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

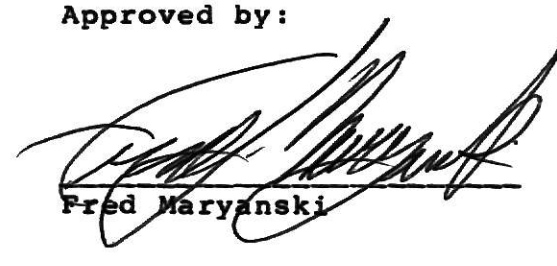
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1979

Approved by:



Fred Maryanski

TABLE OF CONTENTS

FIGURES

TABLE

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 descreases 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.

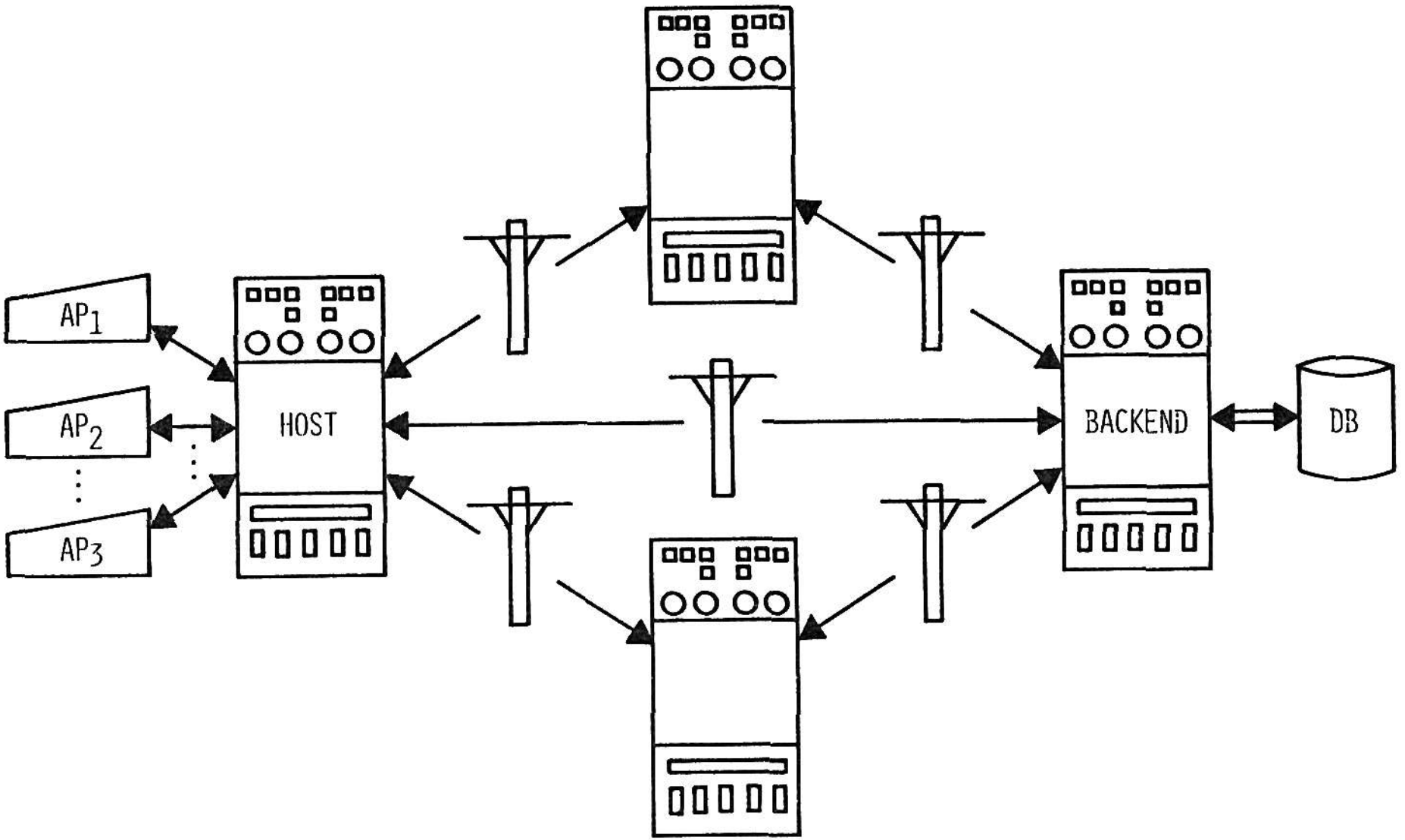


FIGURE 1

A FULLY DISTRIBUTED DATA PROCESSING NETWORK

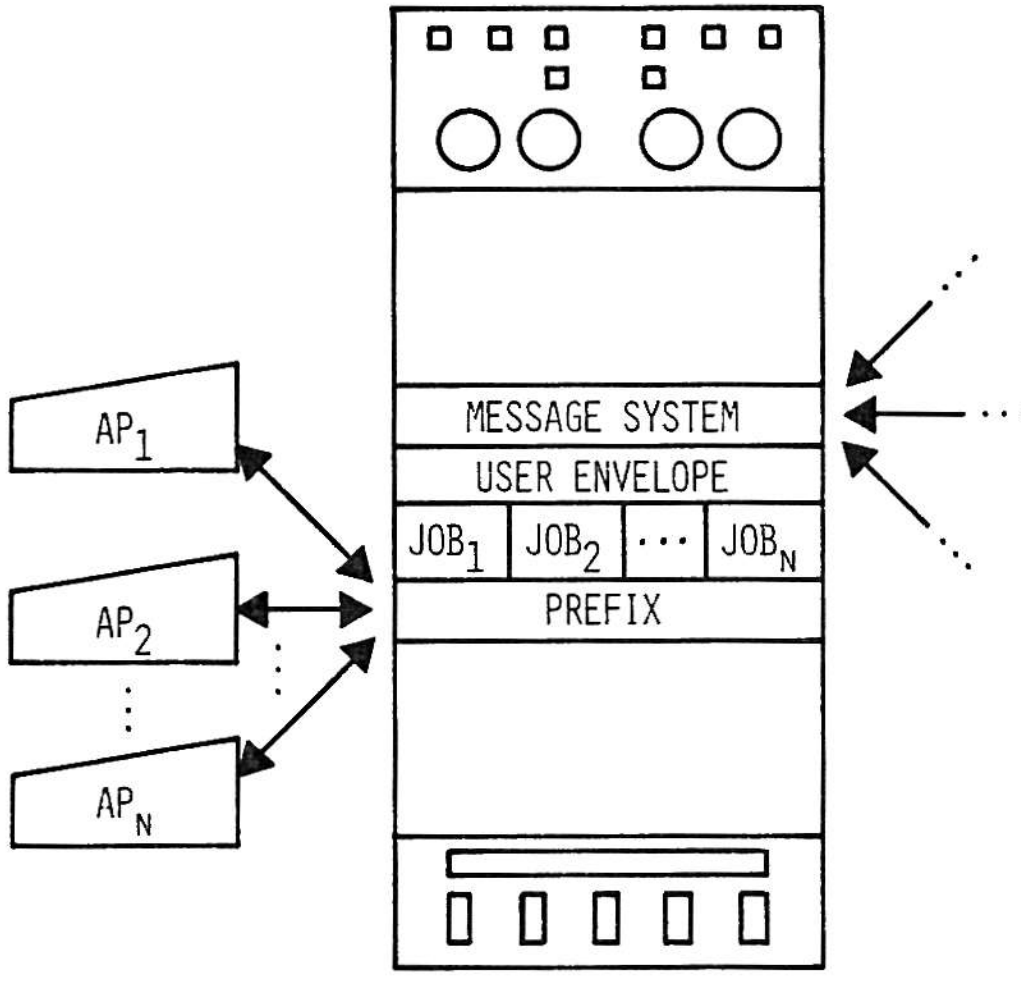


FIGURE 2

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

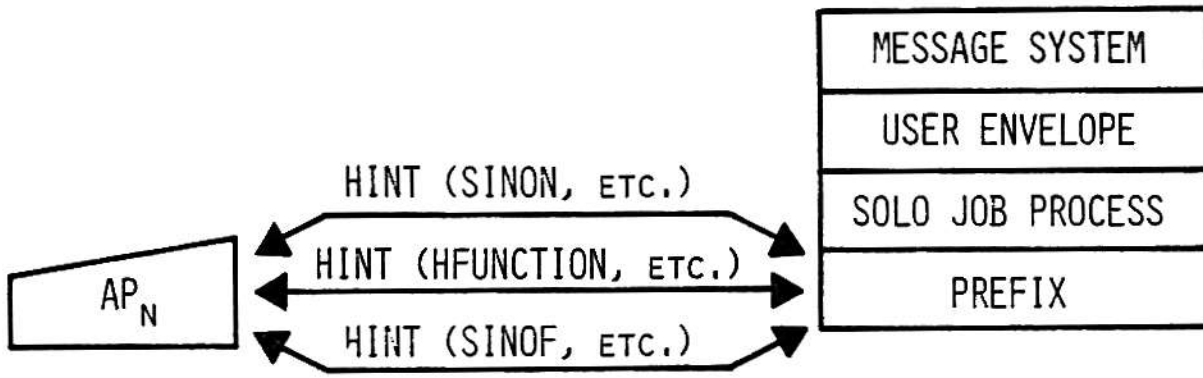


FIGURE 3

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 SINOF.

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 visable 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.

