CONCURRENT PROGRAMMING OF THE USER ENVELOPE IN
A DISTRIBUTED DATA BASE MANAGEMENT SYSTEM

by

MICHAEL WAYNE FARRELL

B.S., Kansas State University, Manhattan, Kansas, 1975

A MASTER’S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1979

Approved by:

Fred Maryanski
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>II. Distributed Data Processing Trends</td>
<td>5</td>
</tr>
<tr>
<td>III. Distributed Data Processing Prototype</td>
<td>11</td>
</tr>
<tr>
<td>IV. User Envelope Component Types</td>
<td>24</td>
</tr>
<tr>
<td>V. Prototype Trace</td>
<td>42</td>
</tr>
<tr>
<td>VI. Project Results</td>
<td>66</td>
</tr>
<tr>
<td>VII. Further Study Areas</td>
<td>71</td>
</tr>
<tr>
<td>VIII. Conclusions</td>
<td>74</td>
</tr>
<tr>
<td>References</td>
<td>77</td>
</tr>
<tr>
<td>Appendix</td>
<td></td>
</tr>
<tr>
<td>A. Application Program in Sequential Pascal</td>
<td>A-1</td>
</tr>
<tr>
<td>B. User Envelope Enhancements to SOLO</td>
<td>B-1</td>
</tr>
</tbody>
</table>
# FIGURES

1. A Fully Distributed Data Processing Network ..... 12
2. Functional Components of a Host Machine ..... 13
3. The Entry Point to a Distributed Network from an Application Program ..... 15
4. Functional Components of a Backend Machine ..... 17
5. A Backend Machine with a Single Threaded Data Base Management System ..... 18
6. A Backend Machine with a Multithreaded Data Base Management System ..... 19
7. Functional Components of a Bi-functional Machine ..... 21
8. A Distributed Data Processing Network Prototype ..... 22
9. Frontend_CONNECT Process ..... 25
10. Backend_CONNECT Process ..... 26
11. Access to the User Envelope by the JOB Process ..... 27
12. FORWARD_REQUEST Process ..... 29
15. Backend_Network Resource Controller (BE_NRC) Process ..... 33
16. Message System_REQUEST Process ..... 34
17. CALL_DATABASE Process ..... 35
18. DIRECTORY Monitor ..... 37
19. HINTMON Monitor ..... 38
20. FOR_REQ_MON Monitor ..... 39
21. ID_TABLE_MON Monitor ..... 40
22. BINTMON Monitor ..... 41
23. NRCs at System Initialization ..... 43
24. All Processes Initialized ..... 44
25. Application Program Request ..... 46
26. JOB Process SENDING Variables ..... 47
27. FORWARD_REQUEST Process ..... 48
28. FE_NRC Process ..... 49
29. BE_NRC Process ..... 50
30. FE_NRC Returning to HINTMON ..... 52
31. JOB Process Returning ..... 53
32. User Envelope After SINON ..... 54
33. A Data Base Request ..... 55
34. JOB SENDs to HINTMON ..... 56
35. FORWARD_REQUEST Process SENDs ..... 58
36. MS_REQUEST RECEIVES ..... 59
37. CALL_DATABASE Process ..... 60
38. MS_RESPONSE RECEIVES ..... 61
39. JOB Process Returns to AP ..... 62
40. FE_NRC at SINOF ..... 63
41. BE_NRC at SINOF ..... 65
TABLE

1. Project Results ........................................ 70
ACKNOWLEDGMENTS

I would like to thank Dr. Fred J. Maryanski for his support and guidance throughout this project. Thanks to Dr. Virgil E. Wallentine and Dr. David A. Gustafson for serving as members of my committee. A special thanks to Rhonda Terry and Madi McArthur for the many hours they spent putting the report into final form.
I. INTRODUCTION

The material presented in this report is an extension of the work currently underway in the Computer Science Department at Kansas State University. The majority of that work has evolved around the concept of a distributed data processing system. According to Enslow [4], a distributed system contains at least four components that might be distributed—the hardware or processing logic, the data, the processing itself, and the control.

The Department began its work in the area of distributed data processing by developing a functionally distributed network of heterogeneous computing systems [27]. The software necessary for physically distributed hardware components to communicate messages (i.e., transfer control commands and data buffers), is called the MINI MICRO Computer System (MIMICS). The MIMICS software is a mechanism to support a distributed data processing system. The software provides communication among physically distributed hardware components through modems and across phone lines. Each distributed hardware component is still a general purpose resource component that can support both application program processing and MIMICS software. Inherent in MIMICS is distributed control. Each hardware component has the capability to designate and route a message from the source component to a number of destination components. The fourth component of a distributed system,
distributed data, has been examined by Maryanski [14,15,16,17,18,19]. His work involved the functional components of a Backend Distributed Data Base Management System.

An actual prototype of a distributed data processing (DDP) system was implemented by Housh [9], using the previously presented components. His implementation contained an Interdata 8/32 minicomputer acting as a host machine and an Interdata 7/32 minicomputer acting as a backend machine. The host machine executed application programs. The backend machine controlled data access by supporting a data base management system (DBMS). A user envelope of software routines had been suggested by Maryanski [18] to interface between the application programs and MIMICS on the host machine and to interface between the DBMS and MIMICS on the backend machine. In the Housh prototype, the application program was coded in Cobol, the user envelope was coded in Cobol and Interdata assembler, and MIMICS was coded in Concurrent Pascal. The results of the prototype showed a host request, backend response time of seventeen seconds. Housh suggests the response time may be lowered with faster hardware, high speed modems and transmission lines, and software modifications to MIMICS. The aim of the work reported on in this document is to develop a prototype that contains software enhancements to Housh's prototype and to reduce the host request and backend response time in the new prototype.
These software enhancements include restructuring of the user envelope to support concurrent processing, implementation of a network resource controller (NRC), and implementation of a directory of active network names that contains connected host and backend machines and connected application programs and data base tasks. In the new prototype, the application program was coded in Sequential Pascal, the restructured user envelope was coded in Concurrent Pascal, and MIMICS was used as originally developed and coded in Concurrent Pascal.

There are several benefits gained by restructuring the envelope and coding the user interface routines in Concurrent Pascal [3]. These benefits are listed below:

1. A framework that is modular and in a high-level systems programming language that can be easily understood, so that many of the issues, trends, and components of a distributed data processing system can be implemented and their results studied;

2. Further decentralization of control in a distributed data processing system by implementing a network resource controller and a directory of network names that are available to be allocated and connected to tasks on a dynamic basis; and
3. Increased system throughput which effectively decreases host request and backend response times.

In the next chapter of this report, some of the recent trends in distributed data processing are presented. The third chapter discusses the prototype used in this report. The component types of the user envelope are presented in chapter four. In chapter five a walk-through trace, of different commands being processed through the user envelope, is provided. Chapter six deals with the project results. Further study areas involving the prototype framework in the area of distributed data processing are presented in chapter seven. The conclusions of this project are discussed in chapter eight.
II. DISTRIBUTED DATA PROCESSING TRENDS

The distributed data processing concept has evolved around a number of potential application areas that have acted as a forcing function [24]. An automated factory has been described as an application area. Small computers can be distributed throughout the factory to provide automated scheduling, on-line inventory control, automated warehousing information and purchase order applications. An automated office can use small distributed computers for remote order entry, production scheduling, purchasing and credit collections, text editing, and other secretarial functions. The education field can use small computers distributed through homes and offices to provide a self-paced, self-teaching education systems. Distributed computers can be described as the basis for a home medical center. The computerized system can monitor body parameters, manage exercise and diet conditions, track medication usage, and report early warning of any unfavorable trends. Other application areas have been described that use a computerized system to control energy usage and many national defense areas suggest the use of distributed computers in the design of tactical and strategic systems.

Evolving semiconductor technologies have provided a response to these forcing functions. Microprocessors, random access memory, magnetic bubble memory, and charge-coupled devices are types of semiconductor
technologies that are declining in price and expanding in capability. New, high-density, semiconductor technologies have increased component density on a chip. Computer aided chip design and new electron beam and x-ray lithography wafer processing techniques, should give the Very Large Scale Integration (VLSI) technology the capability to build a single-chip, 32-bit, microcomputer with one million bits of memory [24]. The declining cost of computer power is eroding many of the traditional economies of scale for large, centralized computing centers. It is becoming more cost effective for data bases and processing power to be placed at the point of their greatest use.

The potential payoff from the evolving semiconductor technology is due to high volume usage of a small number of chips [2]. It is not the custom designed LSI chip but rather the small set of modular LSI chips that have mainframe-equivalent performance capabilities. An example of a modular LSI chip is Advanced Micro Devices' 2900 bit-slice microprocessor chip. This microprocessor represents a low cost, general purpose computing element. These microprocessors make the revolutionary concepts of large networks of interconnected processing elements much more feasible [5]. The major cost in this type of network, with hundreds of thousands of powerful, low cost computation elements, is the cost of communication between processing elements.
In an effort to reduce the intermodule communications cost, the low cost, modular computing elements can be programmed for software-controlled activation and deactivation of module interconnections [12]. This technique allows a system to reconfigure hardware resources into different, variably sized computers in order to dynamically adjust the computer architecture to the requirements of the executing task. A dynamically, reconfigurable architecture composed of low cost, computing elements will help eliminate some of the problems relating to the allocation of expensive hardware resources among many users. However, to broaden the power and increase the flexibility of the distributed network, the network nodes must be software controlled to redistribute the hardware resources and reconfigure the data link interconnections on a dynamic basis. A major problem of the distributed network will be to identify potential parallelism in the low cost, computational elements [5]. The principle issues will be modularity and simplicity of the interconnect structure. To fully exploit the potentials of parallelism, new systems architecture concepts and new ways of conceptualizing computational processes will be necessary.

Previous digital design theories fall short of the predicted system theories that will be needed [11]. The previous theories fall short for two reasons:
1. They do not go beyond the level of small computer devices such as counters, sequences, and adders; and

2. Minimization techniques cost more than the cost of the hardware components saved.

Some design aids have been suggested to help increase capability and productivity of LSI system designs [25]. These design aids include synthesis (create a structure to optimize objectives through the use of algorithms and system level design tools), simulation (a model for complex systems), modeling (an abstraction of a process to predict behavior), and verification (demonstrate a fabricated system meets specifications).

The advances in semiconductor technologies, the LSI modular computing elements, computer networks and distributed processing are requiring a new view of the organization of computer systems. These advances have affected all levels of the system--operating systems, data base management systems, teleprocessing systems, various programming languages, and application packages [1]. Data communications have provided the computer system with the capability to provide a front-end multiplexing of terminals with a central processor and now intermodule communication among a number of distributed, programmable computer elements in a computer network. User interfaces that require a structured, top-down refinement process to translate gross performance requirements into detailed
low-level objectives are contributing to computer systems growth and use. There is increased use of high-level structured languages such as Pascal for the programming of computer systems. As the size and complexity of computer systems increase, it is important for a computer system design methodology to consider reliability (recognize failing components and remove them), security (prevent unauthorized access to computers and the data they store), cost vs. performance (too many system wide tables and resource managers do not allow the performance increases expected in a multiprocessing system), and management of storage hierarchies (intermediate storage buffers rather than moving head disks) necessary because of technology advances in storage products such as magnetic bubbles and charge-coupled storage devices. Future trends in data base management system technology consider usability, data integrity, and performance [28]. Better access languages will increase the accessibility to many users. Data protection is the aspect of DBMS technology that provides protection against incorrect data entries, recovery from failure, and protection through concurrent access by multiple users, which is essential to a high throughput systems. DBMS performance is expecting key breakthroughs in distributed, multiprocessor DBMSs and specialized DBMS machines [28]. A DBMS machine is analogous to a front-end communications controller. It is any hardware, firmware, or software that concentrates or dedicates special purpose
computing resources to support the data management portion of a computing system. These data base machines and low cost computing elements will allow a host machine to off-load such tasks as index searching, file maintenance, and list manipulation. These trends and complex distributed computer systems are expected to be part of day-to-day operations in the next decade and bring computer power to people who need it without sacrificing management control over data quality and use.
III. DISTRIBUTED DATA PROCESSING PROTOTYPE

There are many different topologies available to computer network designers. Figure 1 presents a fully distributed data processing network that contains four nodes. A fully distributed network contains the necessary communication software to maintain communication with every other network node. The computers in the network may be classified into any of four categories depending upon the function they perform [19]. These functions are:

1. Frontend--acts as user interface, receives input, transmits output.
2. Host--executes application programs.
3. Backend--controls data access by execution of data base operations.
4. Bi-functional--combines host and backend functions.

The functional components of a host machine are presented in Figure 2. Each computer supports and executes message system software to provide the intercomputer communication necessary for a computer network. The MIMICS software was available at KSU and used in this prototype. It provided six commands to a user of the message system. There were GET_ID, CONNECT, SEND, RECEIVE, DISCONNECT and PURGE [8].

The user envelope is the software routines to interface between the application programs and the message system.
Figure 2
Functional Components of a Host Machine
The network resource controller and the directory of network names was designed into the user envelope. The SOLO operating system, coded in Concurrent Pascal, was used as the operating system to support the application programs [20,21]. In SOLO, each application program is assigned a separate JOB process to perform its requested functions. Each application program makes calls to its JOB process through a series of entry points contained in a PREFIX routine. These entry points are a set of functions that SOLO executes to support the application programs. To prevent any language interface problems, the application programs were coded in Sequential Pascal.

Figure 3 presents the entry point used by an application program to gain access to data controlled by a data base management system (DBMS) on another computer in the network. The application programmer does not need to know the network name of the computer on which it resides or the network names of other computers in the network. The programmer does not need to know the specifics of connecting the application computer to another computer, how to send the other computer a request message, or when to receive a response message from another computer. All of these network functions are transparent to the application program user. When the application program needs access to data controlled on another computer, it makes a minimum of three calls through one module. The module is called HINT as specified in the original work done by Maryanski [18]. The
Figure 3
The Entry Point to a Distributed Network
from an Application Program
HINT module of the host machine requires the parameters necessary to process the data base call on the backend. The first call to the backend would be to SINON. The next call would be to access the data base (DB) through ADD, DELETE, WRITE, MODIFY, or other commands supported by the DBMS. The last call to the backend would be to SINOP.

The user envelope formats the parameters from the application program into a buffer and make the necessary calls to the message system. The user envelope also returns status and variable information from the DBMS to the application program. If the DBMS on the backend was to change, the only change visible to the application program would be the parameters necessary to make the HINT call. The necessary parameters would be those expected by the DBMS call.

The functional components of a backend machine are presented in Figure 4. The backend computer also contains the MIMICS message system software. The user envelope on a backend waits for DB requests from a host application program. It is the backend user envelope that makes the DB calls and formats the return buffer with status and variable information returned from the DB call. In Figure 5, a single threaded DBMS is presented. Each data base task (DBT) requires its own copy of the DB. In a multithreaded DBMS, as presented in Figure 6, a single copy of the DB will support all the DBTs on the backend.
Figure 4
Functional Components of a Backend Machine
Figure 5
A Backend Machine with a Single Threaded Data Base Management System
Figure 6
A Backend Machine with a Multithreaded Data Base Management System
Figure 7 presents the functional components of a bi-functional machine. It contains the software necessary to support application programs and make the DBMS request to a backend. It also contains the backend software to process DB requests from a host, and to return responses to an application program.

The network prototype implemented in this project is presented in Figure 8. This prototype consists of a two computer network—one computer acting as a host processing APs; and the other computer is a backend, processing DBMS commands and DB calls. There are two implementation restrictions in this prototype. The first restriction is associated with the multiple application programs. The entry point into the user envelope will support and queue up multiple JOB process requests, i.e., application programs. The current KSU environment provides the JOB process access to the console device that started the process. The ideal situation to support multiple APs would be to give each JOB process access to a separate console device but access to a single SOLO task. The effort to assign multiple JOB processes to separate console devices is dependent on further KSU work and determined to be beyond the scope of this project.

The second restriction is associated with the backend DBMS. There are language interface restrictions between the TOTAL DBMS and Concurrent Pascal. An effort is underway to gain access to a DBMS written in Pascal. The DBT process,
FIGURE 7
FUNCTIONAL COMPONENTS
OF A BI-FUNCTIONAL MACHINE
Figure 8
A Distributed Data Processing Network Prototype
rather than make the DBMS call, prints out the parameters passed to it from a host machine, the application program request, and returns to the application program, a status and dummy variable area.

In this chapter the functional components of a distributed data processing network and the functional components of the prototype implemented in this project are presented. The next chapter in this report presents a detailed view of the user envelope component types contained in the implemented prototype.
IV. USER ENVELOPE COMPONENT TYPES

This chapter provides the functional specifications (i.e., access rights and data that is shared between processes) for each of the user envelope component types. Each process is presented in a separate figure along with the monitors to which the process has access rights. Each monitor is also presented in a separate figure. Processes with access rights to the monitor are also included in the figure.

PROCESSES

Figure 9 presents the Frontend_CONNnect (FE_CONN) process of the user envelope. The process RECEIVES from the DIRECTORY monitor, the IDs of the host and backend machines. The CONNECT command is used to transfer the IDs to the Message System (MS) monitor.

The Backend_CONNnect (BE_CONN) process is presented in Figure 10. It has the same rights and variable specifications as the FE_CONN process.

Figure 11 presents the JOB process of the SOLO operating system. There is a one-to-one correspondence of JOB processes to Application Programs (APs). An AP, requesting data controlled by a data base management system (DBMS) on a backend machine, has access to the JOB process through the HINT entry point. The AP passes to the process
Figure 9
Frontend_CONNECT Process
Figure 10

Backend_CONNECT Process
Figure 11
Access to the User Envelope by the JOB Process

Diagram showing the flow of data between HINTMON, JOB_N, and AP_N.
the parameters expected by the DBMS. The AP is delayed at this entry point until the STATus and VAREA variables are returned by the JOB process.

The JOB process has access to the HINTMON monitor. The SEND entry point passes the number of the application program, SPNUM, making the request and the necessary DBMS parameters in a variable called SEND_BUFF. The JOB process then gains access to HINTMON through APRETURN. It passes the monitor the AP number and the process is delayed until HINTMON returns a response to the AP in HR_BUFF.

The FORWARD_REQUEST process is presented in Figure 12. The process has access to the HINTMON monitor through the RECEIVE entry point where it receives FOR_REQ_BUFF. The process can either SEND the FOR_REQ_BUFF buffer to the FOR_REQ_MON monitor or SEND the buffer to the MS monitor.

The Frontend_Network Resource Controller (FE_NRC) process is presented in Figure 13 and discussed below. The RECEIVE access right of the FOR_REQ_MON monitor, permits the passage of the FEBUFF buffer to the FE_NRC process. The FE_NRC process has six entry points into the MS monitor. GET_ID returns NAME to the process. SEND passes BUFFER to the MS monitor. RECEIVE returns BUFFER to the process. CONNECT and DISCONNECT pass ID_LOC and ID_REM to the MS monitor. The PURGE entry point passes ID_LOC to the monitor. The FE_NRC process has access to the DIRECTORY monitor to RECEIVE the FE_NRC and BE_NRC four character IDs. The PUT and REMOVE entry points to the ID_TABLE_MON monitor
Figure 12
FORWARD_REQUEST Process
Figure 13
Frontend Network Resource Controller (FE_NRC) Process

Message System
pass ID_LOC and ID_REM from the FE_NRC process. } RSPRETURN permits the passage of FEBUFF to HINTMON monitor.

Figure 14 presents the access rights of the MS_RESPONSE process. The process has access to the MS monitor through the RECEIVE entry point. BUFFER is passed to the process from the monitor. The IDREAD entry point into the ID_TABLE_MON returns the IDNUM and RESPIDS variables to the MS_RESPONSE process. The process then passes RSPBUFF to the HINTMON monitor.

The Backend_Network Resource Controller (BE_NRC) in Figure 15 has the same access rights and variable specifications as the FE_NRC.

The MS_REQUEST process in Figure 16 has the same access rights and variable specifications as the MS_RESPONSE process with the following exception. The MS_REQUEST process SENDS to the BINTMON monitor, DBBUFF rather than passing RSPBUFF to HINTMON.

