END;

1100330 1 0 ELSE DO: /* LEVEL = VP.LEVEL */
1100340 1 1 PT = PT_ADDR;
1100350 1 1 IF (TYPE = '5') THEN DO;
1100360 1 2 ALLOCATE PF_S;
1100370 1 2 PF_S, LEN = LEN;
1100380 1 2 PF_S_PTR = PT_ADDR;
1100390 1 2 PT = PF_S;

PL/1 OPTIMIZING COMPILER
SEND: PROC (LEVEL, VPID, BOXID, TYPE, LEN, MPT):

NUMBER LEV NT

- SAFE PTR,
- PROCNAME CHAR (7) VAR,
- STATUS CHAR (13) VAR,
- MAIL PTR,
- WAIT,
- BOX FIXED BIN,
- MSG PTR,
- VPID FIXED BIN,
- LEVEL FIXED BIN,
- TIME FIXED BIN (31)

*******************************************************************************

**INCLUDE DEVENT;**************************************************

90010 1 0 DCL DEVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR,
QEVENT ENTRY (FIXED BIN (31), PTR);
QEVENT ENTRY;

***************

**INCLUDE STIMER;**************************************************

100010 1 0 DCL STIMER ENTRY (FIXED BIN (31));
STIMER ENTRY;

***************

**INCLUDE SVCER;**************************************************

110010 1 0 DCL SVCER ENTRY (FIXED BIN, PTR);

***************

110010 1 0 DCL SVCER ENTRY;

1100170 1 0 IF (LEVEL = VP.LEVEL) & (VPID = 0) THEN DO;
1100190 1 1 CALL SVCER (BOXID, MPT);
1100200 1 1 RETURN;
1100210 1 1 END;

1100240 1 0 ALLOCATE PF_ADDR;
1100250 1 0 PF_ADDR, BOXID = BOXID;
1100260 1 0 PF_ADDR, VPID = VPID;
1100270 1 0 PF_ADDR, MPT = MPT;

1100290 1 0 IF (LEVEL = VP.LEVEL) THEN DO;
1100300 1 1 CALL DEVENT (VP.LEVEL+3, STIMER, 'LOGMSG', 0, PT_ADDR);
1100310 1 1 END;

1100330 1 0 ELSE DO: /* LEVEL = VP.LEVEL */
1100340 1 1 PT = PT_ADDR;
1100350 1 1 IF (TYPE = '5') THEN DO;
1100360 1 2 ALLOCATE PF_S;
1100370 1 2 PF_S, LEN = LEN;
1100380 1 2 PF_S_PTR = PT_ADDR;
1100390 1 2 PT = PF_S;
PL/I OPTIMIZING COMPILER

SEND: PROC (LEVEL, VPID, BOXID, TYPE, LEN, MPT):

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>LEVEL</th>
<th>NT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100400</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1100410</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1100420</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1100430</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1100440</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1100450</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1100460</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1100470</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1100480</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1100490</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1100500</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1100510</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

END SEND;
PL/I OPTIMIZING COMPILER

SOURCE LISTING

NUMBER LEVT NT

10 0 SHELL: PROC OPTIONS (MAIN):

30 4 DCL PLIXOPT CHAR(40) VAR EXTERNAL STATIC

INIT ('ISASIZE(-50K)');

100010 1 DCL SQUEUE (0: 6) PTR EXTERNAL STATIC;

100020 1 DCL EQUEUE (0: 6) PTR EXTERNAL STATIC;

100080 1 DCL LOSES (0: 1) STATIC,

70 LEVEL FIXED BIN,

70 END.

71 SIZE FIXED BIN,

71 MAX FIXED BIN,

70 DBG.

71 SIZE FIXED BIN,

71 MAX FIXED BIN,

70 DATA.

71 SIZE FIXED BIN,

71 MAX FIXED BIN,

70 VPS,

71 TABLE ( 20),

60 DBSA FIXED BIN (31), /* ADDR (BOTTOM DSA) */

60 TOSA FIXED BIN (31), /* ADDR (TOP DSA) */

60 SAVESIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED */

60 SAFE PTR,

60 PROCNAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC */

60 STATUS CHAR (12) VAR,

60 MAIL PTR,

60 WAIT,

40 BOX FIXED BIN, /* BOX WAITING MAIL */

80 MSG PTR, /* MSG IN THE VPST */

80 LEVEL FIXED BIN, /* LEVEL */

70 VTIME FIXED BIN (31)

400090 1 DCL (EPT.SPT) PTR;

400100 1 DCL EVENT BASED (EPT),

70 NEXT PTR, /* NEAT EVENT */

70 STIME FIXED BIN (31),

70 TYPE CHAR (12) VAR,

70 INDEX FIXED BIN (31),

70 PTR PTR

500110 1 DCL SVR BASED (SPT),

70 NEXT PTR,

70 STIME FIXED BIN (31), /* ACCUMULATED STIME */

70 RTIME FIXED BIN (31),

70 RATE FIXED BIN (15.7), /* REAL CPU TICK RATE */

70 ID FIXED BIN
NUMBER LEV NT

```
600120 1 0 DCL (1.LEVEL,EARLIEST) FIXED BIN;
600130 1 0 DCL (STM,STMO) FIXED BIN (31);
600140 1 0 DCL DAT CHAR(6);
600150 1 0 DCL TIM CHAR(9);
600160 1 0 DCL (YR,MO,DY,HR,MN) CHAR(2);
600170 1 0 DCL STATS FILE OUTPUT EXTERNAL;
700010 1 0 DCL 1 DEBUG EXTERNAL STATIC.
700200 1 0 DCL (ADDR,NULL,MAX,MOD,DATE,TIME,SUBSTR) BUILTIN;
800010 1 0 DCL RTIMER ENTRY RETURNS (FIXED BIN (31));
900010 1 0 DCL DEVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR, FIXED BIN (31), PTR);
900210 1 0 DCL (GBER,GER,SAHER,AAMER) ENTRY;
900270 1 0 OPEN FILE (STATS) TITLE ('STATS');
900290 1 0 TIM = TIME;
900300 1 0 YR = SUBSTR(DAT,1,2);
900310 1 0 NO = SUBSTR(DAT,3,2);
900320 1 0 DY = SUBSTR(DAT,5,2);
900330 1 0 HR = SUBSTR(TIM,1,2);
900350 1 0 PUT FILE (STATS) SKIP EDIT ('***',YR,'/',MO,'/',DY,' ',
         HR,':',MN);
900360 1 0 PUT FILE (STATS) SKIP EDIT ('*** PARAMETERS + (A):'
         (A),
900400 1 0 CALL DEBUG;
900430 1 0 CALL SET_LOS;
900450 1 0 CALL SET_P;
900460 1 0 PUT FILE (STATS) SKIP (3) EDIT ('*** SIMULATION:('
         (A),
900510 1 0 LOOP:
900520 1 0 TIM = 2147483647;
```
PL/I OPTIMIZING COMPILER