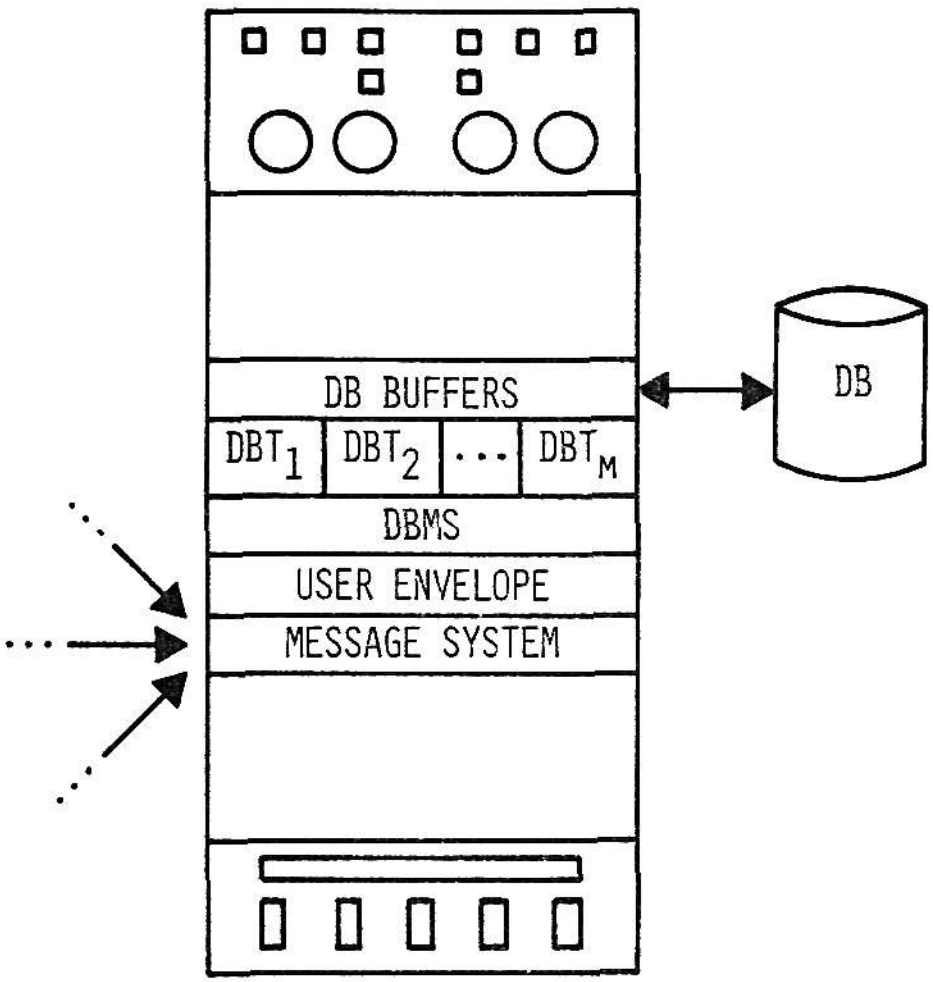


FIGURE 4

FUNCTIONAL COMPONENTS
OF A BACKEND MACHINE

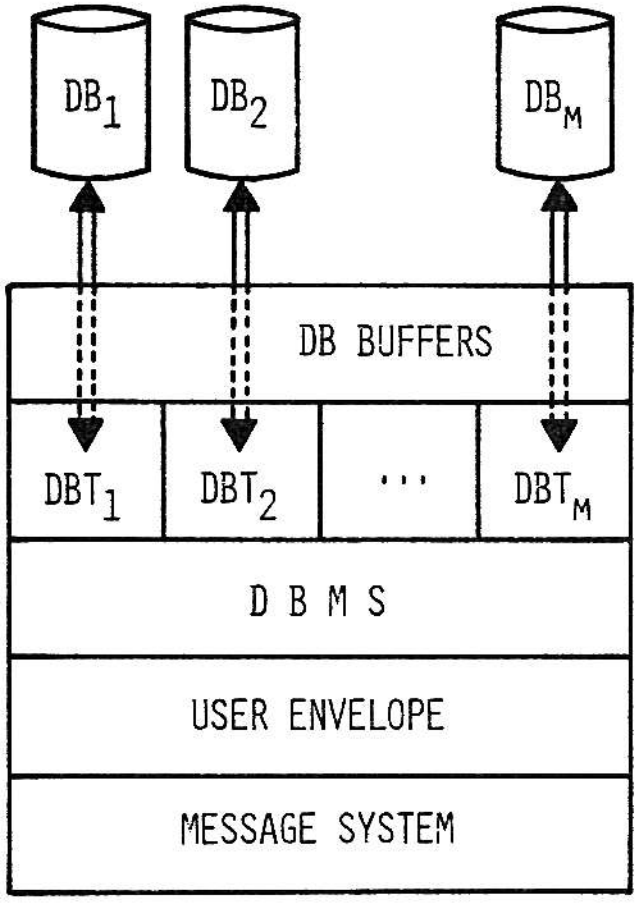


FIGURE 5

A BACKEND MACHINE WITH A SINGLE THREADED DATA BASE MANAGEMENT SYSTEM

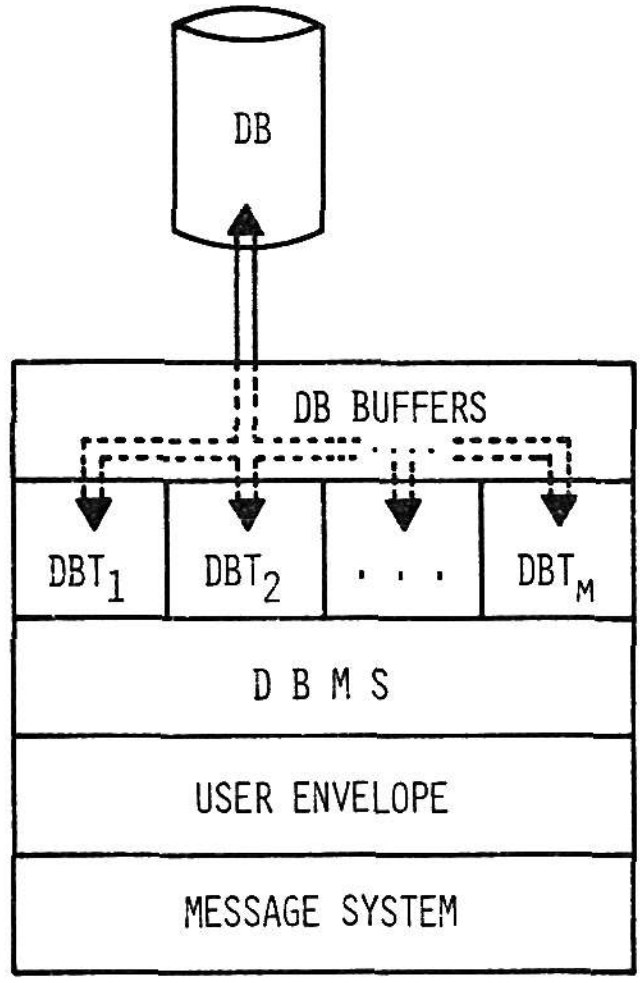


FIGURE 6

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,

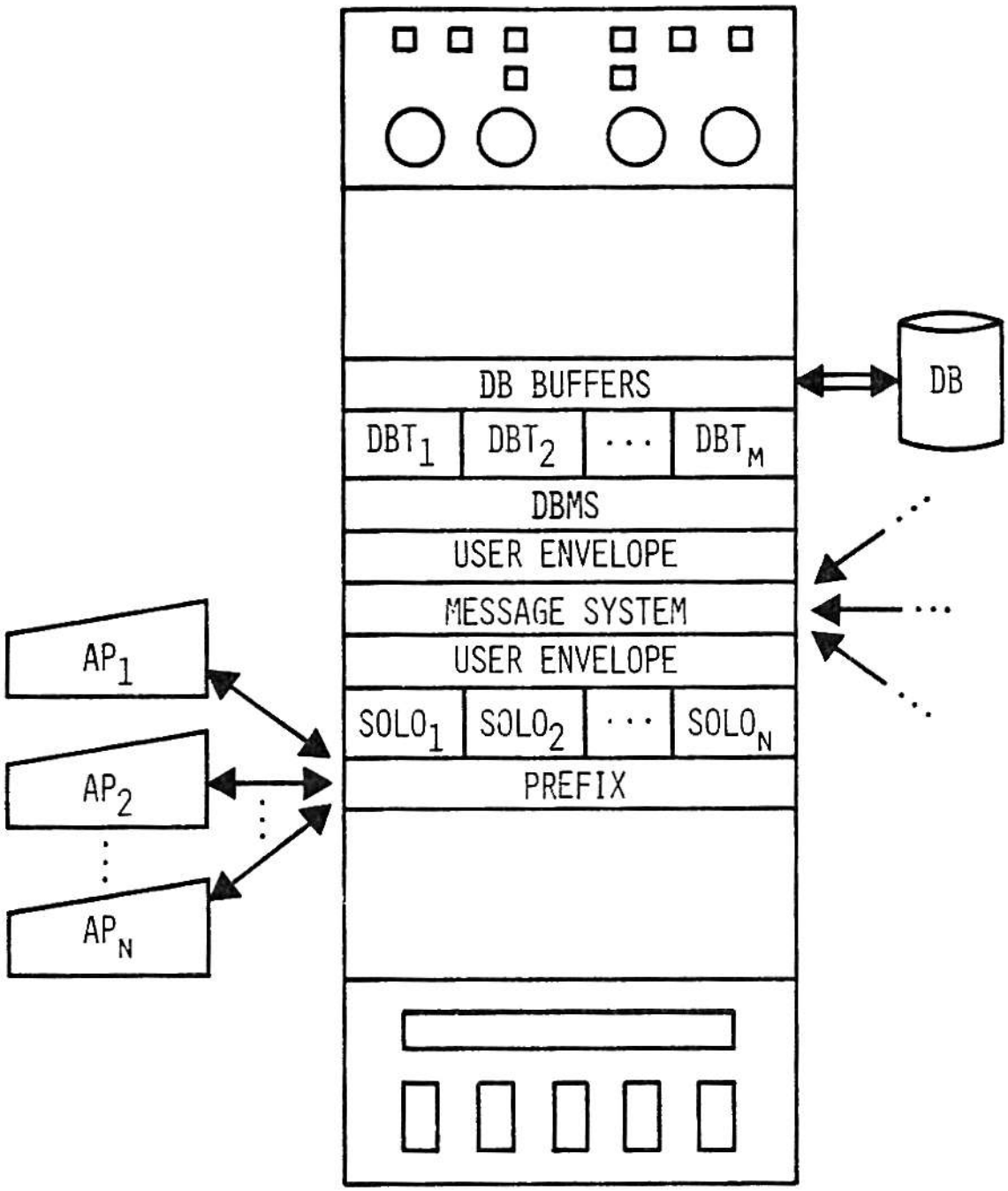


FIGURE 7

FUNCTIONAL COMPONENTS
OF A BI-FUNCTIONAL MACHINE

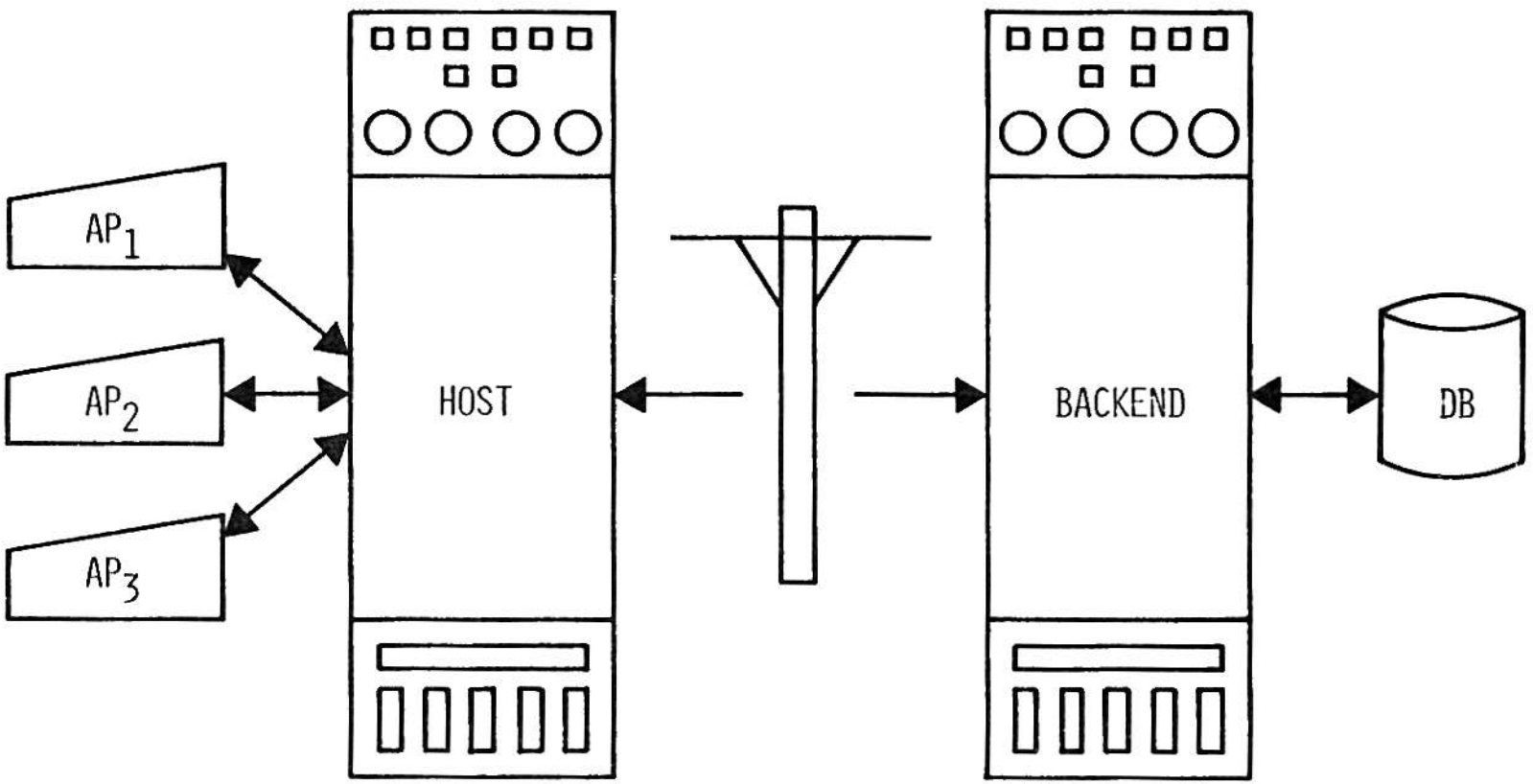


FIGURE 8

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_CONNect (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_CONNect (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