The CALL_DATABASE process is presented in Figure 17. The process has access to the BINTMON monitor to RECEIVE the DAT_BUFF buffer. The Data Base command would be processed in the CALL_DATABASE process, i.e., the data base task. However, in this implementation, the DBMS parameters are output to a printer process. The STATUS and VAREA character string are formatted into a return buffer. The process has access to the MS monitor to SEND the return BUFFER.
FIGURE 14
MESSAGE SYSTEM RESPONSE PROCESS
Figure 15

BACKEND_NETWORK RESOURCE
CONTROLLER (BE_NRC) Process
FIGURE 16
MESSAGE SYSTEM REQUEST Process
FIGURE 17
CALL_DATABASE Process
MONITORS

Figure 18 presents the access rights of the processes of the user envelope to the DIRECTORY monitor. The FE_NRC, FE_CONN, BE_CONN, and BE_NRC processes, each RECEIVE (PHY_)IDs from the monitor that designate the host and backend machines of the prototype.

The HINTMON monitor is presented in Figure 19. The monitor contains two WORK_BUFF buffers that are shared between the five processes that have access to HINTMON. The JOB and FORWARD_REQUEST processes share one of the buffers to forward a request. The FE_NRC, MS_RESPONSE, and JOB process share the other buffer to return a response.

The FOR_REQ_MON monitor contains a WORK_BUFF buffer that is shared between the FORWARD_REQUEST and FE_NRC processes. It is presented in Figure 20.

The ID_TABLE_MON monitor is presented in Figure 21. The monitor contains an array of PHY_ID pairs that are shared between the FE_NRC and MS_RESPONSE processes.

Figure 22 presents the BINTMON monitor. The monitor contains an array of BUFF_ARRAY buffers. The buffers are shared between the MS_REQUEST and CALL_DATABASE processes.

This chapter has presented the functional specifications of the processes and monitors in the user envelope that were added to the SOLO operating system. The next chapter provides a walk-through trace of user commands as they are processed in the user envelope.
Figure 19

HINTMON Monitor
Figure 21
ID_TABLE_MON Monitor
V. PROTOTYPE TRACE

This section provides a walk-through trace of the application program (AP) commands as they are routed through the user envelope. The following commands are discussed: SINON, HFUnCtIoN, and SINOF. HFUnCtIoN can be any command such as READ, ADD, MODIFY, INSERT, DELETE, or any other DBMS supported command. The walk-through provides a step-by-step view of each process encountered by the command. As each process is encountered, a new figure presents the access rights and data transfers of that process. This approach provides a functional view of the command at each process in the user envelope and the route the command has traversed to reach that process. At the end of a complete command trace, the route taken through the user envelope and the function of each process will have been fully presented.

Figure 23 presents the user envelope at initialization time. The FE_CONN and BE_CONN processes each call their respective DIRECTORY monitor for FENRC and BENRC IDs. The processes then create a virtual communication path by CONNECTing the Frontend_Network Resource Controller (FE_NRC) and the Backend_Network Resource Controller (BE_NRC) to the Message System (MS). The FE_NRC and BE_NRC processes each RECEIVE IDs of backend and host machines to which they can communicate. Figure 24 shows that all processes in the user envelope then issue RECEIVES or IDREADs and are delayed at their respective processes awaiting data.
Figure 23
NRCs at System Initialization
Figure 24
All Processes Initialized
Figure 25 shows an AP calling the HINT entry point of the JOB process and passing the process the SINON command and other variables necessary for the BE data base management system (DBMS) to process the request. Figure 26 shows the JOB process sending these variables in SEND_BUFF to HINTMON. The process then waits at APRETURN for the response to route back to HINTMON.

The FORWARD_REQUEST process in Figure 27 RECEIVES the buffer from HINTMON. It determines the function is a SINON command and SENDs the buffer to FOR_REQ_MON.

Figure 28 shows the FE_NRC process RECEIVING the buffer. FE_NRC then calls the MS monitor to GET_(an)ID. The MS returns a NAME that is linked to the requesting AP (i.e., APID). The process then SENDs to the BE to which it is connected, the buffer containing the command and name. The process then issues a RECEIVE waiting for the BE to respond.

The BE_NRC process is presented in Figure 29. It RECEIVED from the FE_NRC process, a BUFFER. It decodes the SINON command. The process then calls MS to GET_(an)ID. The MS returns a NAME that is linked to the BE data base task (DBT) (i.e., DBID). The name is formatted into the buffer and the process SENDs the buffer back to the FE_NRC. The BE_NRC process then issues a CONNECT command to link the AP name to the DBT name and the pair is PUT into the backend ID_TABLE_MON monitor.
Figure 25
APPLICATION PROGRAM REQUEST
Figure 27
FORWARD_REQUEST Process
HOST

FORWARD
REQUEST

PS,HRC

GET,RC

BUFFER

RECEIVE

BACKEND

Figure 28
FE_HRC Process
Figure 29
BE_NRC Process
The FE_NRC process RECEIVES the buffer from the BE_NRC process as presented in Figure 30. FE_NRC then issues a CONNECT command to link the AP name to the DBT name and the pair is PUT into the host ID_TABLE_MON monitor. The process then passes to the HINTMON monitor a buffer that contains the name pair and status of the MS commands.

Figure 31 shows the JOB process returning to the AP the STATus and VAREA variables.

Figure 32 presents that state of the user envelope after the SINON command has been processed. There is an APID and DBID pair in each ID_TABLE_MON monitor of the host and backend machines. This has established a virtual path between the AP on a host machine and a DBT on a backend machine. The MS_RESPONSE and MS_REQUEST processes are READING the (ID) names from the monitor and are issuing RECEIVES to the MS to receive messages that have the corresponding IDs.

Figure 33 presents the AP requesting that a data base command, HFUNCTION, be processed. The command and associated parameters are passed to the JOB process through the HINT entry point.

The JOB process SENDS HINTMON the command and associated parameters in AP_BUFF as shown in Figure 34. The process then is delayed at the APRETURN entry point of HINTMON waiting for a response to the request.
Figure 30
FE_NRC RETURNING TO HINTMON
Figure 31
JOB Process Returning
Figure 32
User Envelope After SINON
Figure 33
A Data Base Request
FIGURE 34
JOB SENDS TO HINTMON
The FORWARD_REQUEST process RECEIVES the FOR_REQ_BUFF buffer from HINTMON and SENDS the BUFFER on to the MS monitor as shown in Figure 35.

Figure 36 shows the MS_REQUEST process calling the MS monitor to RECEIVE a BUFFER identified by the APID and DBIDs in the ID_TABLE_MON monitor. The process then SENDS the BUFFER to the BINTMON monitor.

The CALL_DATABASE process RECEIVES the DAT_BUFF buffer as shown in Figure 37. The process then executes the AP request by accessing the DB through the DBMS. After the DBMS call has been executed successfully, the results are formatted in a response BUFFER and the process SENDS the BUFFER to the MS monitor.

Figure 38 presents the MS_RESPONSE process waiting to RECEIVE from the MS monitor, a BUFFER identified by the APID and DBIDs in the ID_TABLE_MON monitor. The process then RETURNS the BUFFER to the HINTMON through the RSPRETURN entry point.

The JOB process receives the response buffer, HR_BUFF as shown in Figure 39. The STATus and VAREA variables are removed from the buffer and returned to the AP delayed and waiting to receive a response to its request.

The SINOF command trace is presented in the next two figures. The SINOF trace is the same as the SINON trace except for the FE_NRC and BE_NRC processes. Figure 40 shows the FE_NRC process. The process DISCONNECTs the APID and DBID. The process then PURGES the APID from the MS monitor
Figure 35
FORWARD_REQUEST Process SENDs
Figure 37
CALL_DATABASE Process
Figure 38
MS_RESPONSE RECEIVES
Figure 39
JOB Process Returns to AP
Figure 40
FE_NRC at SINOF
and removes the ID pair from the ID_TABLE_MON monitor. Figure 41 shows that the BE_NRC DISCONNECTs the APID and DBID, PURGES the DBID, and removes the ID pair from the ID_TABLE_MON monitor.

A walk-through trace of typical commands processed by the user envelope in a distributed database management system was presented in this chapter. Each command was routed through the user envelope, a process at a time, to aid in the understanding of the user envelope components. The next chapter presents the project results in terms of lines of code added and hours spent on the project in the different phases.
FIGURE 41
BE_NRC AT SINOF
VI. PROJECT RESULTS

This chapter of the report presents a discussion of the project results in terms of the size of the code added to the SOLO operating system and the time required to complete the project. A discussion of the errors encountered during the project implementation are also presented in this section.

The Application Program (AP) contained ten pages of Sequential Pascal code [Appendix A]. The function of the AP was to format a data base (DB) call based upon an interactive session with the user. This process continued until the user issued a CEASE command and the AP issued a SINOP command.

The user envelope functions of the prototype added twenty pages of Concurrent Pascal text to the SOLO operating system [Appendix B]. The component types of the user envelope include nine processes and eight monitors. Each component type is self-documenting due to the high level nature of the programming language Concurrent Pascal. A few comment statements are provided to make the synchronization of multiple application processes, that have access to the HINTMON and RINTMON monitors, understandable.

A top down, structured approach was used in the project. Some software engineering techniques such as adequate documentation, hierarchical access graphs, and structured walk-throughs were used. This approach resulted
in a programmer productivity rate of twenty four lines of code per day.

A total of 400 hours was spent on the project. Project definition, requirement specifications, and system design required 120 of the hours or thirty percent of the project time. The result of the system design was the high level system code. The next phase of the project required 140 hours or thirty-five percent of the project time. That phase included a walk-through of the high level system code, the translation from the system code into Concurrent Pascal code, a walk-through of the final code and the correct compiling of the final code. The final phase, the test phase required 140 hours or thirty-five percent of the project time. The testing phase contained a larger percent of the project than was expected with the use of a high level language and the practice of software engineering techniques. However, the testing phase required two separate parts: one part included a complete test of the code without the message system task and the final test, which did include a complete test of the code with valid message system calls and responses. The final test phase can be broken down into a number of sub test phases. In the first sub phase, fourteen percent of the test time was spent resolving implementation errors local to the KSU Computer science Department environment. These local implementation errors were due to virtual limit errors and code length errors. The KSU implementation accommodates multiple SOLO
users on an Interdata 8/32 with the OS-32 MT operating system [22]. Each SOLO user task is provided approximately 72 K bytes of main storage. A virtual limit error occurs when code is added to SOLO and the SOLO task then requires more than the 72 K bytes of main storage. The error is corrected by allocating more than 72 K bytes of main storage to the enhanced SOLO task. The code length error deals with the address space of a process in a Concurrent Pascal program. Each process is divided into two logical segments [22]: the common segment which contains the concurrent code and monitor data space; and the private segment which contains the process data space. The code length error occurred when the enhanced SOLO version exceeded the address space of the common segment. The error was corrected by allocating more data space to the common segment by reducing the address space of the private segment. The next sub phase required twenty-one percent of the test phase time to completely test the code without the message system task. After that sub phase was complete, twenty-one percent of the test time was spent configuring the prototype code for the actual test. The final forty-four percent of the time was spent in actually testing the prototype with the message system task providing intertask communication.

During the final test sub phase, six run time execution errors were encountered. Two of the errors related to message system restrictions not fully accounted for in the system design. The first error was related to the message
system return codes. Each message system call returns to
the issuing process a numeric code that designates the
status of the call: such as completed correctly; timed out;
or not correctly completed. The system design was modified
so that the issuing process would cycle through the message
system call until the call was completed correctly. This
would prevent processes, that were to receive the results of
the message system call, from receiving incorrect data.

The second error related to the message system
restriction that allows only one process to be waiting to
receive data from the message system. In the frontend user
envelope, both the FE_NRC and MS_RESPONSE processes can be
waiting to receive data from the message system. In the
backend user envelope, both the BE_NRC and MS_REQUEST can be
waiting to receive data from the message system. The
message system was modified so that the time a process was
waiting to receive data from the message system was reduced.
The prototype system was modified so that the receiving
processes would cycle back to the message system call until
correct data was received. The modification requires the
processes to alternate issuing receive commands until
correct data is received by the process. This message
system restriction prevents concurrent processing in the
user envelope when both of the processes each issue message
system receive commands. This restriction prevents
increasing system throughput in the prototype; one of the
project goals. Any response time data gathered will be
invalid because of the bottleneck when issuing receive commands. However, current response times range from ten to twenty seconds. The message system will have to be modified to allow multiple processes to be waiting to receive data before valid system response times can be measured. This type of message system modification is beyond the scope of this project.

The remaining four errors were system design errors. The design errors in this project were easier to find than any previous debugging experiences of the author. The errors each required a somewhat trivial one line change. Table 1 summarizes the project results.

<table>
<thead>
<tr>
<th>HOURS</th>
<th>PHASE OF PROJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>System Design</td>
</tr>
<tr>
<td>40</td>
<td>Walkthrough and Code</td>
</tr>
<tr>
<td>20</td>
<td>Local Implementation Errors</td>
</tr>
<tr>
<td>30</td>
<td>Test Without Message System</td>
</tr>
<tr>
<td>30</td>
<td>Test Configuration</td>
</tr>
<tr>
<td>60</td>
<td>Actual Test</td>
</tr>
<tr>
<td></td>
<td>Total Test</td>
</tr>
<tr>
<td>140</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td></td>
</tr>
</tbody>
</table>

400 hours at 8 hrs/day = 50 days

1200 lines of code designed, written, and tested during the project

24 lines/day = programmer productivity

Table 1. PROJECT RESULTS
VII. FURTHER STUDY AREAS

This prototype has provided a set of routines in the user envelope and defined data structures and data types to aid in further work in the intermodule communications aspect of a distributed data processing system. The programming language Concurrent Pascal and the hierarchical structuring that is promoted by the language, has enabled two previous projects to be incorporated in this and other advanced research projects. The SOLO operating system [3] is an example of a modular, hierarchically structured operating system written in Concurrent Pascal. It was designed and implemented in a manner that makes it simple, readable, and understandable. It has been used as a model to understand operating systems and a building block that has been tailored or incorporated into other research projects. The MIMICS software [27] is a model of the interprocessor communication software necessary in a network of computers. It too was written in Concurrent Pascal and provides a number of system user commands that makes it an important building block in any distributed computing system. This prototype of envelope routines also written in Concurrent Pascal has used these two projects as building blocks and gone one step further in extending the decentralized control aspect of a distributed data processing system. This project has relieved the application user of any of the networking functions or any knowledge that the data the
programmer wishes to access is controlled by a database management system residing on another computer in the distributed computer network. It is hoped by the author that results of this project will too be used as a building block in further distributed data processing research projects. Some further study and research areas are suggested below:

1. incorporate the operating system function, the message system access calls, and the network control routines of the user envelope into a truly multiple user operating system;

2. with multiple users, provide a Concurrent Pascal compatible, multiple threaded, database management system so that multiple database tasks can be executed and response times and performance measurements taken;

3. construct a distributed network with two or more backend machines so that routing algorithms to different data bases can be studied;

4. output a menu of available data bases in the distributed network to application programmers;

5. construct and maintain a journal of network accesses on both the host and backend machines so that fault detection, diagnosis, recovery, and repair techniques can be implemented and studied [23];
6. Incorporate enough intelligence in the backend NRCs to reject frontend NRC requests and have the frontend delay the request or reroute it to another backend that can service the request, this would imply more than a master/slave relationship [10];

7. Study new system design methodologies and development tools to enhance parallel thinking and design highly concurrent systems [4];

8. Apply network flow algorithms, network queuing models, dynamic assignment graphs, and trees to understand the control aspects of a distributed system to prevent a thrashing monster [26];

9. Study a firmware implementation of the user envelope and communication software that includes microprocessors and semiconductor mass memory technologies [13].

Distributed data processing systems will continue to provide many interesting and challenging research projects and soon many cost effective implementations.
VIII. CONCLUSIONS

The original goal of this project was to enhance a previous, distributed data base management system prototype which would support multiple data base requests in a backend machine. The desired results were to support multiple application programs on a host machine that required access to data bases contained on a backend machine. This would insure that a data base management system on a backend machine would be operating at its maximum capacity.

Restrictions in the SOLO, single user operating system prevented the prototype from supporting multiple application programs. Language incompatibilities provided the other constraint. Application programs were written in Sequential Pascal because there is no interface between Cobol and Concurrent Pascal. There were no actual data base calls made in the prototype because the DBMSs available in the Computer Science Department at KSU does not interface to Concurrent Pascal. However, the enhanced prototype did demonstrate decentralized control through network resource controllers and the prototype provided a framework to study other aspects of a distributed data processing system.

The prototype was written in Concurrent Pascal because it is a high level systems programming language. This provided compatibility with MIMICS which was written in Concurrent Pascal. On the host side, multiple application programs (processes) can be supported through the HINTMON
monitor. On the backend side, multiple data base requests can be processed through the BINTMON monitor.

The approach used in enhancing the prototype, maintained the philosophy that an application programmer should remain network naive. The programmer does not need to know the location of the data requested in a data base command. The NRC and user envelope provide this network function. The NRC allocates network names on both the host and backend machines and connects and disconnects the tasks according to the request. The remaining user envelope routines route the data base request commands and responses according to a table of network names, contained in the host and backend machines, which show connected tasks.

The project has shown the enhanced prototype does support a network naive application programmer who requires access to data maintained in a distributed data processing system. The data structures, processes, monitors, and data types defined in the prototype have demonstrated a modular, structured approach to the design of a distributed data processing system. The implementation of the NRC and ID tables have further decentralized the system from a centrally controlled hierarchical system in which computing elements are statically defined as a host or backend machine, to a distributed system in which a control module, the NRC, of the computing element has the intelligence to dynamically designate the machine as a host, backend, or bi-functional machine [7]. By decentralizing the control,
the prototype has shown an approach to reach a fully distributed data processing system.
REFERENCES


21. Neal, David, and Barbara North, "Solo Tutorials," TR CS 77-20, Department of Computer Science, Kansas State University, Manhattan, Kansas 66506, October 1977.

22. Neal, David, and Virgil Wallentine, "Experiences with the Portability of Concurrent Pascal," TR CS 77-09, Department of Computer Science, Kansas State University, Manhattan, Kansas 66506, October 1977.