SHELL: PROC OPTIONS (MAIN):

NUMBER LEVT

900540 1 0  DO I = 0 TO 6;
900550 1 1  IF (EQUEUE(I) = NULL) THEN DO;
900560 1 2  EPT = EQUEUE(I);
90057O 1 2  SPT = SQUEUE(I);
900580 1 2  STM = MAX (EVENT.STIME,SVR.STIME);
900590 1 2  IF (STM < STM) THEN DO;
900600 1 3  EARLIEST = 1;
900610 1 3  END;
900620 1 3  END;
900630 1 2  END;
900640 1 1  END;
900660 1 0  IF (DEBUG.SHELLS) THEN DO;
900670 1 1  PUT SKIP EDIT ('SHELL: EARLIEST = ' , EARLIEST) (A,F(2)) I;
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.NEXTI;
900740 1 0  LEVEL = (EARLIEST-1)/3;
900750 1 0  SVR.STIME = STM;
900760 1 0  SVR.RTIME = RTIME;
900770 1 0  IF (EARLIEST = 0) THEN DO;
900780 1 1  CALL GDER (SVR,EVENT);
900790 1 1  END;
900800 1 0  ELSE IF (MOD(EARLIEST,3) = 1) THEN DO: /* QA */
900810 1 1  CALL GCEN (LOSES(LEVEL),SVR,EVENT);
900820 1 1  END;
900830 1 1  END;
900840 1 0  ELSE IF (MOD(EARLIEST,3) = 2) THEN DO: /* AAM */
900850 1 1  CALL AHER (LOSES(LEVEL),SVR,EVENT);
900860 1 1  END;
900870 1 0  ELSE DO: /* SAN */
900880 1 1  CALL SAHER (LOSES(LEVEL),SVR,EVENT);
900890 1 1  END;
900900 1 1  FREE EVENT;
900910 1 0  IF (EARLIEST = 0) & (MOD(EARLIEST,3) = 1) THEN DO;
900920 1 1  CALL ADEC (SVR,EVENT);
900930 1 1  SVR.NEXT = NULL;
900940 1 1  CALL ADD (SQUE(EARLIEST),SPT);
900950 1 1  END;
900960 1 1  END;
900980 1 0  GOTO LOOP;
901000 1 0  EXIT;

PUT FILE (STATS) SKIP (3) EDIT (' *** STATISTICS***') (A);
PL/I OPTIMIZING COMPILER

SHELL PROC OPTIONS (MAIN):

NUMBER LEVELS

90100 10 PUT FILE (STATS) SKIP (2):
901040 10 CALL REPORT:
901060 10 CLOSE FILE (STATS):
901060 10 RETURN:

901090 10 SET_LOS: PROC:
901100 20 DCL (L,NV) FIXED BIN:

901130 20 DO L = 0 TO 1:
901140 21 LOSES(L).LEVEL = L;
901150 21 LOSES(L).SQZ.MAX = 0;
901160 21 LOSES(L).SQZ.MAX = 0;
901170 21 LOSES(L).OBS.MAX = 0;
901190 21 DO IV = 1 TO 20:
901200 22 LOSES(L).VPS.TABLE(IV).LEVEL = L;
901210 22 LOSES(L).VPS.TABLE(IV).VPOP = IV;
901220 22 LOSES(L).VPS.TABLE(IV).STATUS = 'VOID';
901230 22 END:
901240 21 END:

901250 20 END SET_LOS;

901290 10 SET_Q: PROC:
901310 20 DCL (Q,L,NP,IP) FIXED BIN:
901320 20 DCL (Rerate) FIXED BIN (15.7):

901340 20 DO IQ = 0 TO 6:
901350 21 EQUEUE(IQ) = NULL;
901360 21 SQUEUE(IQ) = NULL;
901370 21 END:

901370 20 DO L = 0 TO 1:
901400 21 PUT SKIP EDIT (' LEVEL ', L, ' ') (A,F(2),A);
901410 21 PUT FILE (STATS) SKIP EDIT (' LEVEL ', L, ' ') (A,F(2),A);
901420 21 PUT SKIP LIST (' NO. OF PROCESSORS, RRATE> '):
901430 21 GET LIST (NP, RRATE);
901440 21 PUT FILE (STATS) DATA (NP, RRATE);
901450 21 DO (P = 1 TO NP):
901460 22 CALL NEW_SVR(RRATE,IP,SQUEUE(L+3));
901470 22 CALL NEW_SVR(RRATE,IP,SQUEUE(L+3));
901480 22 END:
901490 21 CALL NEW_SVR(RRATE,IP,SQUEUE(L+3));
901500 21 END: /* L */
PL/I OPTIMIZING COMPILER  SHELL: PROC OPTIONS (MAIN):

NUMBER  LEV  NT
 901520  2  0 CALL NEW_SVR(1,1,SQUEUE(0));
 901530  2  0 CALL GEVENT (3,0,'INIT',0,0);  /* END */
 901540  2  0