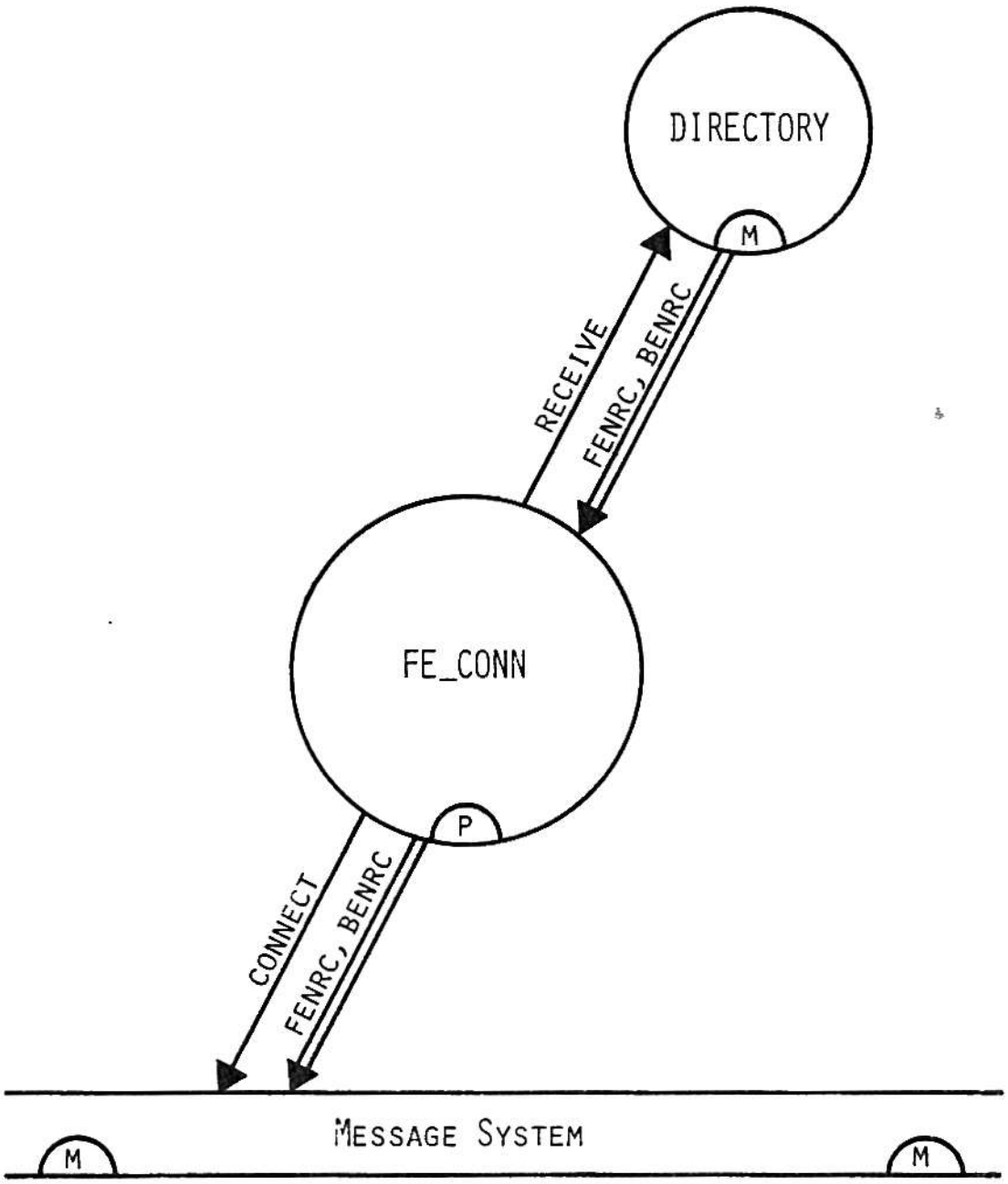


FIGURE 9

FRONTEND_CONNECT PROCESS

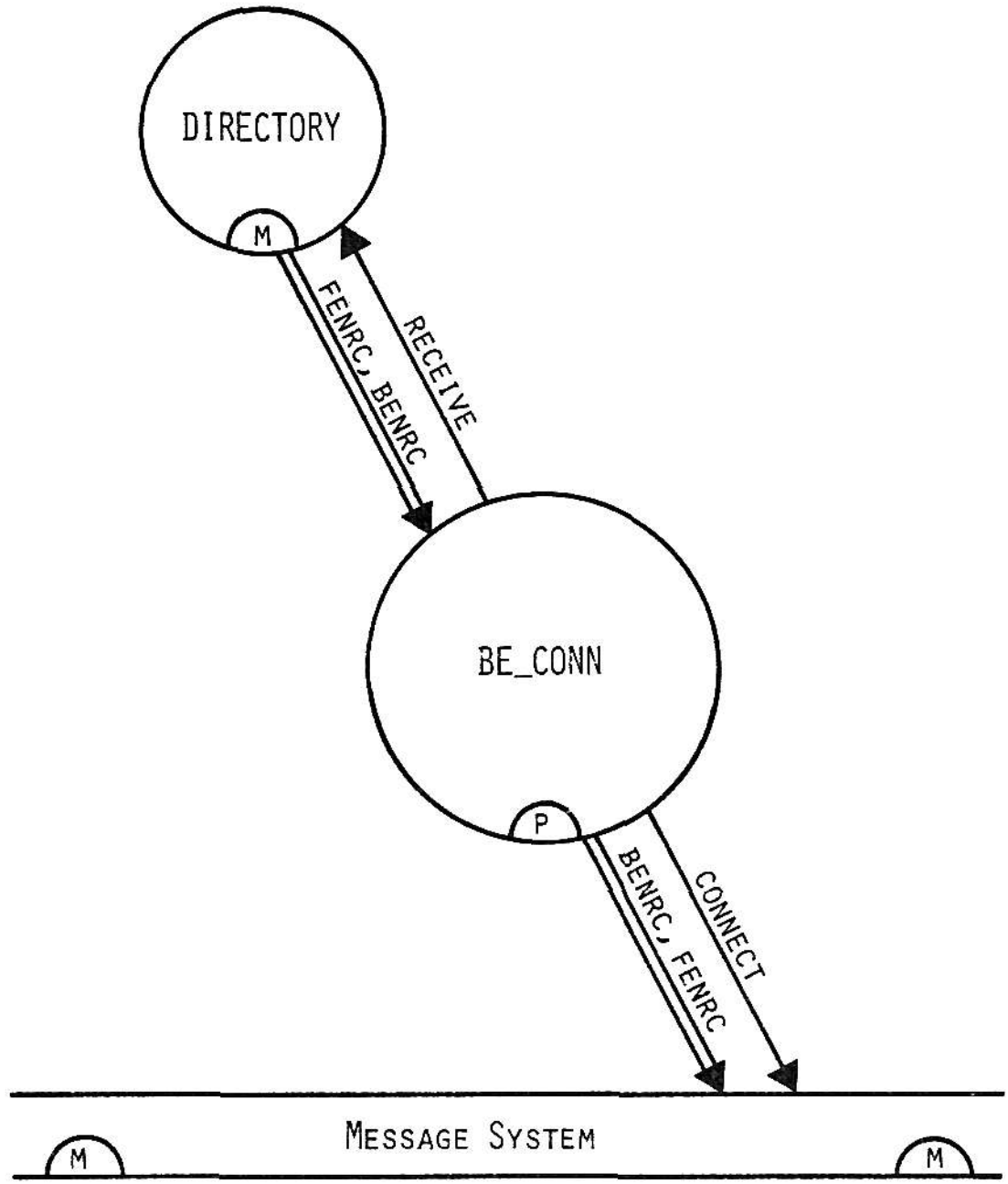


FIGURE 10

BACKEND_CONNECT PROCESS

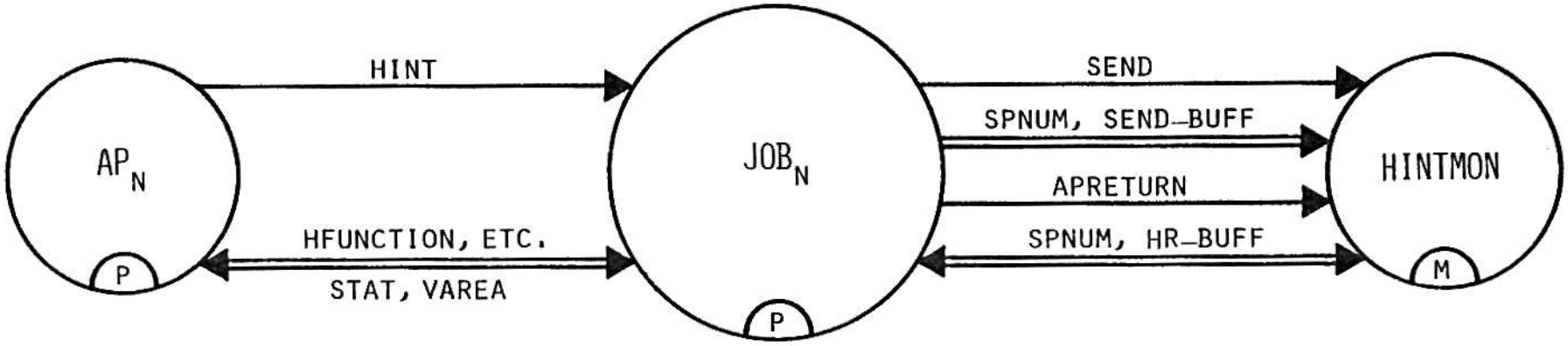


FIGURE 11

ACCESS TO THE USER ENVELOPE
BY THE JOB PROCESS

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