(NUMBER)

"PER BRINCH HANSEN
* AS MODIFIED FOR THE
* INTERDATA 8/32
* UNDER OS/32-MT AT
* DEPT. OF COMPUTER
* SCIENCE, KANSAS
* STATE UNIVERSITY

UTILITY PROGRAMS FOR
THE SOLO SYSTEM

18 MAY 1975
1 DEC 1976"

"#######
# PREFIX #
#######"

CONST NL = "(:10:)"; FF = "(:12:)"; CR = "(:13:)";
EM = "(:25:)";

CONST PAGELENGTH = 512;
TYPE PAGE = ARRAY (.1..PAGELENGTH.) OF CHAR;
CONST LINELENGTH = 132;
TYPE LINE = ARRAY (.1..LINELENGTH.) OF CHAR;
CONST IDLENGTH = 12;
TYPE IDENTIFIER = ARRAY (.1..IDLENGTH.) OF CHAR;

TYPE FILE = 1..2;

TYPE FILEKIND = (EMPTY, SCRATCH, ASCII, SEQCODE, CONCODE);

TYPE FILEATTR = RECORD
    KIND: FILEKIND;
    ADDR: INTEGER;
    PROTECTED: BOOLEAN;
    NOTUSED: ARRAY (.1..5.) OF INTEGER
END;

TYPE IODEVICE =
    (TYPEDEVICE, DISKDEVICE, TAPEDEVICE, PRINTDEVICE,
     CARDDEVICE);

TYPE IOOPERATION = (INPUT, OUTPUT, MOVE, CONTROL);

TYPE IOARG = (WRITEEOF, REWIND, UPSPACE, BACKSPACE);

TYPE IORESULT =
    (COMPLETE, INTERVENTION, TRANSMISSION, FAILURE,
     ENDFILE, ENDMEDIUM, STARTMEDIUM);
55 TYPE IOPARAM = RECORD
56   OPERATION: IOOPERATION;
57   STATUS: IORESULT;
58   ARG: IOARG
59 END;
60
61 TYPE TASKKIND = (INPUTTASK, JOBTASK, OUTPUTTASK);
62
63 TYPE ARGTAG =
64   (NILTYPE, BOOLTYPE, INTTYPE, IDTYPE, PTRTYPE);
65
66 TYPE POINTER = &BOOLEAN;
67
68 TYPE ARGTYPE = RECORD
69   CASE TAG: ARGTAG OF
70     NILTYPE, BOOLTYPE: (BOOL: BOOLEAN);
71     INTTYPE: (INT: INTEGER);
72     IDTYPE: (ID: IDENTIFIER);
73     PTRTYPE: (PTR: POINTER)
74 END;
75
76 CONST MAXARG = 10;
77 TYPE ARGLIST = ARRAY (.1..MAXARG.) OF ARGTYPE;
78
79 TYPE ARGSEQ = (INP, OUT);
80
81 TYPE PROGRESS =
82   (TERMINATED, OVERFLOW, POINTERERROR, RANGETERROR,
83     VARIANTERROR, HEAPLIMIT, STACKLIMIT, CODELIMIT,
84     TIMELIMIT, CALLERROR);
85
86 TYPE A1 = ARRAY (.1..6.) OF CHAR;
87
88 TYPE A2 = ARRAY (.1..4.) OF CHAR;
89
90 TYPE A3 = ARRAY (.1..6.) OF CHAR;
91
92 TYPE A4 = ARRAY (.1..26.) OF CHAR;
93
94 TYPE A5 = ARRAY (.1..20.) OF CHAR;
95
96 TYPE A6 = ARRAY (.1..10.) OF CHAR;
97
98 TYPE A7 = ARRAY (.1..6.) OF CHAR;
99
100 TYPE A8 = ARRAY (.1..6.) OF CHAR;
101
102 TYPE A9 = ARRAY (.1..10.) OF CHAR;
103
104 TYPE ACCEPTMSG = ARRAY (.1..26.) OF CHAR;
105
106 TYPE DISPLAYMSG = ARRAY (.1..38.) OF CHAR;
107
109  TYPE ENV_FUNCTION    = ARRAY (..6..) OF CHAR;
110  TYPE ENV_STAT         = ARRAY (..4..) OF CHAR;
112  TYPE ENV_FILE_NAME    = ARRAY (..4..) OF CHAR;
114  TYPE ENVREFER         = ARRAY (..4..) OF CHAR;
116  TYPE ENV_LINKPATH     = ARRAY (..8..) OF CHAR;
118  TYPE ENV_CTRL_FIELD   = ARRAY (..10..) OF CHAR;
120  TYPE ENV_VELEMENTS    = ARRAY (..60..) OF CHAR;
122  TYPE ENV_VAREA        = ARRAY (..82..) OF CHAR;
124  TYPE ENV_ENDP         = ARRAY (..4..) OF CHAR;
126  TYPE PEOP_IN =
128       RECORD
129           PEOP_NUM : A3;
130           FILLER1 : A1;
131           PEOP_NAM : A4;
132           FILLER2 : A1;
133           PEOP_ADR : A5;
134           FILLER3 : A1;
135           PEOP_TEL : A6;
136           FILLER4 : A1;
137           PEOP_HIR : A3;
138           FILLER5 : A1;
139           PEOP_BRT : A3;
140           FILLER6 : A1;
141           PEOP_SOC : A9;
142           FILLER7 : ARRAY (..12..) OF CHAR
143       END;
144
145  TYPE PRINT_LINE =
146       RECORD
147           FILLER1 : ENV_FUNCTION;
148           DATA1 : ENV_FUNCTION;
149           FILLER2 : ENV_FUNCTION;
150           DATA2 : ENV_STAT;
151           FILLER3 : ARRAY (..110..) OF CHAR
152       END;
153
154  TYPE PRINT_COMMAND =
155       RECORD
156           FILLER1 : ENV_FUNCTION;
157           COMM : ENV_FUNCTION;
158           FILLER2 : ENV_FUNCTION;
159           NUMB : A2;
160           FILLER3 : ARRAY (..110..) OF CHAR
161       END;
PROCEDURE READ(VAR C: CHAR);
PROCEDURE WRITE(C: CHAR);
PROCEDURE OPEN(F: FILE; ID: IDENTIFIER;
VAR FOUND: BOOLEAN);
PROCEDURE CLOSE(F: FILE);
PROCEDURE GET(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);
PROCEDURE PUT(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);
FUNCTION LENGTH(F: FILE): INTEGER;
PROCEDURE MARK(VAR TOP: INTEGER);
PROCEDURE RELEASE(TOP: INTEGER);
PROCEDURE IDENTIFY(HEADER: LINE);
PROCEDURE ACCEPT(VAR C: CHAR);
PROCEDURE DISPLAY(C: CHAR);
PROCEDURE READPAGE(VAR BLOCK: UNIV PAGE;
VAR EOF: BOOLEAN);
PROCEDURE WRITEPAGE(BLOCK: UNIV PAGE; EOF: BOOLEAN);
PROCEDURE READLINE(VAR TEXT: UNIV LINE);
PROCEDURE WRITELINE(TEXT: UNIV LINE);
PROCEDURE READARG(S: ARGSEQ; VAR ARG: ARGTYPE);
PROCEDURE WRITEARG(S: ARGSEQ; ARG: ARGTYPE);
PROCEDURE LOOKUP(ID: IDENTIFIER; VAR ATTR: FILEATTR;
VAR FOUND: BOOLEAN);
PROCEDURE IOTRANSFER
(DEVICE: IODEVICE; VAR PARAM: IOPARAM;
VAR BLOCK: UNIV PAGE);
PROCEDURE IOMOVE(DEVICE: IODEVICE; VAR PARAM: IOPARAM);
FUNCTION TASK: TASKKIND;
PROCEDURE RUN(ID: IDENTIFIER; VAR PARAM: ARGLIST;
VAR LINE: INTEGER;
VAR RESULT: PROGRESS)
PROCEDURE HINT(HFUNCTION: ENV_FUNCTION;
VAR STAT: ENV_STAT;
FILE_NAME: ENV_FILE_NAME;
REFER: ENVREFER;
LINKPATH: ENV_LINKPATH;
CTRL_FIELD: ENV_CTRL_FIELD;
VELEMENTS: ENV_VELEMENTS;
VAR VAREA: ENV_VAREA;
ENDP: ENV_ENDP);
PROGRAM P(VAR PARAM: ARGLIST);
VAR SINON, DATA1: ENV_FUNCTION;
STAT, STATERR, DATA2: ENV_STAT;
217    DUMMY1:ENV_FILE_NAME; DUMMY2:ENV_LINKPATH;
218    DUMMY3:ENV_CTRL_FIELD; DUMMY4:ENV_VELEMENTS;
219    SCHEMA:ENV_VAREA; ENDP:ENV_ENDP;
220    PRTLINE:PRINT_LINE; I:INTEGER;
221    INARG,OUTARG: ARGTYPE;
222    C: CHAR; OUTLINE: LINE;
223
224    PROCEDURE RECDCNVT(LINERECD: UNIV LINE;
225                       VAR OUTLINE: LINE);
226    BEGIN
227    OUTLINE:=LINERECD;
228    END;
229
230    PROCEDURE CRT (CRTLINE : UNIV LINE);
231    VAR I : INTEGER; C : CHAR; PROCLINE:LINE;
232    BEGIN
233       FOR I := 1 TO 69 DO
234          BEGIN
235             C := CRTLINE(.I.);
236             DISPLAY (C);
237          END;
238       DISPLAY(NL);
239    END;
240
241    PROCEDURE CLOSE_EM;
242    VAR SINOF : ENV_FUNCTION; STAT : ENV_STAT;
243        DUMMY1 : ENV_FILE_NAME;
244        DUMMY2 : ENV_LINKPATH; DUMMY3 : ENV_CTRL_FIELD;
245        DUMMY4 : ENV_VELEMENTS; SCHEMA : ENV_VAREA;
246        ENDP : ENV_ENDP;
247    BEGIN
248       SINOF := 'SINOF';
249       DUMMY1 := ' ';
250       DUMMY2 := ' ';
251       DUMMY3 := ' ';
252       DUMMY4 := ' ';
253       ENDP := 'END.';
254       HINT(SINOF, STAT, DUMMY1, DUMMY1, DUMMY2,
255          DUMMY3, DUMMY4, SCHEMA, ENDP);
256       "SIGN OFF THE DATA BASE."
257    END;
258
259    PROCEDURE MTDISPLAY (DMSG : DISPLAYMSG);
260    VAR I : INTEGER; C : CHAR;
261    BEGIN
262       FOR I := 1 TO 38 DO
263          BEGIN
264             C := DMSG (.I.);
265             DISPLAY (C);
266          END;
267    END;
PROCEDURE MTACCEPT (ANUM : INTEGER;
VAR AMSG : ACCEPTMSG);
VAR I,J,AMAX : INTEGER; C : CHAR;
BEGIN
AMAX := 26;
FOR I := 1 TO ANUM DO
BEGIN
ACCEPT (C);
AMSG (.I.) := C;
END;
FOR J := I + 1 TO AMAX DO
BEGIN
AMSG (.J.) := ' ';
END;
END;

PROCEDURE MASTER_TRANS (FCOMMAND : ENV_FUNCTION);
VAR MTAMSG, DUM1 : ACCEPTMSG; XNUMBER : A3;
NAME : A4; XADDRESS : A5; I,ANUM : INTEGER;
ENUMBER, TELEPHONE : A6; HIRE_DATE : A7;
BIRTH_DATE : A8; SOC_NUMB : A9;
D3,D4,D5,D6,D7,D8,D9 : DISPLAYMSG;
HFUNCTION,DATA1 : ENV_FUNCTION;
SSTAT,STATERR,DATA2 : ENV_STAT;
PEOP : ENV_FILE_NAME; DUMMY1 : ENV_REFER;
Dummy2 : ENV_LINKPATH;
PEOP_DATA : ENV_VELEMENTS;
PEOP_RECORD : ENV_VAREA; ENDP : ENV_ENDP;
PRTLINE : PRINT_LINE; PEPIN : PEOP_IN;
H7A, H7B, H8A, H8B, H9A, H9B : DISPLAYMSG;
DBRITN1, DBRITN2 : DISPLAYMSG;
BEGIN
D3 := 'ENTER 6 DIGIT EMPLOYEE NUMBER. '(10:10);
H3A := ' ' (10:10);
H3B := ' *****5*R' (10:10);
D4 := 'ENTER 25 CHARACTER NAME.' (10:10);
H4A := ' 1 1 2 2C' (10:10);
H4B := ' *****5*****0*****5*****0*****5R' (10:10);
D5 := 'ENTER 20 CHARACTER ADDRESS.' (10:10);
H5A := ' 1 1 2C' (10:10);
H5B := ' *****5*****0*****5*****0R' (10:10);
D6 := 'ENTER 10 DIGIT TELEPHONE NUMBER.' (10:10);
H6A := ' 1C' (10:10);
H6B := ' *****5*****R' (10:10);
D7 := 'ENTER 6 DIGIT HIRE DATE.' (10:10);
H7A := ' ' (10:10);
H7B := ' *****5*R' (10:10);
D8 := 'ENTER 6 DIGIT BIRTH DATE.' (10:10);
H8A := ' ' (10:10);
H8B := ' *****5*R' (10:10);
D9 := 'ENTER 9 DIGIT SSN.'
H9A := C
H9B := '****5****R'
DBRTN1:= 'RETURNED STATUS --'
DBRTN2:= 'RETURNED VARIABLE AREA --'
STATERR := '****';
ANUM:=1;
MTACCEPT(ANUM,DUM1);
MTDISPLAY (D3);
MTDISPLAY (H3A);
MTDISPLAY (H3B);
ANUM := 6;
MTACCEPT (ANUM, MTAMSG);
FOR I := 1 TO ANUM DO
  XNUMBER (.I.) := MTMSG (.I.);
ANUM:=1;
MTACCEPT(ANUM,DUM1);
MTDISPLAY (D4);
MTDISPLAY (H4A);
MTDISPLAY (H4B);
ANUM := 25;
MTACCEPT (ANUM, MTAMSG);
NAME(.I.):=''
FOR I := 1 TO ANUM DO
  NAME (.I+1.) := MTMSG (.I.);
ANUM:=1;
MTACCEPT(ANUM,DUM1);
MTDISPLAY (D5);
MTDISPLAY (H5A);
MTDISPLAY (H5B);
ANUM := 20;
MTACCEPT (ANUM, MTAMSG);
FOR I := 1 TO ANUM DO
  XADDRESS (.I.) := MTMSG (.I.);
ANUM:=1:
MTACCEPT(ANUM,DUM1);
MTDISPLAY (D6);
MTDISPLAY (H6A);
MTDISPLAY (H6B);
ANUM := 10;
MTACCEPT (ANUM, MTAMSG);
FOR I := 1 TO ANUM DO
  TELEPHONE (.I.) := MTMSG (.I.);
ANUM:=1:
MTACCEPT(ANUM,DUM1);
MTDISPLAY (D7);
MTDISPLAY (H7A);
MTDISPLAY (H7B);
ANUM := 6;
MTACCEPT (ANUM, MTAMSG);
FOR I := 1 TO ANUM DO
  HIRE_DATE (.I.) := MTMSG (.I.);
ANUM:=1;
MTACCEPT(ANUM,DUM1):

MTDISPLAY (D8);
MTDISPLAY (H8A);
MTDISPLAY (H8B);
ANUM := 6;
MTACCEPT (ANUM, MTAMSG);
FOR I := 1 TO ANUM DO
  BIRTH_DATE (.I.) := MTAMSG (.I.);
ANUM := 1;
MTACCEPT (ANUM, DUML);
MTDISPLAY (D9);
MTDISPLAY (H9A);
MTDISPLAY (H9B);
ANUM := 9;
MTACCEPT (ANUM, MTAMSG);
SOC_NUMB (.I.) := ' ';
FOR I := 1 TO ANUM DO
  SOC_NUMB (.I+1.) := MTAMSG (.I.);
WITH PEUPIN DO
BEGIN
  PEOP_NUM := XNUMBER;
  FILLER1 := ' ';
  PEOP_NAM := NAME;
  FILLER2 := ' ';
  PEOP_ADDR := XADDRESS;
  FILLER3 := ' ';
  PEOP_TEL := TELEPHONE;
  FILLER4 := ' ';
  PEOP_HIR := HIRE_DATE;
  FILLER5 := ' ';
  PEOP_BRT := BIRTH_DATE;
  FILLER6 := ' ';
  PEOP_SOC := SOC_NUMB;
END;
RECDCNVT (PEUPIN, OUTLINE);
OUTLINE (.132.) := NL;
FOR I := 1 TO 132 DO
BEGIN
  C := OUTLINE (.I.);
  WRITE (C);
END;
FOR I := 1 TO 4 DO
  ENUMBER (.I.) := ' ';
FOR I := 1 TO 6 DO
  ENUMBER (.I+4.) := XNUMBER (.I.);
HFUNCTION := FCOMMAND;
PEOP := 'PEOP';
DUMMY1 := ' ';
DUMMY2 := ' ';
ENDP := 'END.';
FOR I := 1 TO 60 DO
  PEOP_DATA (.I.) := OUTLINE (.I.);
HINT (HFUNCTION, STAT, PEOP, DUMMY1, DUMMY2,
  ENUMBER, PEOP_DATA, PEOP_RECORD, ENDP);
FOR I := 1 TO 38 DO
433 OUTLINE(.I.):=DBRTNL(.I.);
434 FOR I:= 1 TO 4 DO
435 OUTLINE(.I+38.):=STAT(.I.);
436 FOR I:= 1 TO 89 DO
437 OUTLINE(.I+42.):=ˊˇ;
438 FOR I:= 1 TO 132 DO
439 BEGIN
440 C:= OUTLINE(.I.);
441 WRITE(C);
442 END;
443 FOR I:= 1 TO 38 DO
444 OUTLINE(.I.):=DBRTN2(.I.);
445 FOR I:= 1 TO 82 DO
446 OUTLINE(.I+38.):=PEOP_RECORD(.I.);
447 FOR I:= 1 TO 11 DO
448 OUTLINE(.I+120.):=ˊˇ;
449 FOR I:= 1 TO 132 DO
450 BEGIN
451 C:=OUTLINE(.I.);
452 WRITE(C);
453 END;
454 IF STAT <> STATERR
455 THEN BEGIN "STATUS ERROR"
456 WITH PRTLINES DO
457 BEGIN
458 FILLER1:=ˊˇ;
459 DATA1:=HPFUNCTION;
460 FILLER2:=ˊˇ;
461 DATA2:=STAT;
462 END;
463 CRT (PRTLINES);
464 RECCNV (PRTLINES,OUTLINE);
465 OUTLINE(.132.):=NL;
466 FOR I:= 1 TO 132 DO
467 BEGIN
468 C:=OUTLINE(.I.);
469 WRITE(C);
470 END;
471 END;
472 END;
473
474 PROCEDURE GET_COMMAND;
475 VAR COMM,COMMAND : A1; NUMB,NO_TIMES : A2;
476 D1,D2 : DISPLAYMSG; I,ANUM,LIMIT : INTEGER;
477 CLOSE : BOOLEAN; PRTCOMMAND : PRINT_COMMAND;
478 MTAMSG, DUM2 : ACCEPTMSG; OUTLINE: LINE;
479 H1A, H1B, H2A, H2B: DISPLAYMSG;
480
481 BEGIN
482 D1 := 'ENTER 5 CHARACTER COMMAND. (:10:);'
483 D2 := 'ENTER 1 DIGIT NUMBER OF TRANSACTIONS. (:10:);'
484 H1A:= ' C (:10:);'
485 H1B:= ' ****5R (:10:);'
486 H2A:= ' C (:10:);'
H2B:= *R
CLOSE:=FALSE;
REPEAT
MTDISPLAY (D1);
MTDISPLAY (H1A);
MTDISPLAY (H1B);
ANUM := 5;
MTACCEPT (ANUM, MTAMSG);
COMMAND (.I.):=' ';
FOR I := 1 TO ANUM DO
  COMMAND (.I+1.):= MTAMSG (.I.):
  ANUM:=1;
MTACCEPT(ANUM,DUM2);
MTDISPLAY (D2);
MTDISPLAY (H2A);
MTDISPLAY (H2B);
MTACCEPT (ANUM, MTAMSG);
FOR I:=1 TO 3 DO
  BEGIN
    NO_TIMES (.I.):=' ';
  END;
FOR I := 1 TO ANUM DO
  NO_TIMES (.I+3.):= MTAMSG (.I.):
WITH PRTCOMMAND DO
  BEGIN
    FILLER1:= ' ';
    COMM:=COMMAND;
    FILLER2:= ' ';
    NUMB:=NO_TIMES;
  END;
CRT (PRTCOMMAND);
RECDCNV(T(PRTCOMMAND,OUTLINE);
OUTLINE (.132.):=NL;
FOR I:= 1 TO 132 DO
  BEGIN
    C:=OUTLINE (.I.);
    WRITE(C);
  END;
  IF COMMAND = ' CEASE'
  THEN CLOSE:= TRUE
  ELSE BEGIN
    I := 4;
    LIMIT:=ORD(NO_TIMES (.I.))-48;
    I:=1;
    REPEAT
      MASTER_TRANS (COMMAND);
      I := I + 1;
      UNTIL I > LIMIT;
    ANUM:=1;
    MTACCEPT(ANUM,DUM2);
  END;
UNTIL CLOSE;
CLOSE_EM;
END;
541  **************************************
542  *    SPASCAL  USER'S  PROGRAM      *
543  **************************************
544          
545  BEGIN
546    OUTARG.TAG:=IDTYPE;
547    OUTARG.ID:=`PRINTER`;
548    WRITEARG(OUT,OUTARG);
549      STATERR := `****`;
550    SINON := `SINON`;
551    DUMMY1 := ` `
552    DUMMY2 := ` `
553    DUMMY3 := ` `
554    DUMMY4 := ` `
555      ENDP := `END.`
556    HINT (SINON, STAT, DUMMY1, DUMMY1, DUMMY2, 
557             DUMMY3, DUMMY4, SCHEMA, ENDP);
558    IF STAT = STATERR 
559        THEN GET_COMMAND
560          ELSE BEGIN
561            WITH PRTLNE DO
562              BEGIN
563                  FILLER1 := ` `
564                  DATA1 := `SINON`;
565                  FILLER2 := ` `
566                  DATA2 := STAT;
567              END;
568    CRT (PRTLNE);
569    RECDNVT(PRTLNE,OUTLINE);
570    OUTLINE(.132.) := NL;
571    FOR I := 1 TO 132 DO
572        BEGIN
573            C := OUTLINE(.I.);
574            WRITE(C);
575        END;
576    CLOSE_EM;
577    END;
578    OUTLINE(.1.) := EM;
579    OUTLINE(.2.) := NL;
580    FOR I := 1 TO 2 DO
581        BEGIN
582            C := OUTLINE(.I.);
583            WRITE(C);
584        END;
585    READARG(OUT,OUTARG);
586  END.
(NUMBER, ERRORS ONLY)