901590  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 SVRPTR PTR;
 901640  3  0 DCL 1 SVR BASED (SVRPTR),
     NEXT PTR,
     STIME FIXED BIN (31), /* ACCUMULATED STIME
     RTIME FIXED BIN (31), /* REAL CPU TICK RATE
     RRATE FIXED BIN (15,7),
     ID FIXED BIN
     ;
 1001650  3  0 ALLOCATE SVR;
 1001670  3  0 SVR.STIME = 0;
 1001690  3  0 SVR.RRATE = 0;
 1001700  3  0 SVR.ID = ID;
 1001720  3  0 WHERE = SVRPTR;
 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,
     THRU_RATE FIXED BIN (31,7),
     DELAY_GC_GB FIXED BIN (31),
     TERMINALS FIXED BIN;
1101840  2  0 PUT SKIP LIST ('THRU_RATE,GB_GC,GB_GB,TERMINALS')
1101850  2  0 GET LIST (PARAMS,THRU_RATE,PARAMS,DELAY_GC_GB,
     PARAMS,DELAY_GC_GB,PARAMS,TERMINALS);
1101870  2  0 PUT FILE (STATS) SKIP EDIT (' ') (A);
1101880  2  0 PUT FILE (STATS) DATA (PARAMS);
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;
1101950  2  0   DCL I SVR BASED (SVRPT),
   TO NEXT PTR,
   TO STIME FIXED BIN (31), /* ACCUMULATED STIME
   TO STIME FIXED BIN (31),
   TO RTIME FIXED BIN (31), /* REAL CPU TICK RATE
   TO ID FIXED BIN

1201970  2  0   DCL (I,L) FIXED BIN:
1201990  2  0   DCL TITLE(3) CHAR(3) STATIC INIT ('GC','AAN','SAH');

1202000  2  0   DO L = 0 TO 1:
1202020  2  1   PUT FILE (STATS) SKIP EDIT ('LEVEL ',L,'I ') (A,F(2),A);
1202030  2  1   PUT FILE (STATS) SKIP EDIT ("", SVRPT):
   * SQG.MAX =",LOSES(L).SQG.MAX,
   * SRQ.MAX =",LOSES(L).SRQ.MAX,
   * DBB.MAX =",LOSES(L).DBB.MAX)
   (A,A,F(11),A,F(11));

1202200  2  0   END:

1202200  2  0   END REPORT;
1202200  2  0   END SHELLI.
SOURCE LISTING

NUMBER  LEVEL  ATTRIBUTE  VALUE

10  1  0  SLEEP:  PROC;

30  1  0  DCL (THIS,NEXT) PTR;

40  1  0  DCL (NULL,ADDR,MIN) BUILTIN;

50  1  0  DCL (NSIZE,SIZE,NEW_BLK,OLD_BLK,DIFF_BLK) FIXED BIN (31);

60  1  0  DCL SAVE_ADDR FIXED BIN (31);

100 10  1  DCL THISVP PTR EXTERNAL STATIC; /* -> CURRENT VP, SET BY VPER */

100 30  1  DCL 1 VP BASED (THISVP),

   80 DSA FIXED BIN (31), /* ADDR (BOTTOM DSA) */

   80 MAXAD FIXED BIN (31), /* ADDR (TOP DSA) */

   80 SIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED */

   80 VTIME FIXED BIN (31)

400 70  1  DCL NEWSAVBLK PTR;

400 90  1  DCL 1 SAVBLK BASED (NEWSAVBLK),

   2 NEXT PTR, /* NEXT SAVED ITEM */

   2 ADDR  FIXED BIN (31), /* ADDR OF SAVED ITEM */

   2 SIZE  FIXED BIN (31), /* SIZE OF SAVED ITEM */

   2 AREA ( 250) FIXED BIN (31);

500 10  1  DCL 1 DEBUG EXTERNAL STATIC,

   2 SLEEP B[BIT(1) INIT ('O'B),

   2 SAME B[BIT(1) INIT ('O'B),

   2 SCHED B[BIT(1) INIT ('O'B),

   2 SHEL S B[BIT(1) INIT ('O'B),

   2 AAME S B[BIT(1) INIT ('O'B);

600 10  1  DCL DSA FIXED BIN (31),

   1 NAB  FIXED BIN (31),

   1 SEG  FIXED BIN (31),

   1 EOS  FIXED BIN (31),

PL/I OPTIMIZING COMPILER  SLEEP:  PROC;  PAGE 2
PL/I OPTIMIZING COMPILER  SLEEP: PROC;  

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>LEV</th>
<th>NT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>700530</td>
<td>1</td>
<td>1</td>
<td>END;</td>
</tr>
<tr>
<td>700550</td>
<td>1</td>
<td>0</td>
<td>VP.SAVESIZE = NSIZE;</td>
</tr>
<tr>
<td>700560</td>
<td>1</td>
<td>0</td>
<td>VP.TOSA = DSA;</td>
</tr>
<tr>
<td>700580</td>
<td>1</td>
<td>0</td>
<td>THIS = VP.SAFE;</td>
</tr>
<tr>
<td>700590</td>
<td>1</td>
<td>0</td>
<td>SAVE_ADDR = VP.BOSA;</td>
</tr>
<tr>
<td>700600</td>
<td>1</td>
<td>0</td>
<td>DO WHILE (THIS = NULL);</td>
</tr>
<tr>
<td>700610</td>
<td>1</td>
<td>1</td>
<td>THIS -&gt; SAVBLK.SIZE = MIN(1000,NSIZE);</td>
</tr>
<tr>
<td>700620</td>
<td>1</td>
<td>1</td>
<td>THIS -&gt; SAVBLK.ADDR = SAVE_ADDR;</td>
</tr>
<tr>
<td>700630</td>
<td>1</td>
<td>1</td>
<td>NSIZE = NSIZE - 1000;</td>
</tr>
<tr>
<td>700640</td>
<td>1</td>
<td>1</td>
<td>SAVE_ADDR = SAVE_ADDR + 1000;</td>
</tr>
<tr>
<td>700650</td>
<td>1</td>
<td>1</td>
<td>THIS = THIS -&gt; SAVBLK.NEXT;</td>
</tr>
<tr>
<td>700660</td>
<td>1</td>
<td>1</td>
<td>END;</td>
</tr>
<tr>
<td>700680</td>
<td>1</td>
<td>0</td>
<td>CALL SVSTK (VP.SAFE,VP.BOSA);</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>/* CONTROL SHOULD COME HERE AFTER RTSTK */</td>
</tr>
<tr>
<td>700720</td>
<td>1</td>
<td>0</td>
<td>END SLEEP;</td>
</tr>
</tbody>
</table>
PL/I OPTIMIZING COMPILER