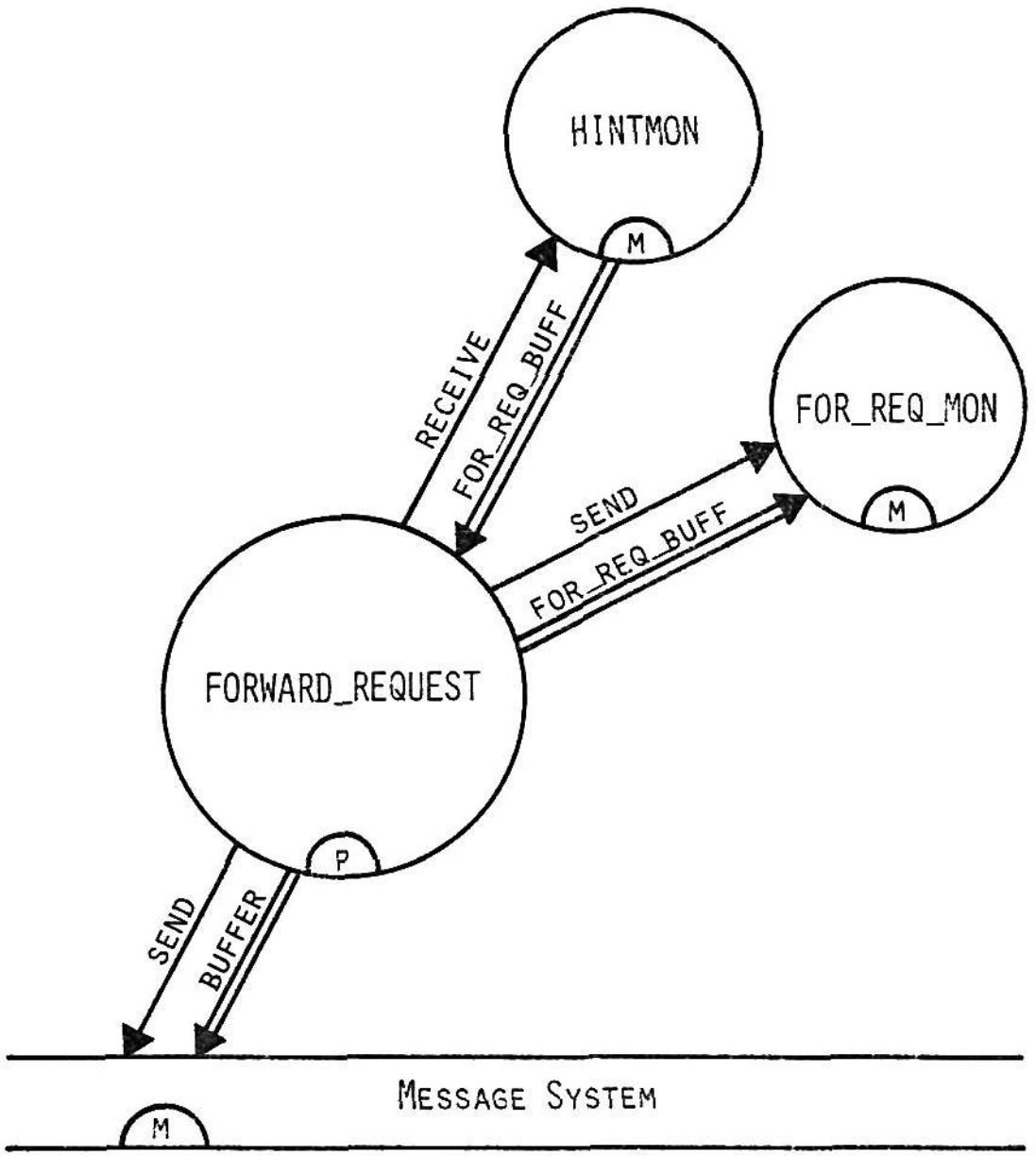


FIGURE 12
FORWARD_REQUEST PROCESS

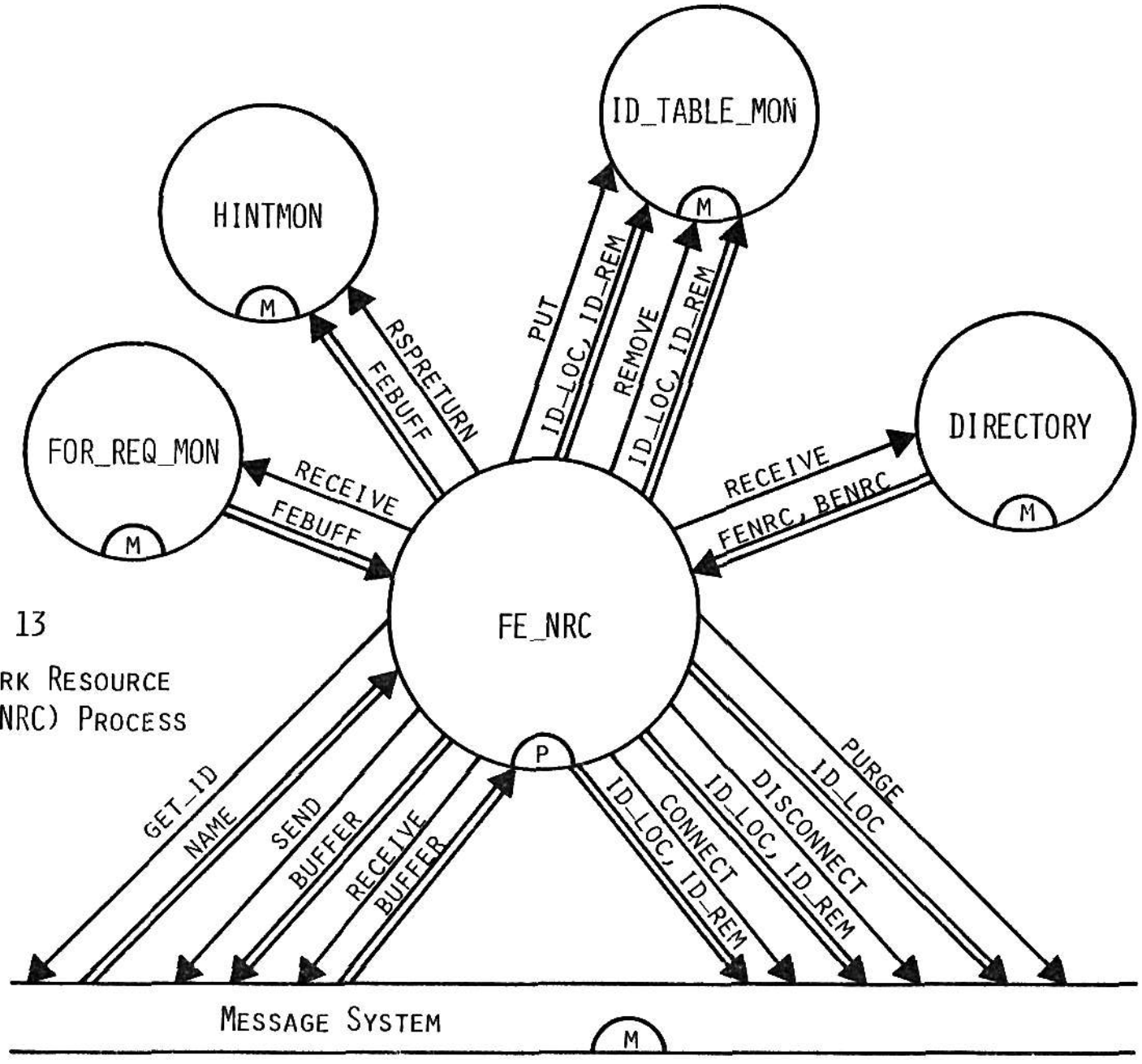


FIGURE 13

FRONTEND_NETWORK RESOURCE CONTROLLER (FE_NRC) PROCESS

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 Resouce 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.