" PER BRINCH HANSEN

INFORMATION SCIENCE
CALIFORNIA INSTITUTE OF TECHNOLOGY

THE SOLO SYSTEM
8 JUNE 1975"

"DAVID NEAL

DEPARTMENT OF COMPUTER SCIENCE
KANSAS STATE UNIVERSITY

MODIFICATIONS OF THE SOLO SYSTEM FOR USE
UNDER INTERDATA'S OS/32-MT AND FOR THE USE
OF TWO VIRTUAL DISKS

OCTOBER 15, 1977"

"#################
# IO TYPES #
#################"

TYPE IODEVICE =
  (TYPEDEVICE, PRIVATEDISK, TAPEDEVICE, PRINTDEVICE,
   CARDDEVICE, TAPE2DEVICE, ITAMDEVICE, VARDEVICE,
   SYSTEMDISK): "KSU"

TYPE IOARG =
  (WRITEOF, REWIND, UPSPACE, BACKSPACE);

TYPE IOOPERATION = (INPUT, OUTPUT, MOVE, CONTROL);

TYPE IORESULT = (COMPLETE, INTERVENTION, TRANSMISSION,
  FAILURE, ENDFILE, ENDMEDIUM, STARTMEDIUM);

TYPE IOPARAM = RECORD
  OPERATION: IOOPERATION;
  STATUS: IORESULT;
  ARG: IOARG
END;

CONST LINELENGTH = 132;
TYPE LINE = ARRAY [1..LINELENGTH] OF CHAR;

CONST PAGELength = 512;
TYPE PAGE = ARRAY [1..PAGELength] OF CHAR;
TYPE OKTYPE = INTEGER;

TYPE NOT_DONE_TYPE = INTEGER;

CONST NL = "(:10:)"; CR = "(:13:)"; NULL = "(:0:)";
FF = "(:12:)"; EM = "(:25:)";

"#################################################
# PROCESSQUEUE AND FIFO #
#################################################"

CONST PROCESSCOUNT = 18;
TYPE PROCESSQUEUE = ARRAY [1..PROCESSCOUNT] OF QUEUE;

TYPE FIFO =
CLASS(LIMIT: INTEGER);

VAR HEAD, TAIL, LENGTH: INTEGER;

FUNCTION ENTRY ARRIVAL: INTEGER;
BEGIN
  ARRIVAL := TAIL;
  TAIL := TAIL MOD LIMIT + 1;
  LENGTH := LENGTH + 1;
END;

FUNCTION ENTRY DEPARTURE: INTEGER;
BEGIN
  DEPARTURE := HEAD;
  HEAD := HEAD MOD LIMIT + 1;
  LENGTH := LENGTH - 1;
END;

FUNCTION ENTRY EMPTY: BOOLEAN;
BEGIN EMPTY := (LENGTH = 0) END;

FUNCTION ENTRY FULL: BOOLEAN;
BEGIN FULL := (LENGTH = LIMIT) END;

BEGIN HEAD := 1; TAIL := 1; LENGTH := 0 END;

"#################################################
# RESOURCE #
#################################################"

TYPE RESOURCE =
MONITOR
VAR FREE: BOOLEAN; Q: PROCESSQUEUE; NEXT: FIFO;

PROCEDURE ENTRY REQUEST;
BEGIN
   IF FREE THEN FREE := FALSE
   ELSE DELAY(Q[NEXT.ARRIVAL]);
END;

PROCEDURE ENTRY RELEASE;
BEGIN
   IF NEXT.EMPTY THEN FREE := TRUE
   ELSE CONTINUE(Q[NEXT.DEPARTURE]);
END;

BEGIN FREE := TRUE; INIT NEXT(PROCESSCOUNT) END;

"#################################################
# TYPERESOURCE #
#################################################"

TYPE TYPERESOURCE =
  MONITOR

VAR FREE: BOOLEAN; Q: PROCESSQUEUE; NEXT: FIFO;
  HEADER: LINE;

PROCEDURE ENTRY REQUEST(TEXT: LINE;
  VAR CHANGED: BOOLEAN);
BEGIN
   IF FREE THEN FREE := FALSE
   ELSE DELAY(Q[NEXT.ARRIVAL]);
   CHANGED := (HEADER <> TEXT);
   HEADER := TEXT;
END;

PROCEDURE ENTRY RELEASE;
BEGIN
   IF NEXT.EMPTY THEN FREE := TRUE
   ELSE CONTINUE(Q[NEXT.DEPARTURE]);
END;

BEGIN
   FREE := TRUE; HEADER[1] := NL;
   INIT NEXT(PROCESSCOUNT);
END;

"#################################################
# TYPEWRITER #
#################################################"
TYPE TYPEWRITER =
CLASS(DEVICE: IODEVICE);

CONST LINELIMIT = 81;
CANCELCHAR = '\\'; "KSU - IMPLEMENTED BY" "OS/32-MT DRIVERS"
CANCELLINE = '\#'; "KSU - IMPLEMENTED BY" "OS/32-MT DRIVERS"

PROCEDURE WRITECHAR(X: CHAR);
VAR PARAM: IOPARAM; C: CHAR;
BEGIN
PARAM.OPERATION:= OUTPUT;
C:= X;
IO(C, PARAM, DEVICE);
END;

PROCEDURE ENTRY WRITE(TEXT: LINE);
VAR PARAM: IOPARAM;
I: INTEGER; C: CHAR;
BEGIN
PARAM.OPERATION:= OUTPUT;
I:= 0;
REPEAT
I:= I + 1; C:= TEXT[I];
IO(C, PARAM, DEVICE);
UNTIL (C = NL) OR (I = LINELIMIT);
IF C <> NL THEN WRITECHAR(NL);
END;

PROCEDURE ENTRY READ(VAR TEXT: LINE);
CONST BEL = '('/'7'')';
VAR PARAM: IOPARAM;
I: INTEGER; C: CHAR;
BEGIN
WRITECHAR(BEL);
PARAM.OPERATION:= INPUT;
I:= 0;
REPEAT
IO(C, PARAM, DEVICE);
"CODE TO CANCEL INPUT LINES"
"AND CHARACTERS DELETED - KSU"
I:= I + 1; TEXT[I]:= C;
UNTIL (C = NL) OR (I = LINELIMIT);
IF C <> NL THEN
BEGIN
WRITECHAR(NL);
TEXT[LINELIMIT + 1]:= NL;
END;
END;

BEGIN END:
"##################
# TERMINAL #
##################"

TYPE TERMINAL =
CLASS (ACCESS: TYPERESOURCE);

VAR UNIT: TYPEWRITER;

PROCEDURE ENTRY READ (HEADER: LINE; VAR TEXT: LINE);
VAR CHANGED: BOOLEAN;
BEGIN
  ACCESS.REQUEST (HEADER, CHANGED);
  IF CHANGED THEN UNIT.WRITE (HEADER);
  UNIT.READ (TEXT);
  ACCESS.RELEASE;
END;

PROCEDURE ENTRY WRITE (HEADER, TEXT: LINE);
VAR CHANGED: BOOLEAN;
BEGIN
  ACCESS.REQUEST (HEADER, CHANGED);
  IF CHANGED THEN UNIT.WRITE (HEADER);
  UNIT.WRITE (TEXT);
  ACCESS.RELEASE;
END;

BEGIN INIT UNIT (TYPEDEVICE) END;

"##################
# TERMINALSTREAM #
##################"

TYPE TERMINALSTREAM =
CLASS (OPERATOR: TERMINAL);

CONST LINESIZE = 80;

VAR HEADER: LINE; ENDINPUT: BOOLEAN;
INP, OUT: RECORD COUNT: INTEGER; TEXT: LINE END;

PROCEDURE INITIALIZE (TEXT: LINE);
BEGIN
  HEADER := TEXT;
  ENDINPUT := TRUE;
  OUT.COUNT := 0;
END;

PROCEDURE ENTRY READ (VAR C: CHAR);
BEGIN
271  WITH INP DO
272  BEGIN
273    IF ENDINPUT THEN
274      BEGIN
275        OPERATOR.READ(HEADER, TEXT);
276        COUNT := 0;
277      END;
278      COUNT := COUNT + 1;
279      C := TEXT[COUNT];
280      ENDINPUT := (C = NL);
281    END;
282  END;
283
284  PROCEDURE ENTRY WRITE(C: CHAR);
285  BEGIN
286    WITH OUT DO
287      BEGIN
288        COUNT := COUNT + 1;
289        TEXT[COUNT] := C;
290        IF (C = NL) OR (COUNT = LINELIMIT) THEN
291          BEGIN
292            OPERATOR.WRITE(HEADER, TEXT);
293            COUNT := 0;
294          END;
295      END;
296  END;
297
298  PROCEDURE ENTRY RESET(TEXT: LINE);
299  BEGIN INITIALIZE(TEXT) END;
300
301  BEGIN INITIALIZE(‘UNIDENTIFIED:(;10:)’) END;
302
303
304  "################
305  # DISK #
306  ################
307
308  TYPE DISK =
309  CLASS(TYPEUSE: TYPERESOURCE);
310
311  "DISKDEVICE ADDED TO ALL PROCEDURES TO ALLOW - KSU
312    THE USE OF MULTIPLE VIRTUAL DISKS - KSU"
313
314  VAR OPERATOR: TERMINAL;
315
316  FUNCTION XLTESTAT(STATUS: UNIV INTEGER): INTEGER;
317  BEGIN
318    XLTESTAT := STATUS;
319  END;
320
321  PROCEDURE TRANSFER(COMMAND: IOOPERATION;
322    PAGEADDR: UNIV IOARG; VAR BLOCK: PAGE;
323    DISKDEVICE: IODEVICE);
"STATUS CHANGES MADE FOR OS/32-MT FOIBLES - KSU"

VAR PARAM: IOPARAM; RESPONSE: LINE;

WITH PARAM, OPERATOR DO

BEGIN

OPERATION := COMMAND;
ARG := PAGEADDR;
IO(BLOCK, PARAM, DISKDEVICE);
IF STATUS = ENDFILE THEN "KSU"

STATUS := COMPLETE; "KSU"

WHILE STATUS <> COMPLETE DO

BEGIN

RESPONSE[1] := CHR(XLTESTAT(STATUS) + ORD('0'));
WRITE('DISK: ERROR(:10:) ', RESPONSE);
READ('PUSH RETURN(:10:) ', RESPONSE);
IO(BLOCK, PARAM, DISKDEVICE);
END;
END;

PROCEDURE ENTRY READ(PAGEADDR: INTEGER;
VAR BLOCK: UNIV PAGE;
DISKDEVICE: IODEVICE);
BEGIN TRANSFER(INPUT, PAGEADDR, BLOCK, DISKDEVICE) END;

PROCEDURE ENTRY WRITE(PAGEADDR: INTEGER;
VAR BLOCK: UNIV PAGE;
DISKDEVICE: IODEVICE);
BEGIN TRANSFER(OUTPUT, PAGEADDR, BLOCK, DISKDEVICE, "\r\n";
BEGIN INIT OPERATOR(TYPEUSE) END;

"################################################################
# FILEMAP AND DISKFILE #
################################################################"

CONST MAPLENGTH = 255;
TYPE FILEMAP = RECORD
FILELENGTH: INTEGER;
PAGESET: ARRAY [1..MAPLENGTH] OF INTEGER
END;

TYPE DISKFILE =
CLASS(TYPEUSE: TYPERESOURCE);

"IODEVICE ADDED TO PROCEDURES TO ALLOW THE USE - KSU
OF MULTIPLE VIRTUAL DISKS - KSU"
379 VAR UNIT: DISK; MAP: FILEMAP; OPENED: BOOLEAN;
380
381 ENTRY LENGTH: INTEGER;
382
383 FUNCTION INCLUDES(PAGENO: INTEGER): BOOLEAN;
384 BEGIN
385   INCLUDES := OPENED &
386     (1 <= PAGENO) & (PAGENO <= LENGTH);
387 END;
388
389 PROCEDURE ENTRY OPEN(MAPADDR: INTEGER; DISK: IODEVICE);
390 BEGIN
391   UNIT.READ(MAPADDR, MAP, DISK);
392   LENGTH := MAP.FILELENGTH;
393   OPENED := TRUE;
394 END;
395
396 PROCEDURE ENTRY CLOSE;
397 BEGIN
398   LENGTH := 0;
399   OPENED := FALSE;
400 END;
401
402 PROCEDURE ENTRY READ(PAGENO: INTEGER;
403   VAR BLOCK: UNIV PAGE; DISK: IODEVICE);
404 BEGIN
405   IF INCLUDES(PAGENO) THEN
406     UNIT.READ(MAP.PAGESET(PAGENO), BLOCK, DISK);
407 END;
408
409 PROCEDURE ENTRY WRITE(PAGENO: INTEGER;
410   VAR BLOCK: UNIV PAGE; DISK: IODEVICE);
411 BEGIN
412   IF INCLUDES(PAGENO) THEN
413     UNIT.WRITE(MAP.PAGESET(PAGENO), BLOCK, DISK);
414 END;
415
416 BEGIN
417   INIT UNIT(TYPEUSE);
418   LENGTH := 0;
419   OPENED := FALSE;
420 END:
421
422 "#################################################################
423 # CATALOG STRUCTURE #
424 #################################################################
425
426
427 CONST IDLENGTH = 12;
428 TYPE IDENTIFIER = ARRAY [1..IDLENGTH] OF CHAR;
430
431 TYPE FILEKIND =
432   (EMPTY, SCRATCH, ASCII, SEQCODE, CONCODE);
TYPE FILEATTR = RECORD
  KIND: FILEKIND;
  ADDR: INTEGER;
  PROTECTED: BOOLEAN;
  NOTUSED: ARRAY [1..5] OF INTEGER
END;

TYPE CATENTRY = RECORD
  ID: IDENTIFIER;
  ATTR: FILEATTR;
  KEY, SEARCHLENGTH: INTEGER
END;

CONST CATPAGELENGTH = 16;
TYPE CATPAGE = ARRAY [1..CATPAGELENGTH] OF CATENTRY;

CONST CATADDR = 154;

"#################
# DISKTABLE #
################";

TYPE DISKTABLE =
  CLASS(TYPEUSE: TYPERESOURCE;
    CATADDR: INTEGER);

"DISKDEVICE PERMANENT VARIABLE AND DISK PARAMETER - KSU"
"ADDED TO ALLOW FOR USE OF MULTIPLE VIRTUAL DISKS - KSU"

VAR FILE: DISKFILE;
  PAGENO: INTEGER; DISKDEVICE: IODEVICE;
  BLOCK: CATPAGE;
ENTRY FLENGTH: INTEGER;

PROCEDURE ENTRY READ(I: INTEGER; VAR ELEM: CATENTRY;
  DISK: IODEVICE);
VAR INDEX: INTEGER;

BEGIN
  IF DISK <> DISKDEVICE THEN
    WITH FILE DO "KSU - ADDED TO SWITCH BETWEEN"
    "VIRTUAL DISKS"
      BEGIN
        CLOSE;
        FILE.OPEN(CATADDR, DISK);
        FLENGTH := FILE.LENGTH * CATPAGELENGTH;
        PAGENO := 0;
        DISKDEVICE := DISK;
      END;
      INDEX := (I - 1) DIV CATPAGELENGTH + 1:
IF PAGENO <> INDEX THEN
BEGIN
  PAGENO := INDEX;
  FILE.READ(PAGENO, BLOCK, DISK);
END;
ELEM := BLOCK[(I - 1) MOD CATPAGELength + 1];
END;

BEGIN
  INIT FILE(TYPEUSE);
  DISKDEVICE := PRIVATEDISK;
  FILE.OPEN(CATADDR, DISKDEVICE);
  FLENGTH := FILE.LENGTH * CATPAGELength;
  PAGENO := 0;
END;

"#################################################
# DISKCATALOG #
#################################################"

TYPE DISKCATALOG =
  MONITOR(TYPEUSE: TYPERESOURCE;
          DISKUSE: RESOURCE; CATADDR: INTEGER);

"THE DISK PARAMETER HAS BEEN ADDED TO ALLOW FOR - KSU
THE USE OF MULTIPLE VIRTUAL DISKS - KSU"

VAR TABLE: DISKTABLE;

FUNCTION HASH(ID: IDENTIFIER): INTEGER;

"THE VALIDITY OF THE HASH FUNCTION DEPENDS ON - KSU"
"THE FACT THAT ALL VIRTUAL DISKS HAVE THE SAME - KSU"
"CATALOG SIZE - KSU"

VAR KEY, I: INTEGER; C: CHAR;
BEGIN
  KEY := 1; I := 0;
  REPEAT
    I := I + 1; C := ID[I];
    IF C <> ' ' THEN
      KEY := KEY * ORD(C) MOD TABLE.FLENGTH + 1;
    UNTIL (C = ' ') OR (I = IDLENGTH);
  HASH := KEY;
END;

PROCEDURE ENTRY LOOKUP
  (ID: IDENTIFIER; VAR ATTR: FILEATTR;
   VAR FOUND: BOOLEAN; DISK: IODEVICE);

VAR KEY, MORE, INDEX: INTEGER; ELEM: CATENTRY;
BEGIN
  DISKUSE.REQUEST;
KEY := HASH(ID);
TABLE.READ(KEY, ELEM, DISK);
MORE := ELEM.SEARCHLENGTH;
INDEX := KEY; FOUND := FALSE;
WHILE NOT FOUND & (MORE > 0) DO
BEGIN
TABLE.READ(INDEX, ELEM, DISK);
IF ELEM.ID = ID THEN
BEGIN
ATTR := ELEM.ATTR; FOUND := TRUE
END
ELSE
BEGIN
IF ELEM.KEY = KEY THEN MORE := MORE - 1;
INDEX := INDEX MOD TABLE.FLENGTH + 1;
END;
END;
DISKUSE.RELEASE;
END;
BEGIN INIT TABLE(TYPEUSE, CATADDR) END;

"#################
# DATAFILE #
#################

TYPE DATAFILE =
CLASS(TYPEUSE: TYPERESOURCE; DISKUSE: RESOURCE;
CATALOG: DISKCATALOG);

VAR FILE: DISKFILE; OPENED: BOOLEAN;
ENTRY LENGTH: INTEGER;
PROCEDURE ENTRY OPEN(ID: IDENTIFIER; VAR FOUND: BOOLEAN);
VAR ATTR: FILEATTR;
BEGIN
CATALOG.LOOKUP(ID, ATTR, FOUND, PRIVATEDISK);
IF FOUND THEN
BEGIN
DISKUSE.REQUEST;
FILE.OPEN(ATTR.ADDR, PRIVATEDISK);
LENGTH := FILE.LENGTH;
DISKUSE.RELEASE;
END;
OPENED := FOUND;
END;
PROCEDURE ENTRY CLOSE;
BEGIN
FILE.CLOSE;
LENGTH := 0;
OPENED := FALSE;
END;
PROCEDURE ENTRY READ(PAGENO: INTEGER;
VAR BLOCK: UNIV PAGE);
BEGIN
IF OPENED THEN
BEGIN
DISKUSE.REQUEST;
FILE.READ(PAGENO, BLOCK, PRIVATEDISK);
DISKUSE.RELEASE;
END;
END;