STIMER: PROC RETURNS (FIXED BIN(31));

SOURCE LISTING

NUMBER LEV NT

10  0 STIMER: PROC RETURNS (FIXED BIN(31));

%INCLUDE SVR;******************************************************************************ST100010
100010 1 0 DCL THISSVR PTR EXTERNAL STATIC: /*-> CURRENT SVR, SET BY SAHER */ SVR000010
100030 1 0 DCL 1 SVR BASED (THISSVR), %INCLUDE SVR;******************************************************************************SVR000020
    70 NEXT PTR,
    70 STIME FIXED BIN (31), /* ACCUMULATED STIME */ SVR000030
    70 STIMEQ FIXED BIN (31), /* REAL CPU TICK RATE */ SVR000040
    70 RTIME FIXED BIN (15,7), /* REAL CPU TICK RATE */ SVR000050
    70 ID FIXED BIN
******************************************************************************1

******************************************************************************

400040 1 0 DCL RT FIXED BIN (31);

%INCLUDE RTIMER;******************************************************************************ST100050
500010 1 0 DCL RTIMER ENTRY RETURNS (FIXED BIN (31));

******************************************************************************

500070 1 0 RT = RTIMER - SVR.RTIME;
500090 1 0 RT = RT + SVR.RRATE;
500090 1 0 RETURN (RT * SVR.STIME);
500110 1 0 END;
SOURCE LISTING

NUMBER  LEV  NT
10   0  SVCER: PROC (SVC.PT);
30   1 0  DCL SVC FIXED BIN;
40   1 0  DCL PT PTR;
100010  1 0  DCL THISVP PTR EXTERNAL STATIC;
100030  1 0  DCL 1 VP BASED (THISVP),
        80 BDSA FIXED BIN (31),
        80 TOSA 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)
100010  1 0  DCL THISLOS PTR EXTERNAL STATIC;
100030  1 0  DCL 1 LOS BASED (THISLOS),
        70 LEVEL FIXED BIN,
        70 SQO,
        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 TOSA 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,
PL/I OPTIMIZING COMPILER  
SVCER: PROC (SVC, PT);  

NUMBER LEV NT  

80 VTIME FIXED BIN (31)  

800010 1 0 DCL THISSVR PTR EXTERNAL STATIC; /* CURRENT SVR, SET BY SANER */  

800030 1 4 DCL 1 SVR BASED (THISSVR),  
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  

100010 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 PT_SVC PTR;  

1200010 1 0 DCL VPSTART ENTRY (/* VP */ PTR, CHAR(*) VAR, PTR);  

1200160 1 0 IF (SVC <= 0) | (SVC > 1) THEN DO;  
1200170 1 1 PUT SKIP LIST ('ERROR: ILLEGAL SVC.');  
1200190 1 1 PUT DATA (SVC);  
1200200 1 1 STOP;  

1200220 1 0 GOTO SVCS (SVC);  

1200240 1 0 START: /* SVC = 1 */  

DO IV = 1 TO 20:  
1200270 1 1 IF LOS.VPS.TABLE(IV).STATUS = 'VQID' THEN GOTO START;  

1200290 1 1 END;  

1200300 1 0 STOP;  

1200320 1 0 START:  

PT_SVC = PT;  

1200330 1 0 CALL VPSTART (LOS.VPS.TABLE(IV), PF_SVC.SVC, PF_SVC.PTR);  

1200360 1 0 FREE PF_SVC;  

1200370 1 0 RETURN;  

1200390 1 0 END;
SOURCE LISTING

NUMBER  LEV  NT

10  0  SYNC:     PROC:  SYN00010
100010  1  0  DCL  THISVP  PTR  EXTERNAL  STATIC;  /*  -->  CURRENT  VP,  SET  BY  VPER */ SYN00030
100030  1  0  DCL  1  VP  BASED  (THISVP),  INCLUDE  VPX:*.*.4...4.4.........**.e..**.**e....tSYN00030
00  BOSA  FIXED  BIN  (31),  /*  ADDR  (BOTTOM  DSA)  */ VP  00010
00  TOSA  FIXED  BIN  (31),  /*  ADDR  (TOP  DSA)  */ VP  00020
00  SAVESIZE  FIXED  BIN  (31),  /*  SIZE  OF  AREA  TO  BE  SAVED  */ VP  00030
00  SAFE  PTR,  /*  -->  1ST  SAVEBLK  */ 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  00080
80  WAIT,  VP  00090
B1  BOX  FIXED  BIN,  /*  BOX  AWAITING  MAIL  */ VP  00100
B1  MSG  PTR,  /*  MSG  IN  WAIT.BOX  */ VP  00110
80  VPID  FIXED  BIN,  /*  INDEX  IN  THE  VPST  */ VP  00120
80  LEVEL  FIXED  BIN,  /*  LEVEL  */ VP  00130
80  VTIME  FIXED  BIN  (31)

100050  1  0  DCL  QEVENT;  ****00050
00  ENTRY  (FIXED  BIN,  FIXED  BIN  (31),  CHAR(*)  VAR,  QEVO0010
     FIXED  BIN  (31),  PTR);  QEVO0020

500010  1  0  DCL  QEVENT  ENTRY  (FIXED  BIN,  FIXED  BIN  (31),  CHAR(*)  VAR,  QEV00010
     FIXED  BIN  (31),  PTR);  QEV00020

500060  1  0  DCL  NULL  BUILTIN;  SYNO0060
500070  1  0  DCL  SLEEP  ENTRY;  SYNO0070

600010  1  0  DCL  STIMER  ENTRY  RETURNS  (FIXED  BIN  (31));  STI00010

600050  1  0  CALL  QEVENT  (VP.LEVEL+3+3,STIMER,'SYNC',VP.VPID,NULL);  SYNO0050
600070  1  0  CALL  SLEEP;  SYNO0070
600130  1  0  END;  SYNO0130
PL/I OPTIMIZING COMPILER