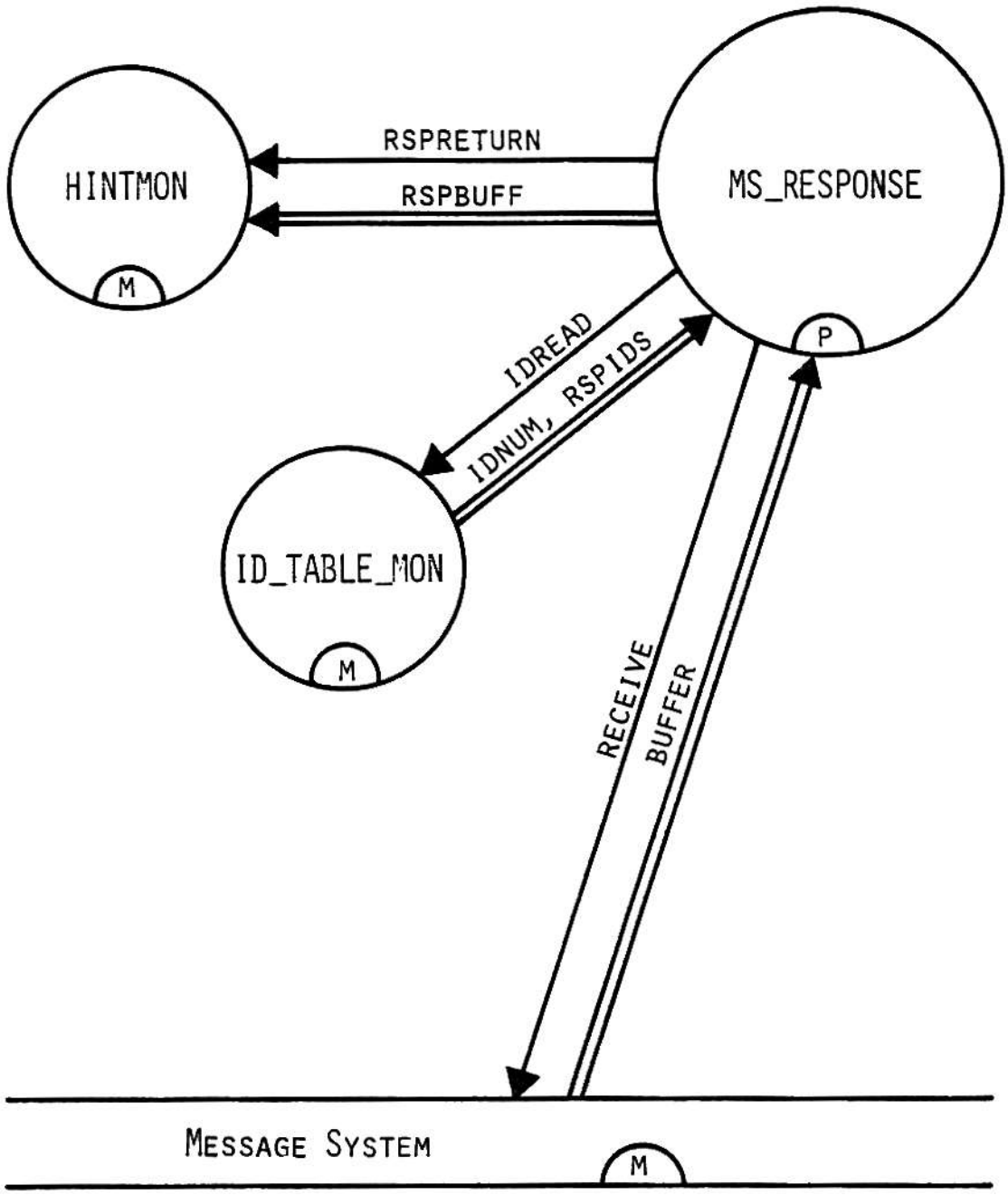


FIGURE 14

MESSAGE SYSTEM_RESPONSE PROCESS

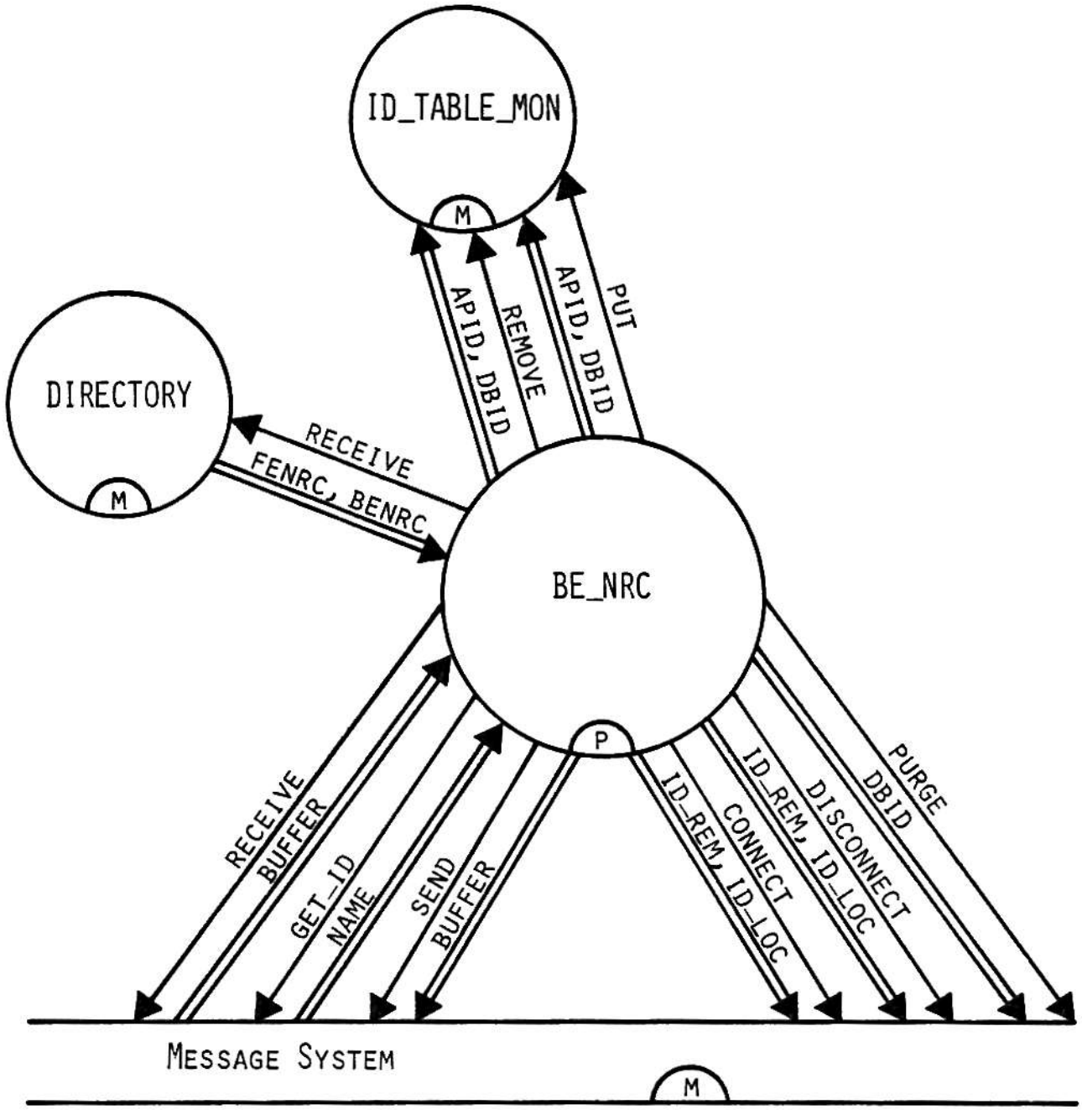


FIGURE 15

BACKEND_NETWORK RESOURCE
CONTROLLER (BE_NRC) PROCESS

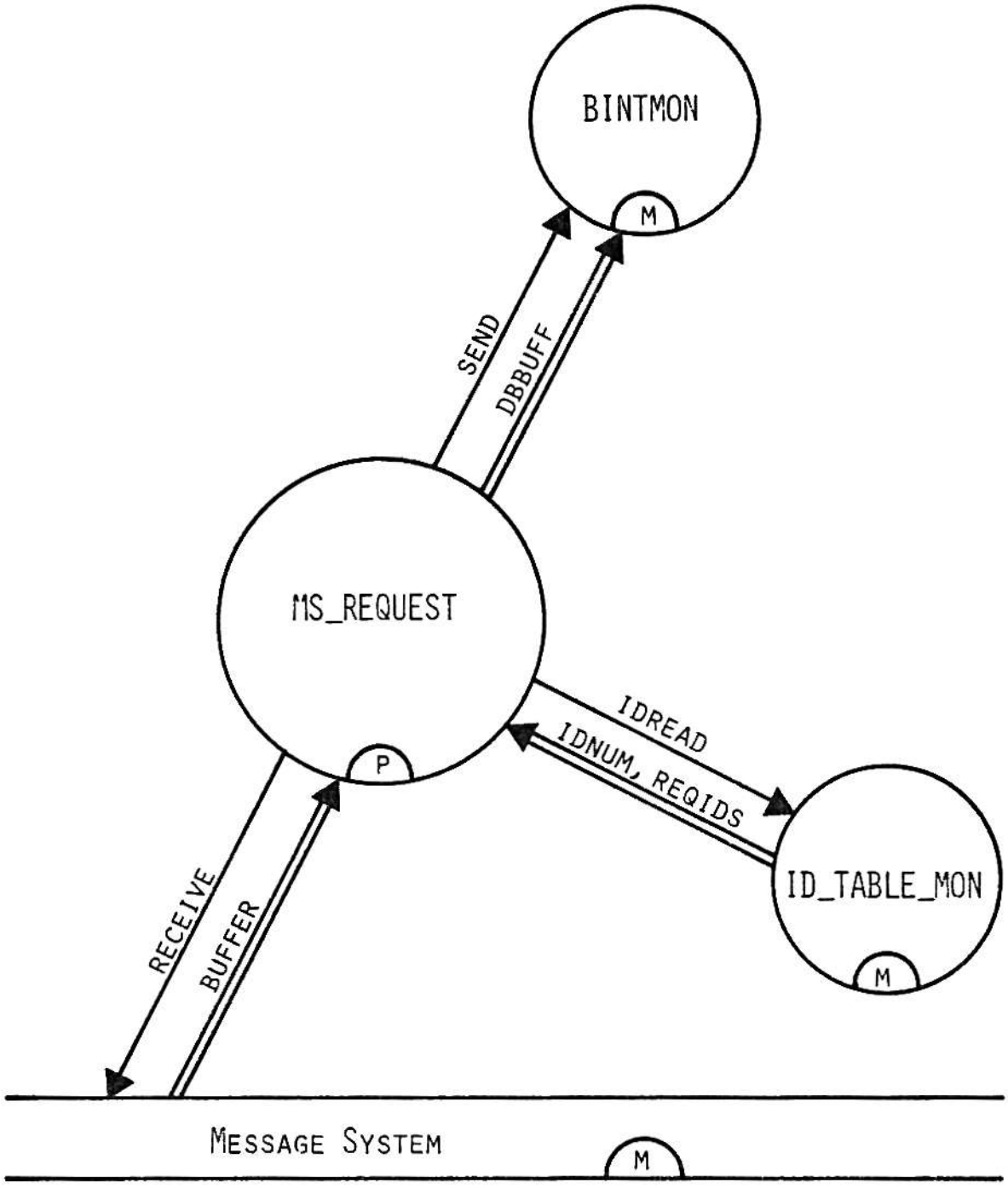


FIGURE 16

MESSAGE SYSTEM_REQUEST PROCESS

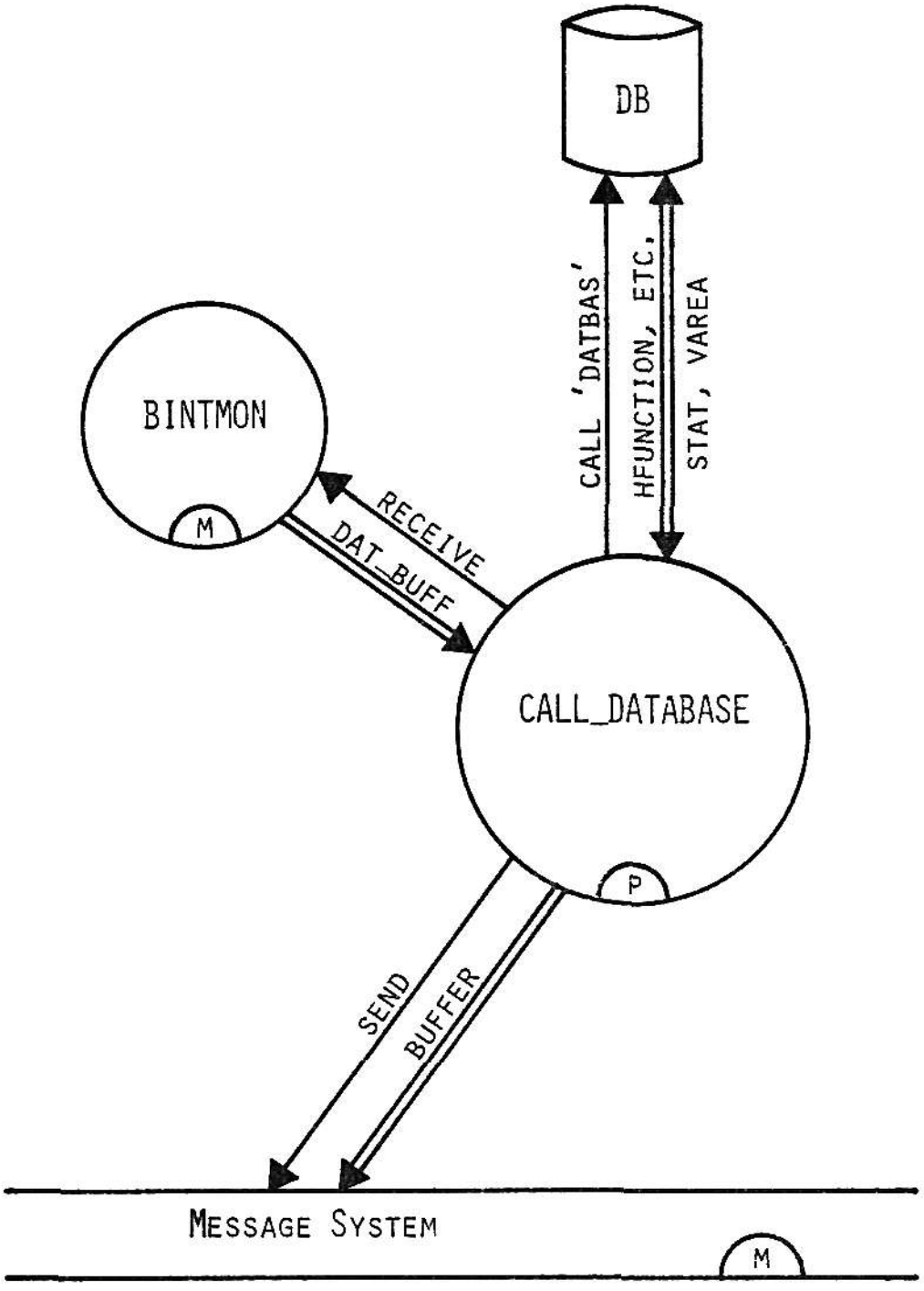


FIGURE 17

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.