PROCEDURE ENTRY WRITE(PAGENO: INTEGER;
VAR BLOCK: UNIV PAGE):
BEGIN
IF OPENED THEN
BEGIN
DISKUSE.REQUEST;
FILE.WRITE(PAGENO, BLOCK, PRIVATEDISK);
DISKUSE.RELEASE;
END;
END;
BEGIN
INIT FILE(TYPEUSE);
LENGTH:= 0;
OPENED:= FALSE;
END;

"########################################################################
# PROGSTORE AND PROGFILER
########################################################################"

TYPE PROGSTATE = (READY, NOTFOUND, NOTSEQ, TOOBIG);

CONST STORELENGTH1 = 40;
TYPE PROGSTORE1 =
ARRAY [1..STORELENGTH1] OF PAGE;

TYPE PROGFILER1 =
CLASS(TYPEUSE: TYPERESOURCE; DISKUSE: RESOURCE;
CATALOG: DISKCATALOG):

"PROGRAM FILES ARE FOUND BY FIRST SEARCHING THE     - KSU"
"PRIVATE DISK, IF NOT FOUND THERE, THE SYSTEM DISK - KSU"
"IS THEN SEARCHED     - KSU"

VAR FILE: DISKFILE;
ENTRY STORE: PROGSTORE1;
649  PROCEDURE ENTRY OPEN(ID: IDENTIFIER;
650            VAR STATE: PROGSTATE);
651                VAR ATTR: FILEATTR; FOUND: BOOLEAN; PAGENO: INTEGER;
652            DISK: IODEVICE;
653
654 BEGIN
655        CATALOG.LOOKUP(ID, ATTR, FOUND, PRIVATE_DISK);
656     IF FOUND THEN DISK := PRIVATE_DISK
657    ELSE BEGIN
658        CATALOG.LOOKUP(ID, ATTR, FOUND, SYSTEM_DISK);
659            DISK := SYSTEM_DISK;
660    END;
661 WITH DISKUSE, FILE, ATTR DO
662     IF NOT FOUND THEN
663       STATE := NOTFOUND ELSE
664       IF KIND <> SEQ_CODE THEN
665          STATE := NOTSEQ ELSE
666    BEGIN
667       REQUEST;
668       OPEN(ADDR, DISK);
669       IF LENGTH <= STORELENGTH1 THEN
670     BEGIN
671 FOR PAGENO := 1 TO LENGTH DO
672  READ(PAGENO, STORE[PAGENO], DISK);
673        STATE := READY;
674 END ELSE
675   STATE := TOOBIG;
676   CLOSE;
677   RELEASE;
678 END;
679 END;
680
681 BEGIN
682    INIT FILE(TYPEUSE);
683 END;
684
685
686 CONST STORELENGTH2 = 8;
687 TYPE PROGSTORE2 =
688   ARRAY [1..STORELENGTH2] OF PAGE;
689
690
691 TYPE PROGFILE2 =
692 CLASS(TYPEUSE: TYPE_RESOURCE; DISKUSE: RESOURCE;
693       CATALOG: DISKCATALOG);
694
695 VAR FILE: DISKFILE;
696
697 ENTRY STORE: PROGSTORE2;
698
699 PROCEDURE ENTRY OPEN(ID: IDENTIFIER;
700            VAR STATE: PROGSTATE);
701    VAR ATTR: FILEATTR; FOUND: BOOLEAN; PAGENO: INTEGER;
702            DISK: IODEVICE;
BEGIN
CATALOG.LOOKUP(ID, ATTR, FOUND, PRIVATEDISK);
IF FOUND THEN DISK := PRIVATEDISK
ELSE BEGIN
CATALOG.LOOKUP(ID, ATTR, FOUND, SYSTEMDISK);
DISK := SYSTEMDISK;
END;
WITH DISKUSE, FILE, ATTR DO
IF NOT FOUND THEN
STATE := NOTFOUND ELSE
IF KIND <> SEQCODE THEN
STATE := NOTSEQ ELSE
BEGIN
REQUEST;
OPEN(ADDR, DISK);
IF LENGTH <= STORELENGTH2 THEN
BEGIN
FOR PAGENO := 1 TO LENGTH DO
READ(PAGENO, STORE[PAGENO], DISK);
STATE := READY;
END ELSE
STATE := TOOBIG;
CLOSE;
RELEASE;
END;
END;
BEGIN
INIT FILE(TYPEUSE);
END;

"#####################################################
# RESULTTYPE AND PROGSTACK #
#####################################################"

TYPE RESULTTYPE =
(TERMINATED, OVERFLOW, POINTERERROR, RANGEERROR,
VARIANTERROR, HEAPLIMIT, STACKLIMIT, CODELIMIT,
TIMELIMIT, CALLERROR);

TYPE ATTRINDEX =
(CALLER, HEAPTOP, PROGLINE, PROGRESSULT, RUNTIME);

TYPE PROGSTACK =
MONITOR
CONST STACKLENGTH = 5;

VAR STACK: ARRAY [1..STACKLENGTH] OF
RECORD
  PROGID: IDENTIFIER;
HEAPPAD = INTEGER

TOP := 0..STACKLENGTH;

FUNCTION ENTRY SPACE: BOOLEAN;
BEGIN SPACE := (TOP < STACKLENGTH) END;

FUNCTION ENTRY ANY: BOOLEAN;
BEGIN ANY := (TOP > 0) END;

PROCEDURE ENTRY PUSH(ID: IDENTIFIER);
BEGIN
IF TOP < STACKLENGTH THEN
BEGIN
TOP := TOP + 1;
WITH STACK[TOP] DO
BEGIN
PROGID := ID;
HEAPPAD := ATTRIBUTE(HEAPTOP);
END;
END;
END;

PROCEDURE ENTRY POP
(VAR LINE, RESULT: UNIV INTEGER);
CONST TERMINATED = 0;
BEGIN
LINE := ATTRIBUTE(PROGLINE);
RESULT := ATTRIBUTE(PROGRRESULT);
IF RESULT <> TERMINATED THEN
SETHEAP(STACK[TOP].HEAPPAD);
TOP := TOP - 1;
END;

PROCEDURE ENTRY GET(VAR ID: IDENTIFIER);
BEGIN
IF TOP > 0 THEN ID := STACK[TOP].PROGID;
END;
BEGIN TOP := 0 END;

"# TASKKIND and ARGTYPE #
#------------------------------------------#
TYPE TASKKIND = (INPUTTASK, JOBTASK, OUTPUTTASK):
TYPE ARGTAG =
(NILTYPE, BOOLTYPE, INTTYPE, IDTYPE, PTRTYPE):
ARGTYPE = RECORD
  TAG: ARGTAG;
ARG: IDENTIFIER

END;

CONST MAXARG = 10;
TYPE ARGLIST = ARRAY [1..MAXARG] OF ARGTYPE;

TYPE ARGSEQ = (INP, OUT);

"###################
# ARGBUFFER #
###################"

TYPE ARGBUFFER =
MONITOR
VAR BUFFER: ARGTYPE; FULL: BOOLEAN;
SENDER, RECEIVER: QUEUE;

PROCEDURE ENTRY READ(VAR ARG: ARGTYPE);
BEGIN
IF NOT FULL THEN DELAY(RECEIVER);
ARG:= BUFFER; FULL:= FALSE;
CONTINUE(SENDER);
END;

PROCEDURE ENTRY WRITE(ARG: ARGTYPE);
BEGIN
IF FULL THEN DELAY(SENDER);
BUFFER:= ARG; FULL:= TRUE;
CONTINUE(RECEIVER);
END;
BEGIN FULL:= FALSE END:

"###################
# LINEBUFFER #
###################"

TYPE LINEBUFFER =
MONITOR
VAR BUFFER: LINE; FULL: BOOLEAN;
SENDER, RECEIVER: QUEUE;

PROCEDURE ENTRY READ(VAR TEXT: LINE);
BEGIN
IF NOT FULL THEN DELAY(RECEIVER);
TEXT:= BUFFER; FULL:= FALSE;
CONTINUE(SENDER);
END;

PROCEDURE ENTRY WRITE(TEXT: LINE);
BEGIN
IF FULL THEN DELAY(SENDER);
BUFFER := TEXT; FULL := TRUE;
CONTINUE(RECEIVER);
END;

BEGIN FULL := FALSE END;

"#############################
# PAGEBUFFER #
############################"

TYPE PAGEBUFFER = 
MONITOR
VAR BUFFER: PAGE; LAST, FULL: BOOLEAN;
SENDER, RECEIVER: QUEUE;

PROCEDURE ENTRY READ(VAR TEXT: PAGE; VAR EOF: BOOLEAN);
BEGIN
IF NOT FULL THEN DELAY(RECEIVER);
TEXT := BUFFER; EOF := LAST; FULL := FALSE;
CONTINUE(SENDER);
END;

PROCEDURE ENTRY WRITE(TEXT: PAGE; EOF: BOOLEAN);
BEGIN
IF FULL THEN DELAY(SENDER);
BUFFER := TEXT; LAST := EOF; FULL := TRUE;
CONTINUE(RECEIVER);
END;

BEGIN FULL := FALSE END;

"#############################
# CHARSTREAM #
############################"

TYPE CHARSTREAM = 
CLASS(BUFFER: PAGEBUFFER):

VAR TEXT: PAGE; COUNT: INTEGER; EOF: BOOLEAN;

PROCEDURE ENTRY READ(VAR C: CHAR);
BEGIN
IF COUNT = PAGELENGTH THEN
BEGIN
BUFFER.READ(TEXT, EOF);
COUNT := 0;
END;

END;
COUNT := COUNT + 1;
C := TEXT[COUNT];
IF C = EM THEN
BEGIN
    WHILE NOT EOF DO BUFFER.READ(TEXT, EOF);
    COUNT := PAGELENGTH;
END;
END;

PROCEDURE ENTRY INITREAD;
BEGIN COUNT := PAGELENGTH END;

PROCEDURE ENTRY WRITE(C: CHAR);
BEGIN
    COUNT := COUNT + 1;
    TEXT[COUNT] := C;
    IF (COUNT = PAGELENGTH) OR (C = EM) THEN
    BEGIN
        BUFFER.WRITE(TEXT, FALSE); COUNT := 0;
        IF C = EM THEN BUFFER.WRITE(TEXT, TRUE);
    END;
END;

PROCEDURE ENTRY INITWRITE;
BEGIN COUNT := 0 END;

BEGIN END;

CONST APCNT = 2;

CONST NOIDCNT = 2;
"NUMBER OF SEQUENTIAL PROGRAMS TO BE DELAYED
BECAUSE THERE ARE NO AVAILABLE DBIDS."

***************
* UNIVERSAL KERNEL TYPES *
***************

TYPE PAGE_BUFFER = ARRAY [1..512] OF CHAR;

TYPE ERRSET = SET OF 0..127;

TYPE SUBCHAR = SET OF ":0:.."(127:);

TYPE DISPLAY_MESSAGES = (COMMAND_ERROR,ERROR,ENTER_DEST_ID,
ENTER_SOURCE_ID,ENTER_TEXT,
ENTER_DATA_LENGTH,NI);

TYPE INPUTLINE = ARRAY [1..80] OF CHAR;

TYPE PHY_ID = ARRAY [1..4] OF CHAR;

TYPE LOG_ID = ARRAY [1..12] OF CHAR;
973 TYPE USER_SCB_RECORD = RECORD
974      REMOTE_ID : PHY_ID;
975      LOCAL_ID : PHY_ID;
976      COMMAND_LEN : INTEGER;
977      DATA_LEN : INTEGER;
978      IN_CODE : INTEGER;
979      OUT_CODE : INTEGER;
980      COMMAND : INTEGER;
981      COMMD_TEXT : ARRAY [1..128] OF CHAR
982 END;
983
984 TYPE NRC_RECORD = RECORD
985      LOGICAL_ID : LOG_ID;
986      PHYSICAL_ID : PHY_ID
987 END;
988
989 TYPE IDARRAY = ARRAY (.1..4.) OF PHY_ID;
990
991 "***********************
992 * MESSAGE SYSTEM MONITOR *
993 ***********************"
994
995 TYPE MESSAGE_SYSTEM = MONITOR;
996 VAR
997      SCB : USER_SCB_RECORD;
998      MSCONTENTS : PAGE_BUFFER;
999      HIT, FULL : BOOLEAN;
1000      SENDER, RECEIVER : QUEUE;
1001      I, J, RCODE : INTEGER;
1002      GET_ID_ARRAY : IDARRAY;
1003      HOST_ID, TEMPNAME : PHY_ID;
1004      COMMAND : LOG_ID;
1005
1006 PROCEDURE ENTRY CONNECT(ID.Rem : PHY_ID;
1007      ID.LOC : PHY_ID;
1008      VAR RCODE : INTEGER);
1009 BEGIN
1010 WITH SCB DO
1011 BEGIN
1012      REMOTE_ID := ID.Rem;
1013      LOCAL_ID := ID.LOC;
1014      IN_CODE := 3;
1015      OUT_CODE := 0;
1016      COMMAND := 0;
1017      COMMAND_LEN := 0;
1018      DATA_LEN := 0;
1019      SYSCUE(-2,RCODE,SCB,SCB);
1020      RCODE := OUT_CODE;
1021 END;
1022 END;
1023
1024 PROCEDURE ENTRY DISCONNECT(ID.Rem : PHY_ID;
1025      ID.LOC : PHY_ID;
1026      VAR RCODE : INTEGER);
1027 BEGIN
WITH SCB DO
  BEGIN
    REMOTE_ID := ID.Rem;
    LOCAL_ID := ID.Loc;
    IN_CODE := 4;
    COMMAND := 1;
    OUT_CODE := 0;
    SYSQUE(-2,RCODE,SCB,SCB);
    RCODE := OUT_CODE;
  END;
END;

PROCEDURE ENTRY SEND(ID.Rem:PHY_ID;
  ID_Loc:PHY_ID;
  VAR BUFFER:PAGE_BUFFER;
  LENGTH:INTEGER;
  VAR RCODE:INTEGER);
BEGIN
  WITH SCB DO
  BEGIN
    REMOTE_ID := ID.Rem;
    LOCAL_ID := ID.Loc;
    IN_CODE := 60;
    COMMAND := 2;
    COMMAND_LEN := 0;
    DATA_LEN := LENGTH;
    OUT_CODE := 0;
    SYSQUE(-2,RCODE,BUFFER,SCB);
    RCODE := OUT_CODE;
  END;
END;

PROCEDURE ENTRY RECEIVE(ID.Rem:PHY_ID;
  ID_Loc:PHY_ID;
  VAR BUFFER:PAGE_BUFFER;
  VAR LENGTH:INTEGER;
  VAR RCODE:INTEGER);
BEGIN
  WITH SCB DO
  BEGIN
    REMOTE_ID := ID.Rem;
    LOCAL_ID := ID.Loc;
    IN_CODE := 63;
    COMMAND := 3;
    OUT_CODE := 0;
    DATA_LEN := LENGTH;
    COMMAND_LEN := 128;
    SYSQUE(-2,RCODE,BUFFER,SCB);
    LENGTH := DATA_LEN;
    RCODE := OUT_CODE;
  END;
END;

PROCEDURE ENTRY GET_ID (VAR IDNAME:PHY_ID );
BEGIN
  I := I + 1;
  GET_ID_ARRAY(.I.) := TEMPNAME;
  IDNAME := TEMPNAME;
  TEMPNAME(.4.) := CHR(ORD(TEMPNAME(.4.)) + 1):
      "CMTP := CMTP + 1"
END;

PROCEDURE ENTRY PURGE (PURGENAME:PHY_ID):
BEGIN
  J := 0;
  REPEAT
    J := J + 1;
    IF PURGENAME = GET_ID_ARRAY(.J.)
      THEN HIT := TRUE;
    UNTIL HIT;
  IF J < I
    THEN REPEAT
      GET_ID_ARRAY(.J.) := GET_ID_ARRAY(.J + 1.);
      J := J + 1;
    UNTIL J = I;
  I := I - 1;
END;

BEGIN
  PULL := FALSE;
  HOST_ID := '
  FOR J := 1 TO 4 DO
    GET_ID_ARRAY(.J.) := '
  I := 0;
  SYSQUE(0, RCODE, HOST_ID, COMMAND);
  TEMPNAME := HOST_ID;
END;

TYPE ENV_FUNCTION = ARRAY (.1..6.) OF CHAR;
TYPE ENV_STAT = ARRAY (.1..4.) OF CHAR;
TYPE ENV_FILE_NAME = ARRAY (.1..4.) OF CHAR;
TYPE ENVREFER = ARRAY (.1..4.) OF CHAR;
TYPE ENV_LINKPATH = ARRAY (.1..8.) OF CHAR;
TYPE ENV_CTRL_FIELD = ARRAY (.1..10.) OF CHAR;
TYPE ENV_VELEMENTS = ARRAY (.1..60.) OF CHAR;
TYPE ENV_VAREA = ARRAY (.1..82.) OF CHAR;
TYPE ENV_ENDP = ARRAY (.1..4.) OF CHAR;
TYPE BOOLARRAY = ARRAY (.1..APCINT.) OF BOOLEAN;
1135 TYPE SPQUEUE = ARRAY (.1..NOIDCNT.) OF QUEUE;
1137 TYPE DBCMD = ARRAY (.1..38.) OF CHAR;
1139 TYPE WORK_BUF = RECORD
1140   BFUNCTION : ENV_FUNCTION;
1141   BSTAT : ENV_STAT;
1142   BFILE_NAME : ENV_FILE_NAME;
1143   BREFER : ENVREFER;
1144   BLINKPATH : ENV_LINKPATH;
1145   CTRL_FIELD : ENV_CTRL_FIELD;
1146   BLELEMENTS : ENV_VELEMENTS;
1147   BAREA : ENV_VAREA;
1148   BENDP : ENV_BENDP;
1149   APID : PHY_ID;
1150   DBID : PHY_ID;
1151   ACK : BOOLEAN;
1152   SPID : INTEGER;
1153   FILLER : ARRAY (.1..318.) OF CHAR
1154 END;
1156 TYPE ID_PAIR = ARRAY (.1..APCNT.) OF RECORD
1159   APID : PHY_ID;
1161   DBID : PHY_ID
1162 END;
1163 TYPE BEBUFF = RECORD
1165   BE_ID_REM : PHY_ID;
1166   BE_ID_LOC : PHY_ID;
1167   BE_BUFFER : PAGE_BUFFER;
1168   BE_LENGTH : INTEGER;
1169   BE_RCODE : INTEGER
1170 END;
1171 TYPE BUFBARRAY = ARRAY (.1..APCNT.) OF BEBUFF;
1173 TYPE ERRMESSAGE = ARRAY (.1..24.) OF CHAR;
1175 TYPE BEQUEUE = ARRAY (.1..APCNT.) OF QUEUE;
1177 TYPE HINTMON = MONITOR
1179 VAR
1180   I, CNT : INTEGER;
1181   SPBOOL, RTBOOL : BOOLARRAY;
1182   AP, SP : SPQUEUE;
1183   SPNEXT : FIFO;
1184   FULL, RFULL : BOOLEAN;
APSENDER, FRPRECEIVER : QUEUE;
BUFF, RETBUFF : WORK_BUFF;
RESPONSE : QUEUE;

PROCEDURE ENTRY SEND (SPNUM : INTEGER;
AP_BUFF : WORK_BUFF);
BEGIN
IF (CNT > APCNT) "NO ROOM FOR MORE APS"
AND NOT SPBOOL (.SPNUM.) "THE AP HAS NOT BEEN"
"SERVICED"
THEN DELAY (SP(.SPNEXT.ARRIVAL.));