TERM: PROC

SOURCE LISTING

NUMBER LEVEL

10 0 TERM: PROC;

200010 1 0 DCL STIMER ENTRY RETURNS (FIXED BIN (31));

300020 1 0 DCL SEND ENTRY (FIXED BIN, FIXED BIN, FIXED BIN, CHAR(*) VAR, FIXED BIN, PTR);

400030 1 0 DCL (NULL.LENGTH) BUILTIN;

500040 1 0 DCL THISSVR PTR EXTERNAL STATIC;

** INCLUDE USERS: ******************************************************

** INCLUDE STIMER: ******************************************************

** INCLUDE SVR: ******************************************************


**PL/I OPTIMIZING COMPILER**

**TERM: PROC:**

**NUMBER  LEV  NT**

```
******
**SINCLUDE PF SVC;******************************
**SINCLUDE MSG;*******************************
******
```

```
1100010 1 0 DCL PT SVC PTR;
1100020 1 0 DCL MSG PT;
1100030 1 0 DCL 1 PF SVC BASED (PT SVC),
        2 SVC CHAR (7) VAR,
        2 PTR PTR;
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(2));
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;
    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;
```
PL/I OPTIMIZING COMPILER  

TERM: PROC;

NUMBER  LEV  NT

I200410  1  1  END;
I200430  1  0  ELSE IF (COMMAND = 'SEND') THEN DO;
   I200440  1  1  PUT SKIP LIST (' LEVEL,VPID,BOXID,TYPE,MESSAGE
   I200450  1  1  :);
   I200460  1  1  GET LIST (LEVEL,VPID,BOXID,TYPE,MESSAGE);
   I200470  1  1  MSGLEN = LENGTH (MESSAGE);
   I200490  1  1  CALL SEND (LEVEL,VPID,BOXID,TYPE,MSGLEN,MSGPT);
   I200490  1  1  GOTO WORK;
   I200500  1  1  END;
I200510  1  1  END:
I200530  1  0  ELSE IF (COMMAND = 'WAIT') THEN DO;
   I200540  1  1  PUT SKIP LIST (' BOX '):
   I200550  1  1  GET LIST (BOXID);
   I200560  1  1  PUT SKIP LIST (' WAITING '):
   I200570  1  1  CALL WAIT (BOXID);
   I200580  1  1  GOTO LOOP;
I200590  1  1  END:
I200610  1  0  ELSE IF (COMMAND = 'SYNC') THEN DO;
   I200620  1  1  CALL SYNC;
   I200630  1  1  GOTO LOOP;
I200640  1  1  END:
I200670  1  0  ELSE IF (COMMAND = 'FINISH') THEN DO;
   I200680  1  1  PUT SKIP LIST (' FINISHING ');
   I200690  1  1  CALL FINISH;
   I200700  1  1  RETURN;
I200710  1  1  END:
I200730  1  0  ELSE DO: /* UNKNOWN COMMAND */
   I200740  1  1  PUT LIST (' ? '):
   I200750  1  1  GOTO WORK;
I200760  1  1  END:
I200780  1  0  END TERM:
PL/I OPTIMIZING COMPILER

VPER: PROC (VP);

SOURCE LISTING

NUMBER
LEV
HT

10 0 VPER: PROC (VP):

30 1 0 DCL (THIS, NEXT) PTR;

40 1 0 DCL THISVP PTR EXTERNAL STATIC;

50 1 0 DCL 1 VP,

80 BOSA FIXED BIN (31), /* ADDR (BOTTOM DSA) */

80 TOSA FIXED BIN (31), /* ADDR (TOP DSA) */

80 SAVE SIZE FIXED BIN (31), /* SIZE OF AREA TO BE SAVED */

80 SAFE PTR,

80 PROCMAME 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 VIME FIXED BIN (31)

300010 1 0 DCL THISSVR PTR EXTERNAL STATIC; /* -> CURRENT SVR, SET BY SANER */

300030 1 0 DCL 1 SVR BASED (THISSVR),

70 NEXT PTR,

70 STIME FIXED BIN (31), /* ACCUMULATED STIME */

70 STIMEQ FIXED BIN (31), /* REAL CPU TICK RATE */

70 RNAME FIXED BIN (10,7),

70 ID FIXED BIN

600010 1 0 DCL BOXPT PTR:

600030 1 0 DCL 1 BOX BASED (BOXPT),

2 NEXT PTR,

2 ID FIXED BIN,

2 LIST PTR;

700070 1 0 DCL NEWSAVBLK PTR;

700090 1 0 DCL 1 SAVBLK BASED (NEWSAVBLK),

2 NEXT PTR,

2 ADDR FIXED BIN (31), /* BEG ADDR OF SAVED ITEM */

2 AREA ( 250) FIXED BIN (31):
NUMBER LEV NT

900010 1 0 DCL EXECUTE ENTRY (CHAR(*) VAR);
100010 1 0 DCL DSA FIXED BIN (31,0),
         NAB FIXED BIN (31,0),
         SEG FIXED BIN (31,0),
         EOS FIXED BIN (31,0);

100070 1 0 DCL GET4 ENTRY (Fixed Bin (31),
         Fixed Bin (31),
         Fixed Bin (31),
         Fixed Bin (31))
         OPTIONS (assembler, inter);
110010 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) MULTIN;
120010 1 0 DCL STIMER ENTRY RETURNS (fixed Bin (31));

1200170 1 0 THISVP = ADDR(VP);
1200190 1 0 IF VP.STATUS = 'NASCENT' THEN GO; /* 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.BDSA = 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;
FREE THIS -> BOX;

VP.MAIL = NULL;

END; /* FREEING */

RETURN;

ELSE DO: /* STATUS = Runnable */

CALL GET4 (DSA.NAS.SEG,EOS);

IF SEG = DSA THEN DO:
  PUT SKIP LIST (' ERROR: ILLEGAL SEGNO (VPER)');
  PUT DATA (SEG);
  STOP;
END;