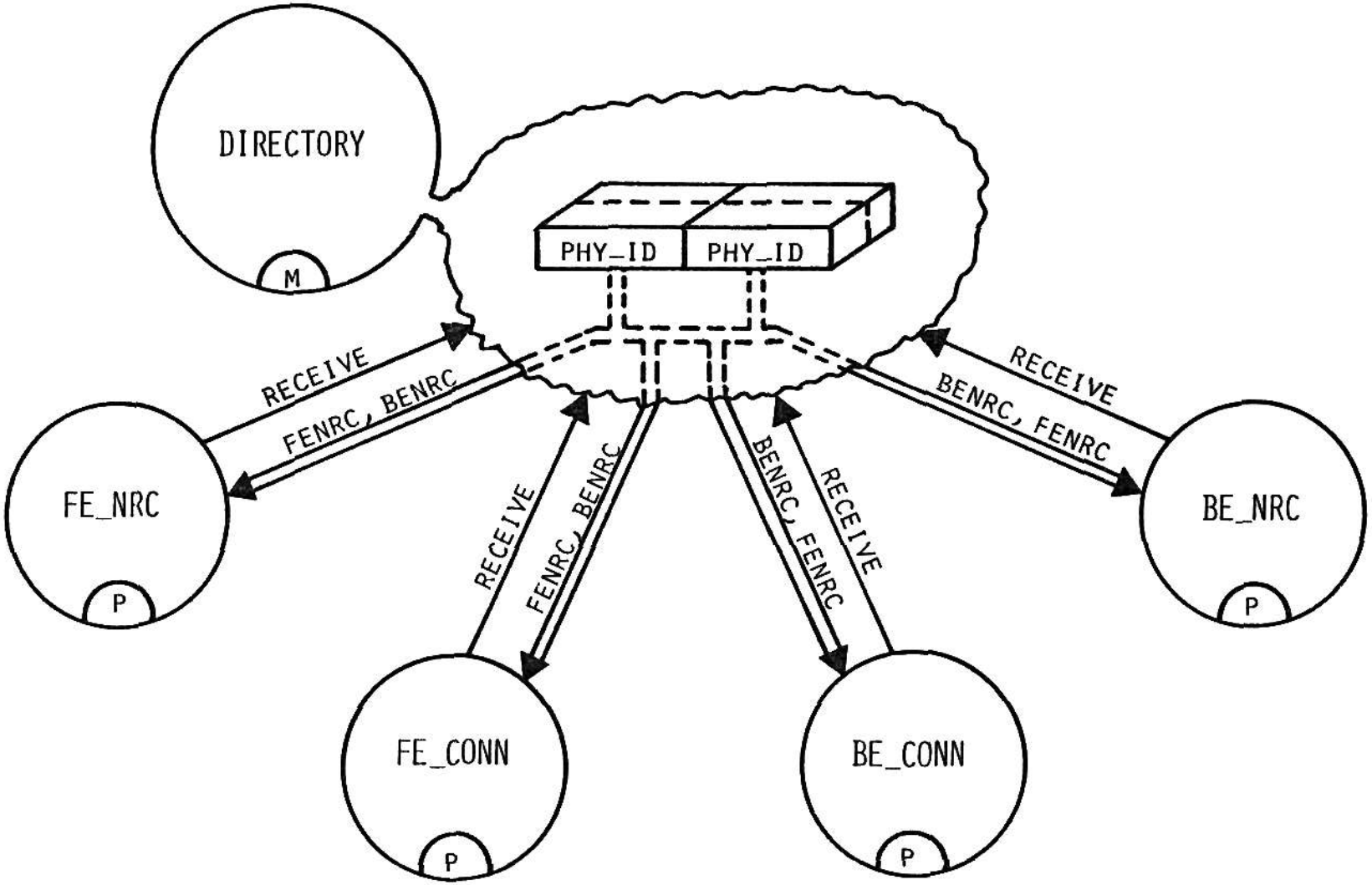


FIGURE 18

DIRECTORY MONITOR

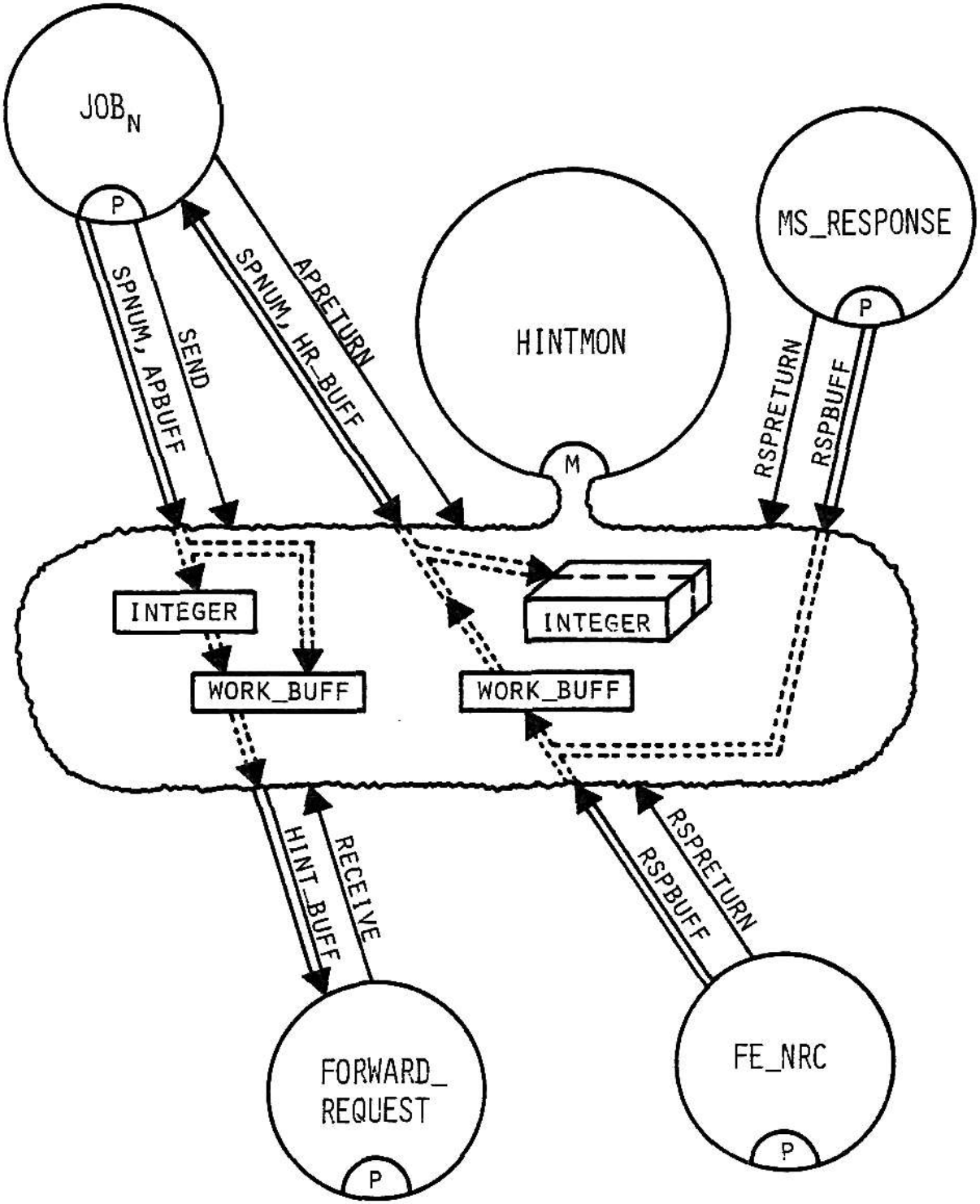


FIGURE 19

HINTMON MONITOR

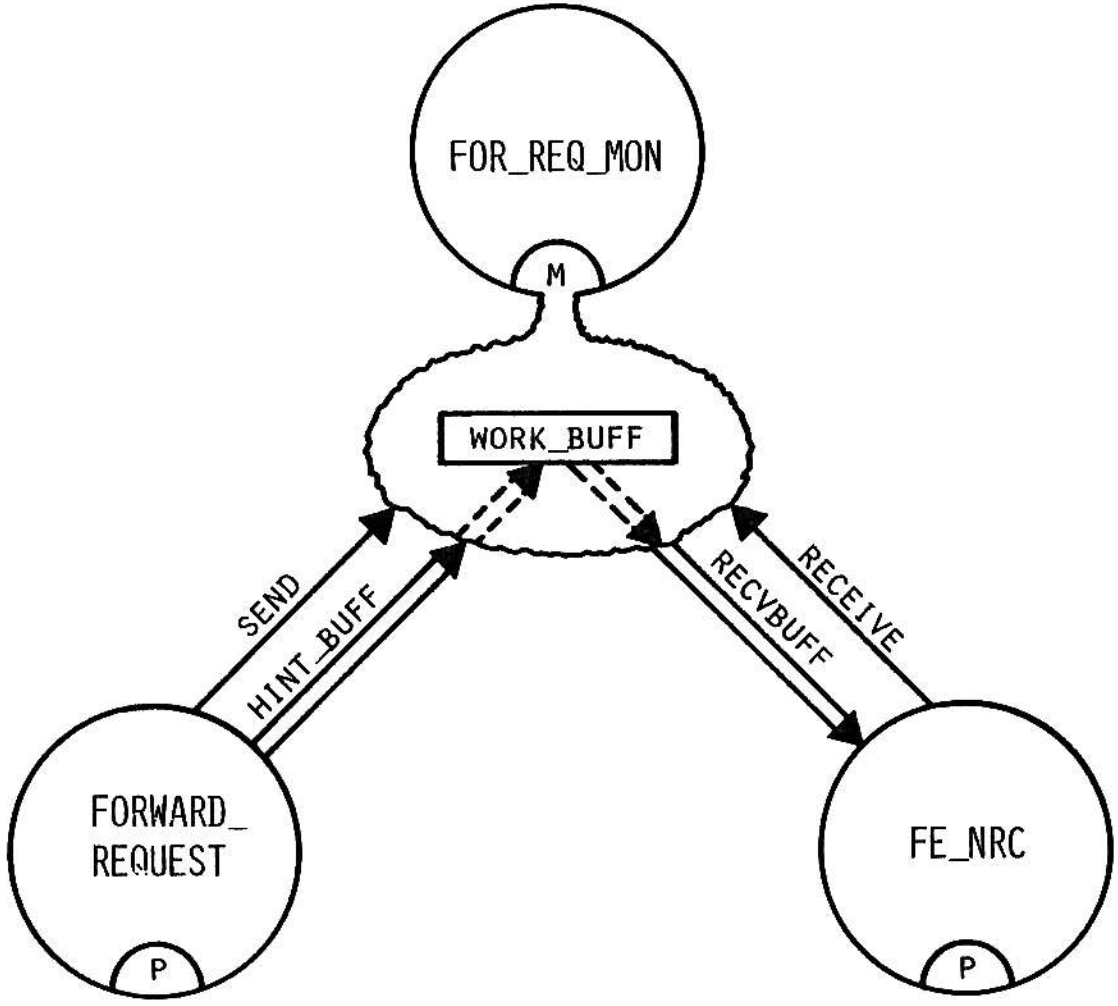


FIGURE 20