IF FULL
THEN DELAY (APSENDER);
IF NOT SPBOOL (.SPNUM.)
THEN BEGIN "FIRST TIME FOR THE AP REQUEST"
SPBOOL (.SPNUM.) := TRUE;
CNT := CNT + 1;
END;
BUFF:=AP_BUFF;
BUFF.SPID := SPNUM;
BUFF.ACK := FALSE;
FULL := TRUE;
CONTINUE (FRPRECEIVER);
END;

PROCEDURE ENTRY RECEIVE (VAR HINT_BUFF : WORK_BUFF);
BEGIN
IF NOT FULL
THEN DELAY (FRPRECEIVER);
HINT_BUFF := BUFF;
FULL := FALSE;
CONTINUE (APSENDER);
END;

PROCEDURE ENTRY APRETURN (SPNUM : INTEGER;
VAR HR_BUFF : WORK_BUFF);
BEGIN
IF NOT RTBOOL (.SPNUM.) "DELAY AN AP IN THE PREFIX"
THEN DELAY (AP(.SPNUM.)) ":UNTIL A RSP"
HR_BUFF := RETBUFF;
RTBOOL (.SPNUM.) := FALSE;
IF HR_BUFF.ACK = TRUE
THEN BEGIN
CONTINUE (SP(.SPNEXT.DEPARTURE.));
"CONTINUE AN AP WAITING AT THE"
"PREFIX BECAUSE THERE WAS NO ROOM"
SPBOOL (.SPNUM.) := FALSE;
CNT := CNT - 1;
END;
RFULL := FALSE;
CONTINUE (RESPONSE);
END;

PROCEDURE ENTRY RSPRETURN (RSPBUFF : WORK_BUFF);
BEGIN
  IF FULL
    THEN DELAY (RESPONSE);
  RETBUFF := RSPBUFF;
  RTBOOL (.RETBUFF.SPID.) := TRUE;
  RFULL := TRUE;
  CONTINUE (AP(.RETBUFF.SPID.));
END:

BEGIN CNT := 1;
  FULL := FALSE;
  RFULL := FALSE;
  FOR I:= 1 TO APCNT DO
    BEGIN
      SPBOOL(.I.):=FALSE;
      SPBOOL(.I.):=FALSE;
    END;
  INIT SPNEXT (NOIDCNT);
END;

***************
* FORWARD MONITOR *
***************

TYPE FOR_REQ_MON = MONITOR
VAR
  FRCONTENTS : WORK_BUF;
  FULL : BOOLEAN;
  NRCRECEIVER, FRSENDER : QUEUE;

PROCEDURE ENTRY SEND (HINT_BUF : WORK_BUF);
BEGIN
  IF FULL
    THEN DELAY (FRSENDER);
  FRCONTENTS := HINT_BUF;
  FULL := TRUE;
  CONTINUE (NRCRECEIVER);
END:

PROCEDURE ENTRY RECEIVE (VAR RECVBUFF : WORK_BUF);
BEGIN
  IF NOT FULL
    THEN DELAY (NRCRECEIVER);
  RECVBUFF := FRCONTENTS;
  FULL := FALSE;
  CONTINUE (FRSENDER);
END;

BEGIN
  FULL := FALSE;
END;

***************
* FORWARD PROCESS *
***************
**----**

1297 TYPE FORWARD_REQUEST = PROCESS (FEHINT : HINTMON;
1298 FRMON : FOR_REQ_MON;
1299 TYPEUSE : TYPERESOURCE;
1300 MS : MESSAGE_SYSTEM);
1301
1302 VAR FOR_REQ_BUFF : WORK_BUFF;
1303 ID_Rem, ID_LOC : PHY_ID;
1304 BUFFER : PAGE_BUFFER;
1305 LENGTH, RCODE : INTEGER;
1306 I : INTEGER;
1307 CRT : TERMINAL;
1308 CRTMSG : TERMINALSTREAM;
1309 C : CHAR;
1310
1311 PROCEDURE WORK_BUFF_CONVERT(SENDBUFF:UNIV PAGE_BUFFER;
1312 VAR RETBUFF:PAGE_BUFFER);
1313 BEGIN
1314 RETBUFF := SENDBUFF;
1315 END;
1316
1317 BEGIN
1318 INIT
1319 CRT (TYPEUSE), CRTMSG (CRT);
1320 LENGTH := 512;
1321 RCODE := 19: "NO EVENTS QUEUED"
1322 CYCLE
1323 FEHINT.RECEIVE (FOR_REQ_BUFF);
1324 CRT.WRITE("FORWARD_REQUEST_PROCESS: (:10:),"
1325 ′RECEIVED FROM HINTMON. (:10:)′);
1326 IF (FOR_REQ_BUFF.BFUNCTION = ′SINON′) OR
1327 (FOR_REQ_BUFF.BFUNCTION = ′SINO′)
1328 THEN FRMON.SEND (FOR_REQ_BUFF)
1329 ELSE BEGIN
1330 ID_Rem := FOR_REQ_BUFF.DBID;
1331 ID_LOC := FOR_REQ_BUFF.APID;
1332 WORK_BUFF_CONVERT(FOR_REQ_BUFF, BUFFER);
1333 CRT.WRITE
1334 ′FORWARD_REQUEST_PROCESS: (:10:),"
1335 ′SENT TO MESSAGE SYSTEM. (:10:)′;
1336 MS.SEND (ID_Rem,
1337 ID_LOC,
1338 BUFFER,
1339 LENGTH,
1340 RCODE);
1341 "CHECK RCODE"
1342 END;
1343 END;
1344 END;
1345 END;
1346
1347 "***************
1348 * ID TABLE MONITOR *
1349 "***************
TYPE ID_TABLE_MON = MONITOR (TYPEUSE : TYPERESOURCE);
VAR
  HIT : BOOLEAN;
  ID_TABLE : ID_PAIR;
  I, J, K, IDNUM : INTEGER;
  C : CHAR;
  CRT : TERMINAL;
  CRTERROR : TERMINALSTREAM;
PROCEDURE ENTRY PUT (BE_APID, BE_DBID : PHY_ID);
BEGIN
  I := I + 1;
  WITH ID_TABLE(.I.) DO
    BEGIN
      APID := BE_APID;
      DBID := BE_DBID;
    END;
END;
PROCEDURE ENTRY REMOVE (BE_APID, BE_DBID : PHY_ID);
BEGIN
  J := 0;
  REPEAT
    J := J + 1;
    IF ID_TABLE(.J.).APID = BE_APID
    THEN HIT := TRUE;
    UNTIL (HIT) OR (J = I);
  IF HIT
  THEN BEGIN
    IF J = I
    THEN BEGIN
      ID_TABLE(.J.).APID := '';
      ID_TABLE(.J.).DBID := '';
    END
    ELSE BEGIN
      FOR K := J TO I DO
        BEGIN
          ID_TABLE(.K.).APID :=
          ID_TABLE(.K+1.).APID;
          ID_TABLE(.K.).DBID :=
          ID_TABLE(.K+1.).DBID;
        END;
    END;
  END;
  I := I - 1;
  HIT := FALSE;
  ELSE CRT.WRITE('ID_TABLE_MON:(:10:)',
                    'REMOVE ERROR IN ID_TABLE.(:10:)');
END;
PROCEDURE ENTRY IDREAD (VAR IDNUM : INTEGER;
  VAR REQIDS : ID_PAIR);
BEGIN
  IDNUM := I;
  REQIDS := ID_TABLE;
END;
BEGIN
    HIT:=FALSE;
    FOR I := 1 TO APCNT DO
        BEGIN
            WITH IDTABLE(.I.) DO
                BEGIN
                    APID := ′′;
                    DBID := ′′;
                END;
        END;
        I := 0;
        J := 0;
        K := 0;
        IDNUM := 0;
        INIT
            CRT (TYPEUSE), CRERROR (CRT);
    END;

"**************************
* NRC DIRECTORY MONITOR *
"**************************

TYPE DIRECTORY = MONITOR
VAR FRONTRC : PHY_ID;
BACKNRC : PHY_ID;
PROCEDURE ENTRY RECEIVE (VAR FENRC : PHY_ID;
                         VAR BENRC : PHY_ID);
BEGIN
    FENRC := FRONTRC;
    BENRC := BACKNRC;
END;
BEGIN
    FRONTRC := ′ABMF′;
    BACKNRC := ′ABAB′;
END;

"**************************
* FRONT_END CONNECT *
"**************************

TYPE FE_CONN_PROCESS =
    PROCESS (NRC_DIRECTORY : DIRECTORY;
             MS : MESSAGE_SYSTEM;
             TYPEUSE : TYPERESOURCE);
    VAR ID_LOC, IDREM : PHY_ID;
    RCODE, I : INTEGER;
    CONN_OK : BOOLEAN;
    CRT : TERMINAL;
    OK : OKTYPE;
BEGIN
RCODE := 19; "NO EVENTS QUEUED"
I := 0;
OK := 0;
CONN_OK := FALSE;
INIT
CRT(TYPEUSE);
NRCDIRECTORY.RECEIVE(ID_LOC, ID_REM);
REPEAT
MS.CONNECT(ID_LOC,
ID_REM,
RCODE);
IF RCODE=OK
THEN CONN_OK := TRUE
ELSE I := I+1;
UNTIL (CONN_OK) OR (I=5);
IF NOT CONN_OK
THEN CRT.WRITE
(´FE_CONN_PROCESS: (:10:)´,
´FE COULD NOT CONNECT WITH BE. (:10:)´)
ELSE IF I=5 THEN CRT.WRITE
(´FE_CONN_PROCESS: (:10:)´,
´CONNECT TIME OUT. (:10:)´)
ELSE CRT.WRITE
(´FE_CONN_PROCESS: (:10:)´,
´FE CONNECTED WITH BE. (:10:)´);
END;

***********************
* BACK_END CONNECT *
***********************

TYPE BE_CONN_PROCESS =
PROCESS (NRCDIRECTORY : DIRECTORY;
MS : MESSAGE_SYSTEM;
TYPEUSE : TYPE_RESOURCE);
VAR ID_LOC, ID_REM : PHY_ID;
RCODE, I : INTEGER;
CONN_OK : BOOLEAN;
CRT : TERMINAL;
OK : OKTYPE;

BEGIN
RCODE := 19; "NO EVENTS QUEUED"
I := 0;
CONN_OK := FALSE;
OK := 0;
INIT
CRT(TYPEUSE);
NRCDIRECTORY.RECEIVE(ID_REM, ID_LOC);
REPEAT
MS.CONNECT(ID_LOC,
ID_REM,
RCODE);
IF RCODE=OK
THEN CONN_OK := TRUE
ELSE I := I+1;
UNTIL (CONN_OK) OR (I=5):
IF NOT CONN_OK
THEN CRT.WRITE
("BE_CONN_PROCESS: (10:),
BE COULD NOT CONNECT WITH FE.(10:),")
ELSE IF I=5 THEN CRT.WRITE
("BE_CONN_PROCESS: (10:),
CONNECT TIME OUT.(10:),")
ELSE CRT.WRITE
("BE_CONN_PROCESS: (10:),
BE CONNECTED WITH FE.(10:),"):
END;
***************
* FRONT-END NRC *
***************

TYPE FE_NRC = PROCESS (FRMON : FOR_REQ_MON;
FEHINT : HINTMON;
FIDT : ID_TABLE_MON;
NRCDIRECTORY : DIRECTORY;
TYPEUSE : TYPERESOURCE;
MS : MESSAGE_SYSTEM);

VAR RETWORKBUFF : WORK_BUFF;
PEBUFF : WORK_BUFF;
BUFFER : PAGE_BUFFER;
TMP REM : PHY_ID;
ID_Rem : PHY_ID;
TMP LOC : PHY_ID;
ID Loc : PHY_ID;
TACK : BOOLEAN;
LENGTH : INTEGER;
I,RCODE : INTEGER;
OK : OKTYPE;
NAME : PHY_ID;
FENRC : PHY_ID;
BENRC : PHY_ID;
C : CHAR;
CRT : TERMINAL;
CRTERROR : TERMINALSTREAM;

PROCEDURE WORK_BUFF_CONVERT(SENDUFF : UNIV PAGE_BUFFER;
VAR RETBUFF : PAGE_BUFFER);
BEGIN
RETBUFF := SENDUFF;
END;

PROCEDURE PAGE_BUFF_CONVERT (BUFF : UNIV WORK_BUFF;
VAR RETWORK : WORK_BUFF);
BEGIN
RETWORK := BUFF;
END;
BEGIN
    INIT
    CRT(TYPEUSE), CRERROR(CRT);
    OK := 0;
    TACK := FALSE;
    LENGTH := 512;
    I := 0;
    RCODE := 19; "NO EVENTS QUEUED"
    CYCLE
    FRMON.RECEIVE(FEBUFF);
    CRT.WRITE('FE_NRC PROCESS:(:10:)','
    RECEIVED FROM FOR_REQ_MON(:10:)');
    IF FEBUFF.BFUNCTION = 'SINON'
    THEN BEGIN
        "CALL DIRECTORY FOR
        FE AND BE NRCS"
        NRCDIRECTORY.RECEIVE(FENRC,BENRC);
        ID_LOC := FENRC;
        ID_REM := BENRC;
        MS.GET_ID(NAME);
        FEBUFF.APID := NAME;
        WORK_BUFF_CONVERT(FEBUFF,BUFFER);
        MS.SEND(ID_REM,
                ID_LOC,
                BUFFER,
                LENGTH,
                RCODE);
        "CHECK RCODE"
        MS.RECEIVE(ID_REM,
                ID_LOC,
                BUFFER,
                LENGTH,
                RCODE);
    "CHECK RCODE"
        TMP_REM := ID_REM;
        TMP_LOC := ID_LOC;
        PAGE_BUFF_CONVERT(BUFFER,REWORKBUFF);
        ID_REM := REWORKBUFF.DBID;
        ID_LOC := REWORKBUFF.APID;
        MS.CONNECT(ID_REM,
                ID_LOC,
                RCODE);
        "CHECK RCODE"
        PIDT.PUT(ID_LOC, ID_REM);
        FEBUFF.APID := ID_LOC;
        FEBUFF.DBID := ID_REM;
        CRT.WRITE('FE_NRC PROCESS:(:10:)','
        FE SINON COMMAND.(:10:)');
    END
    ELSE IF FEBUFF.BFUNCTION = 'SINOF'
    THEN BEGIN
        WORK_BUFF_CONVERT(FEBUFF,BUFFER);
        "CALL DIRECTORY FOR
FE AND BE NRCS"
NRCDIRECTORY.RECEIVE(FENRC,BENRC);
1623     ID_LOC := FENRC;
1624     ID.Rem := BENRC;
1625     MS.SEND (ID.Rem,
1626     ID_LOC,
1627     BUFFER,
1628     LENGTH,
1629     RCODE);
1630     "CHECK RCODE"
1631     TMP.Rem := BENRC;
1632     TMP.LOC := FENRC;
1633     PAGE_BUFF_CONVERG(BUFFER,REWORKBUFF);
1634     ID.Rem := REWORKBUFF.DBID;
1635     ID.LOC := REWORKBUFF.APID;
1636     MS.DISCONNECT (ID.Rem,
1637     ID.LOC,
1638     RCODE);
1639     "CHECK RCODE"
1640     MS.RECEIVE (TMP.Rem,
1641     TMP.LOC,
1642     BUFFER,
1643     LENGTH,
1644     RCODE);
1645     IF RCODE <> OK
1646         THEN TACK := FALSE;
1647     FIDT.REMOVE(ID.LOC,ID.Rem);
1648     MS.PURGE(ID.LOC);
1649     IF RCODE<>OK
1650         THEN TACK:= FALSE;
1651     PAGE_BUFF_CONVERG(BUFFER,REWORKBUFF);
1652     REWORKBUFF.ACK := TACK;
1653     FEBUFF := REWORKBUFF;
1654     CRT.WRITE(‘FE_NRC_PROCESS:(:10:)’,
1655     ‘FE_SINOF COMMAND: (:10:)’):
1656         END
1657     ELSE CRT.WRITE
1658         (‘FE_NRC_PROCESS: (:10:)’,
1659         ‘RECEIVE ERROR IN FE_NRC. (:10:)’);
1660     FEBUFF.BSTAT:=‘****’;
1661     FEHINT.RSPRETURN (FEBUFF);
1662     TACK:= FALSE;
1663     END;
1664     END;
1665     END;
1666 "***************************"
1667 * RESPONSE PROCESS *
1668 "***************************"
1669
1670 TYPE MS_RESPONSE = PROCESS (FEHINT : HINTMON;
1671     FIDT : ID_TABLE_MON;
1672     MS : MESSAGE_SYSTEM;
1673     TYPEUSE : TYPERESOURCE);
1674 VAR I,J,IDNUM : INTEGER;
1675
MSG : BOOLEAN;
RSPIDS : ID_PAIR;
RSPBUFF : WORK_BUFF;
ID_REM : PHY_ID;
ID_LOC : PHY_ID;
C : CHAR;
CRT : TERMINAL;
CRTERROR : TERMINALSTREAM;
BUFFER : PAGE_BUFFER;
LENGTH : INTEGER;
rcode : INTEGER;
OK : OKTYPE;
NOT_DONE : NOT_DONE_TYPE;

PROCEDURE WORK_BUFF_CONVERT(SENDBUFF : UNIV PAGE_BUFFER; VAR RETBUFF : PAGE_BUFFER);
BEGIN
  RETBUFF := SENDBUFF;
END;

PROCEDURE PAGE_BUFF_CONVERT(BUFF : UNIV WORK_BUFF; VAR RETWORK : WORK_BUFF);
BEGIN
  RETWORK := BUFF;
END;

BEGIN
  INIT
    CRT (TYPEUSE), CRTERROR (CRT);
    MSG := FALSE;
    OK := 0;
    NOT_DONE := 28;
    i := 0;
    J := 0;
    IDNUM := 0;
    LENGTH := 512;
    RCODE := 19; "NO EVENTS QUEUED"
  CYCLE
    IDNUM := 0;
    REPEAT
      FIDT.IDREAD (IDNUM, RSPIDS);
      UNTIL IDNUM > 0:
      CRT.WRITE(’MS_RESPONSE PROCESS: (:10:)’,
                  ’RECEIVED FROM FE ID_TABLE_MON. (:10:)’);
      I := 1;
      REPEAT
        WITH RSPIDS (.I.,) DO
          BEGIN
            ID_LOC := APID;
            ID_REM := DBID;
          END;
        FOR J := 1 TO 5 DO
          WAIT;
        MS.RECEIVE (ID_REM,
ID_LOC,
BUFFER,
LENGTH,
RCODE);

IF RCODE = OK
THEN BEGIN
  MSG:=TRUE;
  CRT.WRITE
  ("MS_RESPONSE PROCESS: (:10:)",
   'RECEIVED FROM MESSAGE',
   'SYSTEM. (:10:)');
END
ELSE IF RCODE = NOT_DONE
  THEN I := I + 1
ELSE CRT.WRITE
  ("MS_RESPONSE",
   'PROCESS: (:10:)",
   'MESSAGE SYSTEM RECEIVE',
   'ERROR. (:10:)');

WAIT:
UNTIL (I=IDNUM) OR (MSG):
IF MSG
THEN BEGIN
  PAGE_BUFF_CONVERT(BUFFER,RSPBUFF);
  PFEHINT.RSPRETURN (RSPBUFF);
  MSG:= FALSE;
END;
END;

***************
* BACK-END NRC *
***************

TYPE BE_NRC = PROCESS (BIDT : ID_TABLE_MON;
  NRCDIRECTORY : DIRECTORY;
  TYPEUSE : TYPERESOURCE;
  MS : MESSAGE_SYSTEM);

VAR
  BE_NRC_BUFF : WORK_BUFF;
  TMP_ID.Rem,TMP_ID_LOC : PHY_ID;
  MSG,TACK : BOOLEAN;
  ID.Rem,ID_LOC : PHY_ID;
  BUFFER : PAGE_BUFFER;
  I,LENGTH,RCODE : INTEGER;
  OK : OKTYPE;
  NAME : PHY_ID;
  FENRC, BENRC : PHY_ID;
  C : CHAR;
  CRT : TERMINAL;
  CRTERROR : TERMINALSTREAM;