IF (EOS-DSA) < (VP.SAVESIZE) THEN DO:
  PUT SKIP LIST (' ERROR: NOT ENOUGH SPACE TO RESUME VP (VPER)');
  PUT DATA (VP.VPID,VP.SAVESIZE,DSA,EOS);
  STOP;
END;

IF DSA = VP.DOSA THEN DO:
  PUT SKIP LIST (' ERROR: DOSA DISCREPENCY (VPER)');
  PUT DATA (VP.VPID,VP.DOSA,DSA);
  STOP;
END;

VP.STATUS = 'RUNNING';

CALL RTSTK (VP.SAFE,VP.TDSA);

STOP;

END: /* CONTROL SHOULD NEVER REACH HERE */

END: /* RESUMING A VP */

END VP;
SOURCE LISTING

NUMBER LEV NT

10 0 VPSTART: PROC (VP, PRONAME, ARGPT); 

30 1 0 DCL 1 VP, INCLUDE VP; 

80 BOSA 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 PRONAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC */ 
80 STATUS CHAR (13) VAR, /* CHAIN OF INCOMING MAIL */ 
80 MAIL PTR, /* CHAIN OF INCOMING MAIL */ 
80 WAIT, /* WAITING MAIL */ 
80 MSG PTR, /* MSG IN WAIT.BOX */ 
80 PID FIXED BIN, /* INDEX IN THE VPST */ 
80 LEVEL FIXED BIN, /* LEVEL */ 
80 VTIME FIXED BIN (31) /* VTIME FIXED BIN (31) */ 

-----

200040 1 0 DCL PRONAME CHAR(*) VAR; 
200050 1 0 DCL ARGPT PTR; 
200070 1 0 INCLUDE BUILTIN; 
300010 1 0 INCLUDE STIMER; 
400010 1 0 DCL STIMER ENTRY RETURNS (FIXED BIN (31)); 

-----

400110 1 0 VP.STATUS = 'NASCENT'; 
400120 1 0 VP.PRONAME = PRONAME; 
400130 1 0 VP.SAFE = NULL; 
400140 1 0 VP.WAIT.BOX = 0; 
400150 1 0 VP.WAIT.MSG = ARGPT; 
400160 1 0 VP.SAVESIZE = 0; 
400170 1 0 VP.MAIL = NULL; 
400180 1 0 VP.VTIME = 0; 
400190 1 0 CALL QEVENT (VP.LEVEL+3, STIMER, 'SYNC', VP.VPID, NULL); /* RICK START */ 
400220 1 0 END VPSTART;
PL/I OPTIMIZING COMPILER

WAIT: PROC (BOXID);

SOURCE LISTING

NUMBER LEVEL

10 0 WAIT: PROC (BOXID);

100010 1 0 DCL EVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR,

QEVENT FIXED BIN (31), PTR);

100010 1 0 DCL STIMER ENTRY RETURNS (FIXED BIN (31));

100030 1 0 DCL VP BASED (THISVP), INCLUDE VP;***********************

80 BOSA 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 00400
80 PRONAME CHAR (7) VAR, /* NAME OF TOP LEVEL PROC */ VP 00500
80 STATUS CHAR (12) VAR,
80 MAIL PTR, /* CHAIN OF INCOMING MAIL */ VP 00070
80 WAIT,
80 BOX FIXED BIN, /* BOX WAITING MAIL */ VP 00080
80 MSG PTR, /* MSG IN WAIT BOX */ VP 00090
80 VPID FIXED BIN, /* INDEX IN THE VPST */ VP 00110
80 LEVEL FIXED BIN, /* LEVEL */ VP 00120
80 VTIME FIXED BIN (31)

300010 1 0 DCL VP..PTR EXTERNAL STATIC; /* -> CURRENT VP, SET BY VPEN */

500090 1 0 DCL BOXID FIXED BIN;

500090 1 0 VP.WAIT.BOX = BOXID;

500110 1 0 CALL QEVENT (VP.LEVEL+3*3,STIMER,'WAIT',VP.VPID,NULL);
APPENDIX B: LISTINGS OF ASSEMBLY LANGUAGE ROUTINES

ASSEMBLY LANGUAGE ROUTINES:

GET4
RTIMER
RTSTK
SVSTK
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJECT CODE</th>
<th>ADDR1</th>
<th>ADDR2</th>
<th>STM</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>C7C5E3F440</td>
<td></td>
<td></td>
<td></td>
<td>CSECT</td>
</tr>
<tr>
<td>000005</td>
<td>05</td>
<td></td>
<td></td>
<td></td>
<td>GET4</td>
</tr>
<tr>
<td>00000A</td>
<td>5861 0000</td>
<td></td>
<td></td>
<td></td>
<td>DCL GET ENTRY OPTIONS (ASSEMBLE, INTER)</td>
</tr>
<tr>
<td>00000E</td>
<td>5006 0000</td>
<td></td>
<td></td>
<td></td>
<td>CALL GET4 (OSA,NAB,SEGNO,EOS)</td>
</tr>
<tr>
<td>000012</td>
<td>5861 0004</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000016</td>
<td>587D 004C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00001A</td>
<td>4177 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00001E</td>
<td>5076 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000022</td>
<td>5861 000B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000026</td>
<td>587D 004C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00002A</td>
<td>8670 001B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00002E</td>
<td>5076 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000032</td>
<td>5861 000C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000036</td>
<td>597C 000C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00003A</td>
<td>4177 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00003E</td>
<td>5076 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000042</td>
<td>90EC 000C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000046</td>
<td>07FE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. GET4 GETS THE VALUE OF THE CURRENT OSA, NAB, SEGNO & EOS
2. DCL GET ENTRY OPTIONS (ASSEMBLE, INTER)
3. CALL GET4 (OSA,NAB,SEGNO,EOS)