FOR_REQ_MON MONITOR

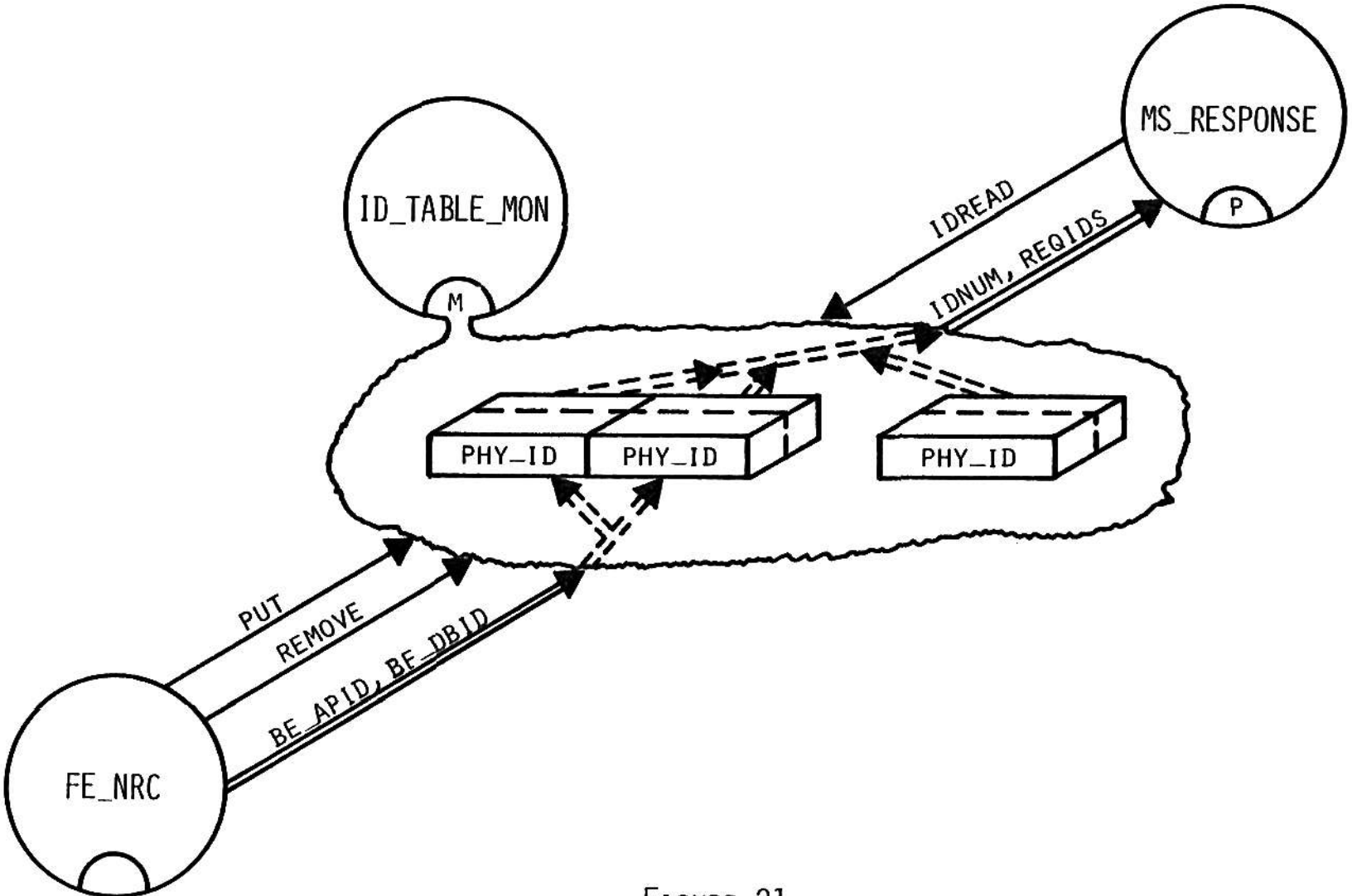


FIGURE 21

ID_TABLE_MON MONITOR

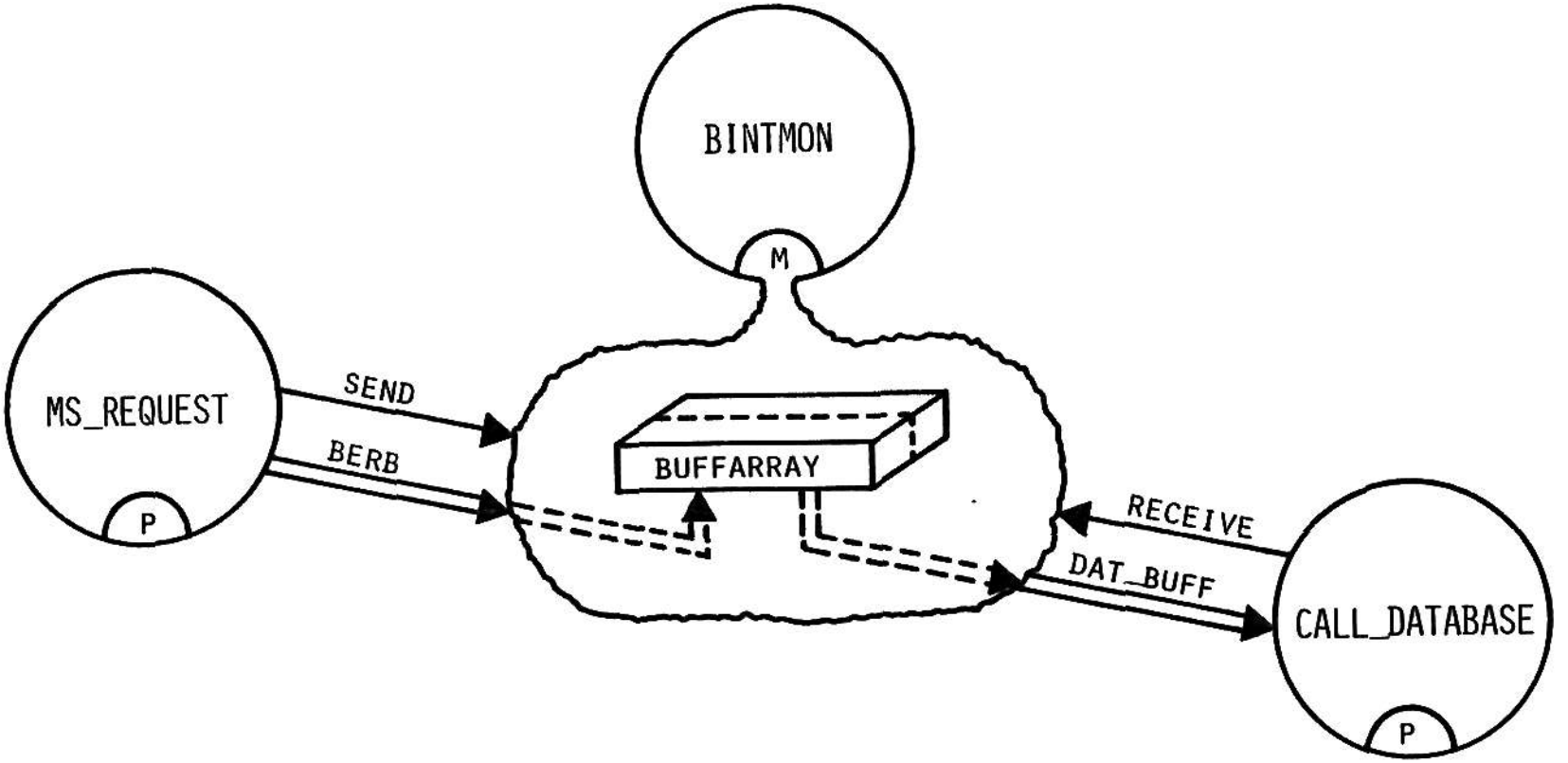


FIGURE 22

BINTMON 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.