PROCEDURE WORK_BUFF_CONVERT(SENDBUFF:UNIV PAGE_BUFFER;
  VAR RETBUFF:PAGE_BUFFER);
BEGIN
1783  RETBUFF:=SENDBUFF;
1784  END;
1785
1786  PROCEDURE PAGE_BUFF_CONVERT(BUFF : UNIV WORK_BUFF;
1787                VAR RETWORK : WORK_BUFF);
1788  BEGIN
1789    RETWORK:=BUFF;
1790  END;
1791
1792  BEGIN
1793    INIT
1794        CRT(TYPEUSE),CRTERROR(CRT);
1795        OK:=0;
1796        MSG:=FALSE;
1797        TACK:=FALSE;
1798        I:=0;
1799        LENGTH:=512;
1800        RCODE:=19;  "NO EVENTS QUEUED"
1801        CYCLE
1802        REPEAT
1803            FOR I:= 1 TO 5 DO
1804                WAIT;
1805                "CALL DIRECTORY FOR FE_NRC AND BE_NRC"
1806                NRCDIRECTORY.RECEIVE(FENRC,BENRC);
1807                ID_LOC := BENRC;
1808                ID.Rem := FENRC;
1809                MS.RECEIVE (ID.Rem,
1810                    ID_LOC,
1811                    BUFFER,
1812                    LENGTH,
1813                    RCODE);
1814                IF RCODE = OK
1815                    THEN MSG:= TRUE;
1816                UNTIL MSG;
1817                TMP_ID.Rem := ID.Rem;  "FE_NRC"
1818                TMP_ID_LOC := ID_LOC;  "BE_NRC"
1819                PAGE_BUFF_CONVERT(BUFFER,BE_NRC_BUFF);
1820                WITH BE_NRC_BUFF DO
1821                BEGIN
1822                    IF BFUNACTION = 'SINON'
1823                        THEN BEGIN
1824                            MS.GET_ID (NAME);
1825                            DBID := NAME;
1826                            WORK_BUFF_CONVERT(BE_NRC_BUFF,BUFFER);
1827                            MS_SEND (ID.Rem,
1828                                ID_LOC,
1829                                BUFFER,
1830                                LENGTH,
1831                                RCODE);
1832                            "CHECK RCODE"
1833                            ID.Rem := APID;
1834                            ID_LOC := DBID;
1835                            MS.CONNECT (ID.Rem,
1836                                               ID_LOC,
"CHECK RCODE"

BIDT.PUT(APID,DBID);

CRT.WRITE(’BE_NRC_PROCESS:(:10:)’,
           ’BE SINON COMMAND.:(:10:)’);

END

ELSE IF BFUNCTI On = ’SINOF’
THEN BEGIN

    ID Rem := APID;
    ID Loc := DBID;
    MS.DISCONNECT (ID Rem,
                    ID Loc,
                    RCODE):

    IF RCODE = OK
    THEN Tack := TRUE
    ELSE Tack := FALSE;
    BIDT.REMOVE (APID, DBID);
    MS.PURGE (DBID);
    IF RCODE <> OK
    THEN Tack := FALSE;
    ID Rem := TMP_IDREM;
    ID Loc := TMP_IDLOC;
    ACK := Tack;
    WORK_BUFF_CONVERT(BE_NRC_BUFF,
                      BUFFER);
    MS.SEND (ID Rem,
             ID Loc,
             BUFFER,
             LENGTH,
             RCODE):

"CHECK RCODE"

CRT.WRITE
       (’BE_NRC_PROCESS:(:10:)’,
        ’BE SINOF COMMAND.:(:10:)’);

END

ELSE CRT.WRITE
       (’BE_NRC_PROCESS:(:10:)’,
        ’RECEIVE ERROR IN’,
        ’BE_NRC(:10:)’);

END;

MSG:=FALSE;
Tack:= FALSE;

END;

END;

********************
* BINT MONITOR *
********************

TYPE BINTMON = MONITOR

VAR BUSY,FULL : BOOLEAN;
Q : BEQUEUE;
NEXT,BNEXT : FIFO:
CONTENTS : BUFFARRAY;

PROCEDURE ENTRY SEND (BERB : BEBUFF);
BEGIN
CONTENTS (.BNEXT.ARRIVAL.) := BERB;
FULL := TRUE;
CONTINUE (Q(.NEXT.DEPARTURE.));
END;

PROCEDURE ENTRY RECEIVE (VAR DAT_BUFF : BEBUFF);
BEGIN
IF NOT FULL
THEN DELAY (Q(.NEXT.ARRIVAL.));
DAT_BUFF := CONTENTS (.BNEXT.DEPARTURE.);
IF BNEXT.EMPTY
THEN FULL := FALSE;
END;
BEGIN
FULL := FALSE; BUSY := FALSE;
INIT NEXT (APCNT), BNEXT (APCNT);
END;

***********************
* REQUEST PROCESS *
***********************

TYPE MS_REQUEST = PROCESS (BEINT : BINTMON;
BIDT : ID_TABLE_MON;
MS : MESSAGE_SYSTEM;
TYPEUSE : TYPERESOURCE);

VAR I,J,IDNUM : INTEGER;
MSG : BOOLEAN;
REQIDS : ID_PAIR;
DBBUFF : BEBUFF;
ID_REN : PHY_ID;
ID_LOC : PHY_ID;
C : CHAR;
CRT : TERMINAL;
CRTCERROR : TERMINALSTREAM;
OK : OKTYPE;
NOT_DONE : NOT_DONE_TYPE;
BUFFER : PAGE_BUFFER;
LENGTH,RCODE : INTEGER;

BEGIN
INIT
CRT (TYPEUSE), CRTCERROR (CRT);
MSG := FALSE;
OK := 0;
NOT_DONE := 28;
I:=0;
J:=0;
IDNUM:=0;
LENGTH := 512;

CYCLE
IDNUM := 0;

REPEAT
    BIDT.IDREAD (IDNUM, REQIDS);
UNTIL IDNUM > 0;

CRT.WRITE("MS_REQUEST PROCESS:(:10:)",
"RECEIVED FROM BE ID_TABLE_MON. (:10:)");
I := 1;

REPEAT
    WITH REQIDS (I) DO
BEGIN
    ID_REM := APID;
    ID_LOC := DBID;
END;

FOR J := 1 TO 5 DO
    WAIT;
    MS.RECEIVE (ID_REM,
    ID_LOC,
    BUFFER,
    LENGTH,
    RCODE);

    IF RCODE = OK
    THEN BEGIN
    MSG := TRUE;
    CRT.WRITE
    ("MS_REQUEST PROCESS:(:10:)",
    "RECEIVED FROM MESSAGE",
    "SYSTEM.(:10:)");
END
ELSE IF RCODE = NOT_DONE
    THEN I := I + 1
ELSE CRT.WRITE
    ("MS_REQUEST PROCESS:(:10:)",
    "MESSAGE SYSTEM RECEIVE",
    "ERROR.(:10:)");

    WAIT;
UNTIL (I = IDNUM) OR (MSG);

IF MSG
    THEN BEGIN
    WITH DBBUFF DO
BEGIN
    BE_ID_REM := ID_REM;
    BE_ID_LOC := ID_LOC;
    BE_BUFFER := BUFFER;
    BE_LENGTH := LENGTH;
    BE_RCODE := RCODE;
END;
    BEBIN.T.SEND (DBBUFF);
    MSG := FALSE;
END;
END;
1999
2000 "******************************************************************
2001 * CALL 'DATBASE' PROCESS *
2002 "******************************************************************
2003
2004 TYPE CALL_DATABASE = PROCESS (BEBINT : BINTMON;
2005   MS   : MESSAGE_SYSTEM;
2006   TYPEUSE : TYPERESOURCE;
2007   DB_BUFFER : LINEBUFFER);
2008
2009 VAR    DAT_BUFF : BEBUFF;
2010       DBCMD_BUFF : WORK_BUFF;
2011     ID_Rem, ID Loc : PHY_ID;
2012       BUFFER : PAGE_BUFFER;
2013     I, LENGTH, RCODE : INTEGER;
2014       DBLINE : LINE;
2015     DBFUNC, DBSTAT, DBFNAM, DBREF, DBLNK,
2016       DBCTR, DBLEM, DBAREA, DBRSP : DBCMDS;
2017     C : CHAR;
2018     CRT : TERMINAL;
2019   CRTRMSG : TERMINALSTREAM;
2020 PROCEDURE WORK_BUFF_CONVERT(SENDBUFF:UNIV PAGE_BUFFER;
2021   VAR RETBUFF:PAGE_BUFFER);
2022 BEGIN
2023   RETBUFF:=SENDBUFF;
2024 END;
2025
2026 PROCEDURE PAGE_BUFF_CONVERT(BUFF: UNIV WORK_BUFF;
2027   VAR RWORK : WORK_BUFF);
2028 BEGIN
2029   RWORK:=BUFF;
2030 END;
2031
2032 BEGIN
2033   INIT
2034     CRT (TYPEUSE), CRTRMSG (CRT);
2035     DBFUNC := 'THE DATA BASE FUNCTION IS '
2036     DBSTAT := 'THE DATA BASE STAT IS '
2037     DBFNAM := 'THE DATA BASE FILE NAME IS '
2038     DBREF := 'THE DATA BASE REFER IS '
2039     DBLNK := 'THE DATA BASE LINKPATH IS '
2040     DBCTR := 'THE DATA BASE CONTROL FIELD IS '
2041     DBLEM := 'THE DATA BASE VARIABLE ELEMENTS ARE '
2042     DBAREA := 'THE DATA BASE VARIABLE AREA IS '
2043     DBRSP := 'DATA BASE VARIABLES RETURNED HERE. '
2044     DBLINE (.131.) := CR;
2045     DBLINE (.132.) := NL;
2046   I:=0;
2047   LENGTH:=512;
2048   RCODE:=19; "NO EVENTS QUEUED"
2049 CYCLE
2050   BEBINRT.RECEIVE (DAT_BUFF);
2051   CRT.WRITE('CALL_DATABASE PROCESS: (:10:)´
2052       ´RECEIVED FROM BINTMON. (:10:)´);}
```
WITH DAT_BUFF DO
    BEGIN
        PAGE_BUFF_CONVERT(BE_BUFFER, DBCMD_BUFF);
        WITH DBCMD_BUFF DO
        BEGIN
            "CALL 'DATBAS' USING
                BFUNCTION BSTAT BFILE_NAME
                BREFER BLINKPATH BCTRL_FIELD
                BVELEMENTS BVARA:";
            FOR I := 1 TO 38 DO
                DBLINE (.I.) := DBFUNC (.I.);
            FOR I := 1 TO 6 DO
                DBLINE (.I + 38.) := BFUNCTION (.I.);
            FOR I := 45 TO 130 DO
                DBLINE (.I.) := ' ';
            DB_BUFFER.WRITE (DBLINE);
            BSTAT := '****';
            FOR I := 1 TO 38 DO
                DBLINE (.I.) := DBSTAT (.I.);
            FOR I := 1 TO 4 DO
                DBLINE (.I + 38.) := BSTAT (.I.);
            FOR I := 43 TO 130 DO
                DBLINE (.I.) := ' ';
            DB_BUFFER.WRITE (DBLINE);
            FOR I := 1 TO 38 DO
                DBLINE (.I.) := DBFNAME (.I.);
            FOR I := 1 TO 4 DO
                DBLINE (.I + 38.) := BFILE_NAME (.I.);
            FOR I := 43 TO 130 DO
                DBLINE (.I.) := ' ';
            DB_BUFFER.WRITE (DBLINE);
            FOR I := 1 TO 38 DO
                DBLINE (.I.) := DBREP (.I.);
            FOR I := 1 TO 4 DO
                DBLINE (.I + 38.) := BREFER (.I.);
            FOR I := 43 TO 130 DO
                DBLINE (.I.) := ' ';
            DB_BUFFER.WRITE (DBLINE);
            FOR I := 1 TO 38 DO
                DBLINE (.I.) := DBLNK (.I.);
            FOR I := 1 TO 8 DO
                DBLINE (.I + 38.) := BLINKPATH (.I.);
            FOR I := 47 TO 130 DO
                DBLINE (.I.) := ' ';
            DB_BUFFER.WRITE (DBLINE);
            FOR I := 1 TO 38 DO
                DBLINE (.I.) := DBCTR (.I.);
            FOR I := 1 TO 10 DO
                DBLINE (.I + 38.) := BCTRL_FIELD (.I.);
            FOR I := 49 TO 130 DO
                DBLINE (.I.) := ' ';
            DB_BUFFER.WRITE (DBLINE);
            FOR I := 1 TO 38 DO
                DBLINE (.I.) := DBELEM (.I.);
```
FOR I := 1 TO 60 DO
   DBLINE (.I + 38.) := BVELEMENTS (.I.);
FOR I := 99 TO 130 DO
   DBLINE (.I.) := ' ';
DB_BUFFER.WRITE (DBLINE);
FOR I := 1 TO 36 DO
   BVAREA (.I.) := DBRSP (.I.);
FOR I := 1 TO 46 DO
   BVAREA (.I + 36.) := ' ';
FOR I := 1 TO 38 DO
   DBLINE (.I.) := DBAREA (.I.);
FOR I := 1 TO 82 DO
   DBLINE (.I + 38.) := BVAREA (.I.);
FOR I := 121 TO 130 DO
   DBLINE (.I.) := ' ';
DB_BUFFER.WRITE (DBLINE);
END;
IDREM := BE_IDREM;
IDLOC := BE_IDLOC;
WORK_BUFF_CONVERT (DBCMD_BUFF, BUFFER);
LENGTH := BE_LENGTH;
MS.SEND (IDREM,
   IDLOC,
   BUFFER,
   LENGTH,
   RCODE);
"CHECK RCODE"
END;

"####################################
# JOBPROCESS #
####################################"

TYPE JOBPROCESS =
PROCESS
(TYPEUSE: TYPERESOURCE; DISKUSE: RESOURCE;
CATALOG: DISKCATALOG;
INBUFFER, OUTBUFFER: PAGEBUFFER;
INREQUEST, INRESPONSE,
OUTREQUEST, OUTRESPONSE: ARGBUFFER;
STACK: PROGSTACK; FEHINT : HINTMON);
"PROGRAM DATA SPACE = " +9000
"KSU - ALLOWS MAX DATA ADDRESSABILITY"

CONST MAXFILE = 2;
TYPE FILE = 1..MAXFILE;

VAR
2161 OPERATOR: TERMINAL; OPSTREAM: TERMINALSTREAM;
2162
2163 INSTREAM, OUTSTREAM: CHARSTREAM;
2164
2165 FILES: ARRAY [FILE] OF DATAFILE;
2166
2167 CODE: PROGFILE1;
2168
2169 SEND_BUFF, HR_BUFF: WORK_BUFF;
2170
2171 PROGRAM JOB(VAR PARAM: ARGLIST; STORE: PROGSTORE1);
2172 ENTRY READ, WRITE, OPEN, CLOSE, GET, PUT, LENGTH,
2173 MARK, RELEASE, IDENTIFY, ACCEPT, DISPLAY, READPAGE,
2174 WRITEPAGE, READLINE, WRITELINE, READARG, WRITETARG,
2175 LOOKUP, IOMove, TASK, RUN, HINT;
2176
2177 PROCEDURE CALL(ID: IDENTIFIER; VAR PARAM: ARGLIST;
2178 VAR LINE: INTEGER; VAR RESULT: RESULTTYPE);
2179 VAR STATE: PROGSTATE; LASTID: IDENTIFIER;
2180 BEGIN
2181     WITH CODE, STACK DO
2182     BEGIN
2183         LINE := 0;
2184         OPEN(ID, STATE);
2185         IF (STATE = READY) & SPACE THEN
2186         BEGIN
2187             PUSH(ID);
2188             JOB(PARAM, STORE);
2189             POP(LINE, RESULT);
2190         END ELSE
2191         IF STATE = TOOBIG THEN RESULT := CODELIMIT
2192         ELSE RESULT := CALLERROR;
2193         IF ANY THEN
2194         BEGIN GET(LASTID); OPEN(LASTID, STATE) END;
2195         END;
2196     END;
2197
2198 PROCEDURE ENTRY READ(VAR C: CHAR);
2199 BEGIN INSTREAM.READ(C) END;
2200
2201 PROCEDURE ENTRY WRITE(C: CHAR);
2202 BEGIN OUTSTREAM.WRITE(C) END;
2203
2204 PROCEDURE ENTRY OPEN
2205 (F: FILE; ID: IDENTIFIER; VAR FOUND: BOOLEAN);
2206 BEGIN FILES[F].OPEN(ID, FOUND) END;
2207
2208 PROCEDURE ENTRY CLOSE(F: FILE);
2209 BEGIN FILES[F].CLOSE END;
2210
2211 PROCEDURE ENTRY GET(F: FILE; P: INTEGER; VAR BLOCK: PAGE);
2212 VAR NEWTIME: INTEGER;
2213 BEGIN
2214     FILES[F].READ(P, BLOCK);
END;

PROCEDURE ENTRY PUT(F: FILE; P: INTEGER;
    VAR BLOCK: PAGE);
BEGIN FILES[F].WRITE(P, BLOCK) END;

FUNCTION ENTRY LENGTH(F: FILE): INTEGER;
BEGIN LENGTH := FILES[F].LENGTH END;

PROCEDURE ENTRY MARK(VAR TOP: INTEGER);
BEGIN TOP := ATTRIBUTE(HEAPTOP) END;

PROCEDURE ENTRY RELEASE(TOP: INTEGER);
BEGIN SETHEAP(TOP) END;

PROCEDURE ENTRY IDENTIFY(HEADER: LINE);
BEGIN OPSTREAM.RESET(HEADER) END;

PROCEDURE ENTRY ACCEPT(VAR C: CHAR);
BEGIN OPSTREAM.READ(C) END;

PROCEDURE ENTRY DISPLAY(C: CHAR);
BEGIN OPSTREAM.WRITE(C) END;

PROCEDURE ENTRY READPAGE(VAR BLOCK: PAGE;
    VAR EOF: BOOLEAN);
BEGIN INBUFFER.READ(BLOCK, EOF) END;

PROCEDURE ENTRY WRITEPAGE(BLOCK: PAGE; EOF: BOOLEAN);
BEGIN OUTBUFFER.WRITE(BLOCK, EOF) END;

PROCEDURE ENTRY READLINE(VAR TEXT: LINE);
BEGIN END;

PROCEDURE ENTRY WRITELINE(TEXT: LINE);
BEGIN END;

PROCEDURE ENTRY READARG(S: ARGSEQ; VAR ARG: ARGTYPE);
BEGIN
    IF S = INP THEN INRESPONSE.READ(ARG)
    ELSE OUTRESPONSE.READ(ARG);
END;

PROCEDURE ENTRY WRITEARG(S: ARGSEQ; ARG: ARGTYPE);
BEGIN
    IF S = INP THEN INREQUEST.WRITE(ARG)
    ELSE OUTREQUEST.WRITE(ARG);
END;

PROCEDURE ENTRY LOOKUP
    (ID: IDENTIFIER; VAR ATTR: FILEATTR; VAR FOUND: BOOLEAN);
"CATALOG IS SEARCHED FOR PRIVATE DISK FOLLOWED - KSU"
"BY SYSTEM DISK ONLY SEQUENCE CODE FILES ARE FOUND ON - KSU"
"THE SYSTEM DISK - KSU"

BEGIN
CATALOG.LOOKUP(ID, ATTR, FOUND, PRIVATE_DISK);
IF NOT FOUND THEN BEGIN
CATALOG.LOOKUP(ID, ATTR, FOUND, SYSTEM_DISK);
FOUND := FOUND AND (ATTR.KIND = SEQUENCE_CODE);
END;
END;