**ASM** 0201 21.34 05/18/81

- GET00010
- GET00020
- GET00030
- GET00040
- GET00050
- GET00060
- GET00070
- GET00080
- GET00090
- GET00100
- GET00110
- GET00120
- GET00130
- GET00140
- GET00150
- GET00160
- GET00170
- GET00180
- GET00190
- GET00200
- GET00210
- GET00220
- GET00230
- GET00240
- GET00250
- GET00260
- GET00270
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJECT CODE</th>
<th>ADDR1</th>
<th>ADDR2</th>
<th>STMNT</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000</td>
<td>4009E3C904C5D9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000007</td>
<td>06</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000008</td>
<td>00EC 000C</td>
<td>000C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000C</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000E</td>
<td>4140 0032</td>
<td>00030</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000012</td>
<td>8340000C</td>
<td>0001C</td>
<td>000C</td>
<td>4,28(4)</td>
<td></td>
</tr>
<tr>
<td>000014</td>
<td>5440 0042</td>
<td>00050</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000018</td>
<td>5011 0000</td>
<td>00000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000022</td>
<td>5041 0000</td>
<td>00000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000028</td>
<td>00EC 000C</td>
<td>0000C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00002A</td>
<td>07FE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000030</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000050</td>
<td>TFFFFFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ASM 0201 21.34 05/18/81**

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

**CSECT**

- RTM: ENTRY RTIMER
- ENTRY RTIMER
- RTIMER
- BALR 11,0
- USING R1,11
- BLK 4
- DC R'SDF0000C'
- 4,28(4)
- 1,0(1)
- 4,0(1)
- 14,12,12(13)
- 14
- DS 40
- END R'SDF0000C'

**SOURCE STATEMENT**

- DIAG R4,'00C', GET BLOCK
- LOWER 32 BITS OF TOTAL CPU TIME
RTSTK restores the stack:

* RTSTK

DC C'RSTK'

ENTRY RTSTK

ESTABLISH BASE

CHAIN BACK

NEW OSA

SET FLAGS

COPY A(LSW), ETC.

IST ARG -->

1ST SAVBLK

2ND ARG --> TOP OSA

RETURN

WHERE RTSTK SHOULD RETURN

12 SAVBLK.

WHERE RTSTK SHOULD RETURN

SAVE AREA

SOURCE STATEMENT

ASM 0201 21.34 05/18/67
LOC  OBJECT CODE  ADDR1  ADDR2  5NT  SOURCE STATEMENT

1   SYSTK SAVES THE STACK:

2   CALL SYSTK (PTR, DSA):

3   WHERE

4   PTR -> SAVBLK: 0(SAVBLK) -> NEXT SAVBLK

5   SIZE OF SAVBLK

6   SAVE AREA

7   DSA -> RETURN DSA (WHERE SYSTK SHOULD RETURN)

8   ENTRY SYSTK

9   USING *11

10  LA 6,13 RB -> OLD DSA

11  LA 13,SAYE R13 -> NEW DSA

12  CHAIN BACK

13  SET FLAGS

14  LOOP AGAIN

15  RESTORE WORD

16  RESTORE FROM SIZE-4 TO 0

17  END
APPENDIX C: LISTINGS OF MACRO FILES (DATA DECLARATIONS)

MACRO FILES:

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

DECLARE MAXLEVEL FIXED;
DECLARE MAXQ FIXED;
MAXLEVEL = 1;
MAXQ = 6;

CD000010
CD000020
CD000030
CD000040
DCL 1 DEBUG EXTERNAL STATIC.
2 SLEEP'S BIT(1) INIT ('0'8).
2 SHELRS BIT(1) INIT ('0'8).
2 SCHEDULER'S BIT(1) INIT ('0'8).
2 SHELS BIT(1) INIT ('0'8).
2 AADHERS BIT(1) INIT ('0'8);
FILE: EVENT PLI A

CONVERSATIONAL MONITOR SYSTEM

70 NEXT PTR.
70 STIME 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
PL1: A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL LISTPT PTR:

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

LIS00010
LIS00020
LIS00030
LIS00040
LIS00050
LIS00060
FILE: LOS PLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

70 LEVEL FIXED BIN.
70 SRO.
71 SIZE FIXED BIN.
71 MAX FIXED BIN.
70 DBB.
71 SIZE FIXED BIN.
71 MAX FIXED BIN.
70 VPS. %DECLARE MAXVP FIXED; %MAXVP = 20;
71 TABLE (MAXVP). %INCLUDE VPS;

LOS00010
LOS00020
LOS00030
LOS00040
LOS00050
LOS00060
LOS00070
LOS00080
LOS00090
FILE: LOSX
PLI: A
CONVERSATIONAL MONITOR SYSTEM

DCL THISLOS PTR EXTERNAL STATIC: /* --> CURRENT LOS; ST BY SAHER *// LOS00010
DCL 1 LOS BASED (THISLOS), %INCLUDE LOS: 1
LO50020
LO50030
LO50040
DCL MSGLEN FIXED BIN;
DCL MSGPT PTR;
DCL MSG BASED (MSGPT),
       2 LEN FIXED BIN,
       2 STR CHAR (MSGLEN REFER (LEN));
DCL 1 PARAMS EXTERNAL STATIC,
  2 THRU_RATE FIXED BIN (31,7),  
  2 DELAY_GB_GQ FIXED BIN (31),  
  2 DELAY_GC_GB FIXED BIN (31),  
  2 TERMINALS FIXED BIN;

PAR00010
PAR00020
PAR00030
PAR00040
PAR00050
PAR00060
DCL PT_ADDR PTR:

DCL 1 PF_ADDR BASED (PT_ADDR),
  2 VP10 FIXED BIN,
  2 BOXID FIXED BIN,
  2 PTR PTR;

PFA00010
PFA00020
PFA00030
PFA00040
PFA00050
PFA00060
PFA00070
DCL PT_LEVEL PTR;
DCL 1 PF_LEVEL BASED (PT_LEVEL),
2 LEVEL FIXED BIN,
2 PTR PTR;

PFL00010
PFL00020
PFL00030
PFL00040
PFL00050
PFL00060
DCL PT_MSG PTR;
DCL PF_MSG BASED (PT_MSG),
   2 LEN FIXED BIN,
   2 TYPE CHAR (12) VAR,
   2 PTR PTR;
FILE: PFS PLI A