PROCEDURE ENTRY IOTRANSFER
(DEVICE: IO_DEVICE; VAR PARAM: IOPARAM; VAR BLOCK: PAGE):
BEGIN
IF (DEVICE = PRIVATE_DISK) OR (DEVICE = SYSTEM_DISK) "KSU"
THEN
BEGIN
DISKUSE.REQUEST;
IO(BLOCK, PARAM, DEVICE);
WITH PARAM DO
IF STATUS = ENDFILE THEN "KSU"
STATUS := COMPLETE; "KSU"
DISKUSE.RELEASE;
END ELSE
IO(BLOCK, PARAM, DEVICE);
END;

PROCEDURE ENTRY IOMOVE(DEVICE: IO_DEVICE;
VAR PARAM: IOPARAM);
BEGIN IO(PARAM, PARAM, DEVICE) END;

FUNCTION ENTRY TASK: TASK_KIND;
BEGIN TASK := JOBTASK END;

PROCEDURE ENTRY RUN
(ID: IDENTIFIER; VAR PARAM: ARG_LIST;
VAR LINE: INTEGER; VAR RESULT: RESULT_TYPE);
BEGIN CALL(ID, PARAM, LINE, RESULT) END;

PROCEDURE ENTRY HINT(HFUNCTION : ENV_FUNCTION;
VAR STAT : ENV_STAT;
FILE_NAME : ENV_FILE_NAME;
REFER : ENVREFER;
LINKPATH : ENV_LINKPATH;
CTRL_FIELD : ENV_CTRL_FIELD;
VELEMENTS : ENV_VELEMENTS;
VAR_VAR : ENV_VAR;
ENDP : ENV_ENDP);
VAR SPNUM : INTEGER;
BEGIN
SPNUM := 1;
WITH SEND_BUFF DO
BEGIN
BFUNCTION := HFUNCTION;
BFILE_NAME := FILE_NAME;
REFER := REFER;
BLINKPATH := LINKPATH;
CTRL_FIELD := CTRL_FIELD;
BELEMENTS := VELEMENTS;
ENDP := ENDP;

FEINTSEND (SPNUM, SEND_BUFF);
FEINT.APRETURN (SPNUM, HR_BUFF);
STAT := HR_BUFF.BSTAT;
VAREA := HR_BUFF.BVAREA;
SEND_BUFF := HR_BUFF;
END;

PROCEDURE INITIALIZE;
VAR I: INTEGER; PARAM: ARGLIST;
LINE: INTEGER; RESULT: RESULTTYPE;
BEGIN
INIT OPERATOR(TYPEUSE), OPSTREAM(OPTION),
INSTREAM(INBUFFER), OUTSTREAM(OUTBUFFER);
INSTREAM.INITREAD; OUTSTREAM.INITWRITE;
FOR I := 1 TO MAXFILE DO
INIT FILES[I](TYPEUSE, DISKUSE, CATALOG);
INIT CODE(TYPEUSE, DISKUSE, CATALOG);
WITH PARAM[2] DO
BEGIN TAG := IDTYPE; ARG := 'CONSOLE '
END;
CALL('DO ', PARAM, LINE, RESULT);
OPERATOR.WRITE('JOBPROCESS:(10):',
"TERMINATED (10:)");
END;
BEGIN INITIALIZE END;

"########################
# IOPROCESS #
####################"

TYPE IOPROCESS =
PROCESS
(TYPEUSE: TYPERESOURCE; DISKUSE: RESOURCE;
CATALOG: DISKCATEGORY; SLOWIO: LINEBUFFER;
BUFFER: PAGEBUFFER; REQUEST, RESPONSE: ARGBUFFER;
STACK: PROGSTACK; IOTASK: TASKKind);

"PROGRAM DATA SPACE = " +2000

TYPE FILE = 1..1;
VAR
OPERATOR: TERMINAL; OPSTREAM: TERMINALSTREAM;
2377 IOSTREAM: CHARSTREAM; IOFILE: DATAFILE;
2379 CODE: PROGFILE2;
2381 PROGRAM DRIVER(VAR PARAM: ARGLIST; STORE: PROGSTORE2);
2383 ENTRY READ, WRITE, OPEN, CLOSE, GET, PUT, LENGTH,
2384 MARK, RELEASE, IDENTIFY, ACCEPT, DISPLAY, READPAGE,
2385 WRITEPAGE, READLINE, WRITELINE, READARG, WRITEARG,
2386 LOOKUP, IOMOVE, TASK, RUN;
2387 PROCEDURE CALL(ID: IDENTIFIER; VAR PARAM: ARGLIST;
2389 VAR LINE: INTEGER; VAR RESULT: RESULTTYPE);
2390 VAR STATE: PROGSTATE; LASTID: IDENTIFIER;
2391 BEGIN
2392 WITH CODE, STACK DO
2393 BEGIN
2394 LINE := 0;
2395 OPEN(ID, STATE);
2396 IF (STATE = READY) & SPACE THEN
2397 BEGIN
2398 PUSH(ID);
2399 DRIVER(PARAM, STORE);
2400 POP(LINE, RESULT);
2401 END ELSE
2402 IF STATE = TOOBIG THEN RESULT := CODELIMIT
2403 ELSE RESULT := CALLERROR;
2404 IF ANY THEN
2405 BEGIN GET(LASTID); OPEN(LASTID, STATE) END;
2406 END;
2407 END;
2408
2409 PROCEDURE ENTRY READ(VAR C: CHAR);
2410 BEGIN IOSTREAM.READ(C) END;
2411
2412 PROCEDURE ENTRY WRITE(C: CHAR);
2413 BEGIN IOSTREAM.WRITE(C) END;
2414
2415 PROCEDURE ENTRY OPEN
2416 (F: FILE; ID: IDENTIFIER; VAR FOUND: BOOLEAN);
2417 BEGIN IOFILE.OPEN(ID, FOUND) END;
2418
2419 PROCEDURE ENTRY CLOSE(F: FILE);
2420 BEGIN IOFILE.CLOSE END;
2421
2422 PROCEDURE ENTRY GET(F: FILE; P: INTEGER;
2423 VAR BLOCK: PAGE);
2424 BEGIN IOFILE.READ(P, BLOCK) END;
2425
2426 PROCEDURE ENTRY PUT(F: FILE; P: INTEGER;
2427 VAR BLOCK: PAGE);
2428 BEGIN IOFILE.WRITE(P, BLOCK) END;
2429
2430 FUNCTION ENTRY LENGTH(F: FILE): INTEGER;
BEGIN LENGTH := IOFILE.LENGTH END;

PROCEDURE ENTRY MARK(VAR TOP: INTEGER);
BEGIN TOP := ATTRIBUTE(HEAPTOP) END;

PROCEDURE ENTRY RELEASE(TOP: INTEGER);
BEGIN SETHEAP(TOP) END;

PROCEDURE ENTRY IDENTIFY(HEADER: LINE);
BEGIN OPSTREAM.RESET(HEADER) END;

PROCEDURE ENTRY ACCEPT(VAR C: CHAR);
BEGIN OPSTREAM.READ(C) END;

PROCEDURE ENTRY DISPLAY(C: CHAR);
BEGIN OPSTREAM.WRITE(C) END;

PROCEDURE ENTRY READPAGE(VAR BLOCK: PAGE;
VAR EOF: BOOLEAN);
BEGIN BUFFER.READ(BLOCK, EOF) END;

PROCEDURE ENTRY WRITEPAGE(BLOCK: PAGE; EOF: BOOLEAN);
BEGIN BUFFER.WRITE(BLOCK, EOF) END;

PROCEDURE ENTRY READLINE(VAR TEXT: LINE);
BEGIN SLOWIO.READ(TEXT) END;

PROCEDURE ENTRY WRITELINE(TEXT: LINE);
BEGIN SLOWIO.WRITE(TEXT) END;

PROCEDURE ENTRY READARG(S: ARGSEQ; VAR ARG: ARGTYPE);
BEGIN REQUEST.READ(ARG) END;

PROCEDURE ENTRY WRITEARG(S: ARGSEQ; ARG: ARGTYPE);
BEGIN RESPONSE.WRITE(ARG) END;

PROCEDURE ENTRY LOOKUP
(ID: IDENTIFIER; VAR ATTR: FILEATTR; VAR FOUND: BOOLEAN);

"PRIVATE DISK IS SEARCHED FIRST, FOLLOWED - KSU"
"BY SYSTEM DISK ONLY SEQUENCES ARE FOUND ON - KSU"
"THE SYSTEM DISK - KSU"
BEGIN
CATALOG.LOOKUP(ID, ATTR, FOUND, PRIVATEDISK);
IF NOT FOUND THEN BEGIN
CATALOG.LOOKUP(ID, ATTR, FOUND, SYSTEMDISK);
FOUND := FOUND AND (ATTR.KIND = SEQUENCES);
END;
END;

PROCEDURE ENTRY IOTRANSFER
(DEVICE: IODEVICE; VAR PARAM: IOPARAM; VAR BLOCK: PAGE);
BEGIN
IF (DEVICE = PRIVATEDISK) OR (DEVICE = SYSTEMDISK) "KSU"
THEN
BEGIN
  DISKUSE.REQUEST;
  IO(BLOCK, PARAM, DEVICE);
  WITH PARAM DO
    IF STATUS = ENDFILE THEN
      STATUS := COMPLETE;
    DISKUSE.RELEASE;
  END ELSE
  IO(BLOCK, PARAM, DEVICE);
END;

PROCEDURE ENTRY IOMOVE DEVICE: IODEVICE;
  VAR PARAM: IOPARAM);
BEGIN IO(PARAM, PARAM, DEVICE) END;

FUNCTION ENTRY TASK: TASKKIND;
BEGIN TASK := IOTASK END;

PROCEDURE ENTRY RUN
(ID: IDENTIFIER; VAR PARAM: ARGLIST;
VAR LINE: INTEGER; VAR RESULT: RESULTTYPE);
BEGIN CALL(ID, PARAM, LINE, RESULT) END;

PROCEDURE INITIALIZE;
VAR PARAM: ARGLIST; LINE: INTEGER; RESULT: RESULTTYPE;
BEGIN
  INIT OPERATOR(TYPEUSE), OPSTREAM(OPERATOR),
  IOSTREAM(BUFFER),
  IOPFILE(TYPEUSE, DISKUSE, CATALOG),
  CODE(TYPEUSE, DISKUSE, CATALOG);
  IF IOTASK = INPUTTASK THEN IOSTREAM.INITWRITE
  ELSE IOSTREAM.INITREAD;
  CALL(′IO′, PARAM, LINE, RESULT);
  OPERATOR.WRITE(′IOPROCESS: (:10:)′,
  ′TERMINATED (:10:)′);
END:

BEGIN INITIALIZE END:

"##################################
# CARDPROCESS #
##################################"

TYPE CARDPROCESS =
PROCESS
  (TYPEUSE: TYPERESOURCE; BUFFER: LINEBUFFER);

"CARDSPROCESS HAS BEEN MODIFIED FOR THE USE OF A" - KSU
"VIRTUAL READER END FILE OR END MEDIUM CAUSES THE" - KSU
"PROCESS TO TERMINATE" - KSU
2539 VAR OPERATOR: TERMINAL; TEXT: LINE;
2540 PARAM: IOPARAM; OK, ENDF: BOOLEAN;
2541 BEGIN
2542 INIT OPERATOR(TYPEUSE);
2543 PARAM.OPERATION:= INPUT;
2544 ENDF:= FALSE;
2545 REPEAT
2546 REPEAT
2547 IO(TEXT, PARAM, CARDDEVICE);
2548 CASE PARAM.STATUS OF
2549 COMPLETE:
2550 OK:= TRUE;
2551 INTERVENTION, FAILURE:
2552 BEGIN OK:= FALSE; WAIT END;
2553 ENDFILE, ENDMEDIUM: "KSU - SUPPORTS"
2554 "VIRTUAL READER"
2555 BEGIN OK:= TRUE; ENDF:= TRUE; END;
2556 TRANSMISSION:
2557 BEGIN
2558 OPERATOR.WRITE(\"CARDS: (:10:)\",
2559 \"ERROR(:10:)\"");
2560 OK:= FALSE;
2561 END
2562 END
2563 END;
2564 UNTIL OK;
2565 BUFFER.WRITE(TEXT);
2566 UNTIL ENDF;
2567 OPERATOR.WRITE("CARDS: (:10:)", "KSU"
2568 \"TERMINATED(:10:) \") ; "KSU"
2569 END;
2570
2571 "#########################
2572 # PRINTERPROCESS #
2573 #########################
2574
2575
2576 TYPE PRINTERPROCESS =
2577 PROCESS
2578 (TYPEUSE: TYPERESOURCE; BUFFER: LINEBUFFER);
2579
2580 VAR OPERATOR: TERMINAL; PARAM: IOPARAM;
2581 TEXT: LINE;
2582 BEGIN
2583 INIT OPERATOR(TYPEUSE):
2584 PARAM.OPERATION:= OUTPUT;
2585 CYCLE
2586 BUFFER.READ(TEXT);
2587 IO(TEXT, PARAM, PRINTDEVICE);
2588 IF PARAM.STATUS <> COMPLETE THEN
2589 BEGIN
2590 OPERATOR.WRITE(\"PRINTER: (:10:)\",
2591 END;
2593 'INSPECT(:10:)');
2594 REPEAT
2595 WAIT;
2596 IO(TEXT, PARAM, PRINTDEVICE);
2597 UNTIL PARAM.STATUS = COMPLETE;
2598 END;
2599 END;
2600 END;
2601
2602 "################################
2603 # LOADERPROCESS #
2604 ################################
2605 TYPE LOADERPROCESS=
2606 PROCESS(DISKUSE: RESOURCE);
2607 CONST SOLOADDR = 24;
2608 VAR PARAM: IOPARAM;
2609
2610 PROCEDURE INITIALIZE(PAGENO: UNIV IOARG):
2611 BEGIN
2612 WITH PARAM DO
2613 BEGIN
2614 OPERATION:= CONTROL;
2615 ARG:= PAGENO;
2616 END;
2617 END;
2618
2619 BEGIN
2620 INITIALIZE(SOLOADDR);
2621 "AWAIT BEL SIGNAL"
2622 IO(PARAM, PARAM, TYPEDEVICE);
2623 "LOAD SOLO SYSTEM"
2624 DISKUSE.REQUEST;
2625 IO(PARAM, PARAM, PRIVATE_DISK);
2626 DISKUSE.RELEASE;
2627 END;
2628
2629 "################################
2630 # INITIAL PROCESS #
2631 ################################
2632 VAR
2633 TYPEUSE: TYPEROUSE;
2634 DISKUSE: RESOURCE: CATALOG: DISKCATEGORY;
2635 INBUFFER, OUTBUFFER: PAGEBUFFER;
2636 CARDBUFFER, PRINTERBUFFER: LEBUFFER;
2647 INREQUEST, INRESPONSE,
2648 OUTREQUEST, OUTRESPONSE: ARGBUFFER;
2649
2650 INSTACK, OUTSTACK, JOBSTACK: PROGSTACK;
2651
2652 READER: CARDPROCESS; WRITER: PRINTERPROCESS;
2653
2654 PRODUCER, CONSUMER: IOPROCESS; MASTER: JOBPROCESS;
2655
2656 OPERATOR: TERMINAL;
2657
2658 WATCHDOG : LOADERPROCESS;
2659
2660 MS: MESSAGE_SYSTEM;
2661
2662
2663 FEHINT: HINTMON;
2664
2665 NRCDIRECTORY: DIRECTORY;
2666
2667 FRP : FORWARD_REQUEST;
2668 FRMON : FOR_REQ_MON;
2669 FIDT : ID_TABLE_MON;
2670 FN P : FE_NRC;
2671 MRESP : MS_RESPONSE;
2672 FCONN P: FE_CONN_PROCESS;
2673
2674 BIDT: ID_TABLE_MON;
2675 BNP : BE_NRC;
2676 MRQSTP : MS_REQUEST;
2677 BEBINT : BINTMON;
2678 CDP : CALL_DATABASE;
2679 BCONN P: BE_CONN_PROCESS;
2680
2681 BEGIN
2682 INIT
2683 TYPEUSE, OPERATOR(TYPEUSE):
2684 OPERATOR.WRITE ('INIT : (:10:)', 'TYPEUSE(:10:)' );
2685 INIT DISKUSE,
2686 CATALOG(TYPEUSE, DISKUSE, CATADDR),
2687 INBUFFER, OUTBUFFER,
2688 CARDBUFFER, PRINTERBUFFER,
2689 INREQUEST, INRESPONSE, OUTREQUEST, OUTRESPONSE,
2690 INSTACK, OUTSTACK, JOBSTACK,
2691 MS, NRCDIRECTORY, FEHINT,
2692 FRMON;
2693 OPERATOR.WRITE ('INIT : (:10:)', 'FRMON(:10:)' );
2694 INIT
2695 FIDT (TYPEUSE);
2696 OPERATOR.WRITE ('INIT : (:10:)', 'FIDT(:10:)' );
2697 INIT
2698 FCONN P(NRCDIRECTORY, MS, TYPEUSE):
2699 OPERATOR.WRITE ( 'INIT : (:10:)',
2700 'FE_CONN_PROCESS(:10:)' );
INIT
FRP(FEHINT, FRMON, TYPEUSE, MS),
FNP(FRMON, FEHINT, FIDT, NRCDIRECTORY,
    TYPEUSE, MS);
OPERATOR.WRITE(´INIT : (:10:)´, ´FNP(:10:)´);
INIT
MRES(FEHINT, FIDT, MS, TYPEUSE),
BCONN(P(NRCDIRECTORY, MS, TYPEUSE);
OPERATOR.WRITE(´INIT : (:10:)´,
    ´BE_CONN_PROCESS(:10:)´);
INIT
BEBINT, BIDT (TYPEUSE);
OPERATOR.WRITE(´INIT : (:10:)´, ´BIDT(:10:)´);
INIT
MRQSTP(BEBINT, BIDT, MS, TYPEUSE),
BNP(BIDT, NRCDIRECTORY, TYPEUSE, MS);
OPERATOR.WRITE(´INIT : (:10:)´, ´BNP(:10:)´);
INIT
CDP(BEBINT, MS, TYPEUSE, PRINTERBUFFER),
READER(TYPEUSE, CARDBUFFER),
WRITER(TYPEUSE, PRINTERBUFFER),
PRODUCER(TYPEUSE, DISKUSE, CATALOG, CARDBUFFER,
     INBUFFER, INREQUEST, INRESPONSE,
     INSTACK, INPUTTASK),
CONSUMER(TYPEUSE, DISKUSE, CATALOG, PRINTERBUFFER,
     OUTBUFFER, OUTREQUEST, OUTRESPONSE,
     OUTSTACK, OUTPUTTASK),
MASTER(TYPEUSE, DISKUSE, CATALOG,
     INBUFFER, OUTBUFFER,
     INREQUEST, INRESPONSE,
     OUTREQUEST, OUTRESPONSE,
     JOBSTACK, FEHINT),
WATCHDOG(DISKUSE);
OPERATOR.WRITE(´INIT : (:10:)´,
    ´WATCHDOG(:10:)´);
END.
CONCURRENT PROGRAMMING OF THE USER ENVELOPE IN
A DISTRIBUTED DATA BASE MANAGEMENT SYSTEM

by

MICHAEL WAYNE FARRELL

B.S., Kansas State University, Manhattan, Kansas, 1975

__________________________
AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1979
The purpose of this project was to decentralize the control component of a distributed data processing system. A previously developed prototype was enhanced by structuring the user envelope to concurrently process data base requests. A network resource controller and a directory of network names was added to the user envelope. A host machine task, on the Interdata 8/32, supported application program requests, through the MIMICS interprocessor communication software, to a backend machine task on the Interdata 7/32 that accessed a data base.

The results of the project demonstrated a high programmer productivity rate, a framework that demonstrates a modular, structured approach to the design of a distributed data processing system, a prototype that supports a network naive application programmer that requires access to data maintained in a distributed data processing system, and a prototype that included the implementation of a network resource controller which has the intelligence to dynamically designate the machine as a host, backend, or bi-functional machine.