CONVERSATIONAL MONITOR SYSTEM

DCL PT_5 PTR;
DCL PTR_BASED (PT_5);
LEN FIXED BIN,
PTR PTR;

DCL PT_5 PTR:
DCL PTR_BASED (PT_5);
LEN FIXED BIN,
PTR PTR;

PF500010
PF500020
PF500030
PF500040
PF500050
PF500060
FILE: PFSVC  PLI   A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DCL P1_SVC PTR:
PFS00010
PFS00020
PFS00030
PFS00040
PFS00050
PFS00060

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

243
FILE: SAVBLK  PLI  A

CONVERSATIONAL MONITOR SYSTEM

DECLARE MAXSAVBYTE FIXED;
DECLARE MAXSAVWORD FIXED;

$MAXSAVBYTE = 1000;
$MAXSAVWORD = 250;

DCL NEWSAVBLK PTR;

DCL 1 SAVBLK BASED (NEVSAVBLK).
  2 NEXT PTR, /* NEXT SAVED ITEM
  2 ADDR FIXED BIN (31), /* BEG ADDR OF SAVED ITEM
  2 SIZE FIXED BIN (31), /* SIZE OF SAVED ITEM
  2 AREA (MAXSAVWORD) FIXED BIN (31);

XDEACTIVATE MAXSAVWORD;

SAVE0010
SAVE0020
SAVE0030
SAVE0040
SAVE0050
SAVE0060
SAVE0070
SAVE0080
SAVE0090
SAVE0100
SAVE0110
SAVE0120
SAVE0130
SAVE0140
SAVE0150
SAVE0160
FILE: SVRPLI A

CONVERSATIONAL MONITOR SYSTEM

PAGE 001

70 HEXIT PIR.
70 STIME FIXED BIN (31).
70 STIMEQ FIXED BIN (31).
70 RTIME FIXED BIN (31).
70 RATE FIXED BIN (15,7).
70 ID FIXED BIN

/* ACCUMULATED STIME */
/* REAL CPU TICK RATE */

SVRO0010
SVRO0020
SVRO0030
SVRO0040
SVRO0050
SVRO0060
FILE: SVRX PLI A CONVERSATIONAL MONITOR SYSTEM PAGE 001

DCL THISSVR PTR EXTERNAL STATIC: /* → CURRENT SVR, SET BY SAHER */  SVR00010
                 SVR00020
DCL I SVR BASED (THISSVR), %INCLUDE SVR:  I
                 SVR00030
                 SVR00040
FILE: VP
PLI A
CONVERSATIONAL MONITOR SYSTEM

« ADDW (BOTTOM OSA) */VP 00010
« ADDW (TOP OSA) */VP 00020
« SIZE OF AREA TO BE SAVED */VP 00030
« 1ST SAVE LR */VP 00040
« NAME OF TOP LEVEL PROC */VP 00050
« SIZE OF AREA TO BE SAVED */VP 00060
« CHAIN OF INCOMING MAIL */VP 00070
« BOX AWAITING MAIL */VP 00080
« MSG IN WAIT BOX */VP 00090
« INDEX IN THE VPST */VP 00100
« LEVEL */VP 00110
« VPID */VP 00120
« VTIME */VP 00130

80 BOSA FIXED BIN (31).
80 TOSA 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.
80 MSG PTR.
80 MSG FIXED BIN.
80 LEVEL FIXED BIN.
80 VTIME FIXED BIN (31)

148
DCL THISVP PTR EXTERNAL STATIC; /* CURRENT VP, SET BY VPER */
DCL I VP BASED (THISVP), %INCLUDE VP; : 

VPX000010
VPX000020
VPX000030
VPX000040
APPENDIX: LISTINGS OF MACRO FILES (ENTRY DECLARATIONS)

MACRO FILES:

CONS
EXECUTE
GET4
QEVENT
RTIMER
RTSTK
STIMER
SVCER
SVSTK
USERS
VPSTART
DCL CONS ENTRY (PTR, PTR) RETURNS (PTR):

CON00010
FILE: EXECUTE PLI A
DCL EXECUTE ENTRY (CHAR(*) VAR): EXEC00010
DCL DSA FIXED BIN (31.0).
NAB FIXED BIN (31.0).
SEL FIXED BIN (31.0).
EOS FIXED BIN (31.0).

DCL GET4 ENTRY (FIXED BIN (31), FIXED BIN (31),
FIXED BIN (31), FIXED BIN (31))
OPTIONS (ASSEMBLER, INTER);
DCL QEVENT ENTRY (FIXED BIN, FIXED BIN (31), CHAR(*) VAR, FIXED BIN (31), PTR);

QEVO0010

QEVO0020
FILE: RTIMER PLI A

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

RT100010
DCL RTSTK ENTRY (PTR, FIXED BIN (31))
    OPTIONS (ASSEMBLER, INTER);

/* ARG1: PTR -> SAVBLK
   ARG2: ADDR OF RETURN DSA (TOP DSA)
*/
FILE: STIMER PLI A CONVERSATIONAL MONITOR SYSTEM PAGE 001

DCL STIMER ENTRY RETURNS (FIXED BIN (31)); ST100010

-157-
FILE: SVCER PLI A CONVERSATIONAL MONITOR SYSTEM PAGE 001

DCL SVCER ENTRY (FIXED BIN, PTR):

SVC00010
DCL SVSTK ENTRY (PTR, FIXED BIN (31))
OPTIONS (ASSEMBLER, INTER);
/* ARG1: PTR -> SAVBLK
ARG2: ADDR OF RETURN DSA (BOTTOM DSA)
*/

<table>
<thead>
<tr>
<th>SVS00010</th>
<th>SVS00020</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVS00030</td>
<td>SVS00040</td>
</tr>
<tr>
<td>SVS00050</td>
<td>SVS00060</td>
</tr>
<tr>
<td>SVS00070</td>
<td></td>
</tr>
</tbody>
</table>
FILE: USERS  PLI  A  CONVERSATIONAL MONITOR SYSTEM  PAGE 001

*include stimer;
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
DCL VPSTART ENTRY /* VP */, CHAR(*) VAR, PTR):  VPS00010