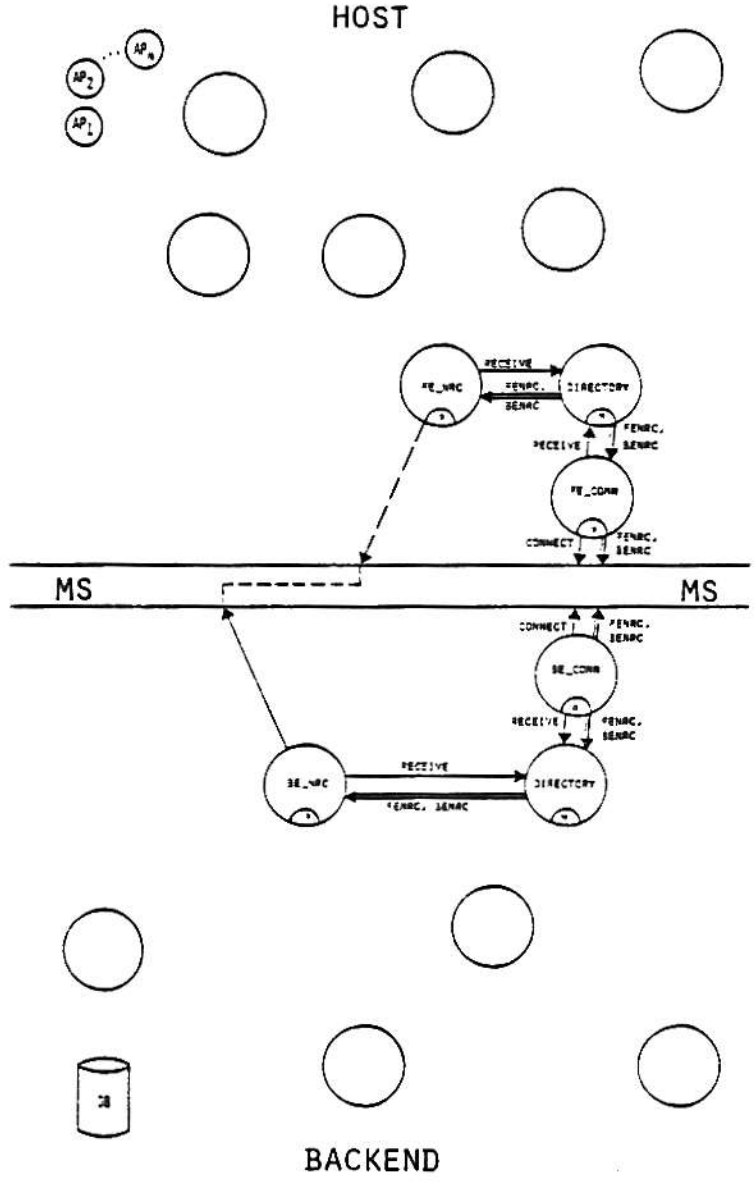


FIGURE 23

NRCs AT SYSTEM INITIALIZATION

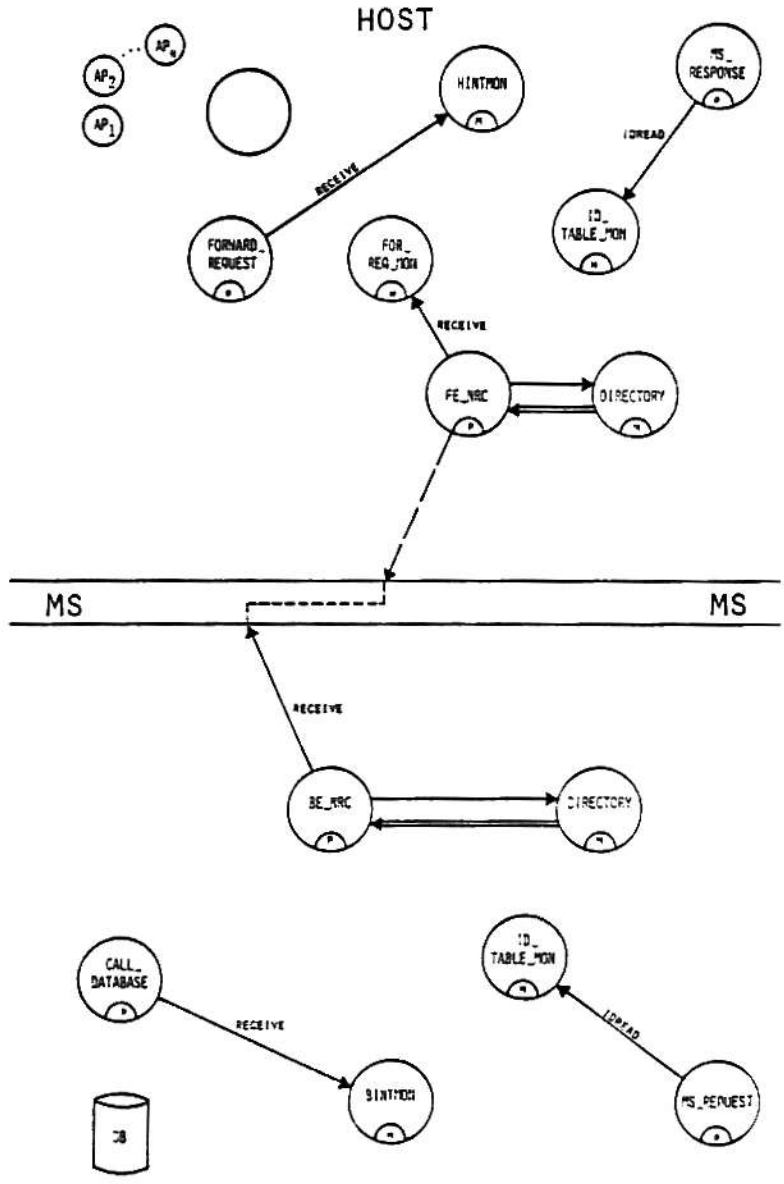


FIGURE 24

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 RECEIV(E)ing 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.

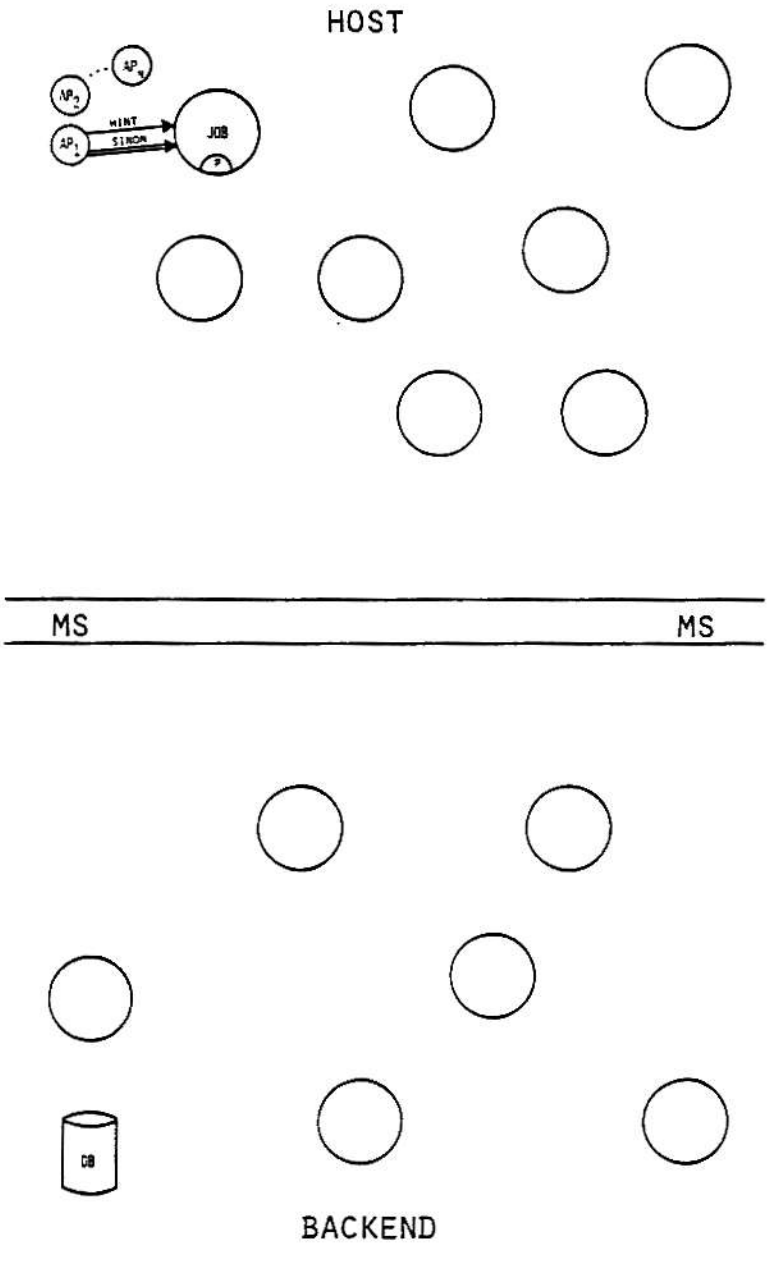


FIGURE 25

APPLICATION PROGRAM REQUEST

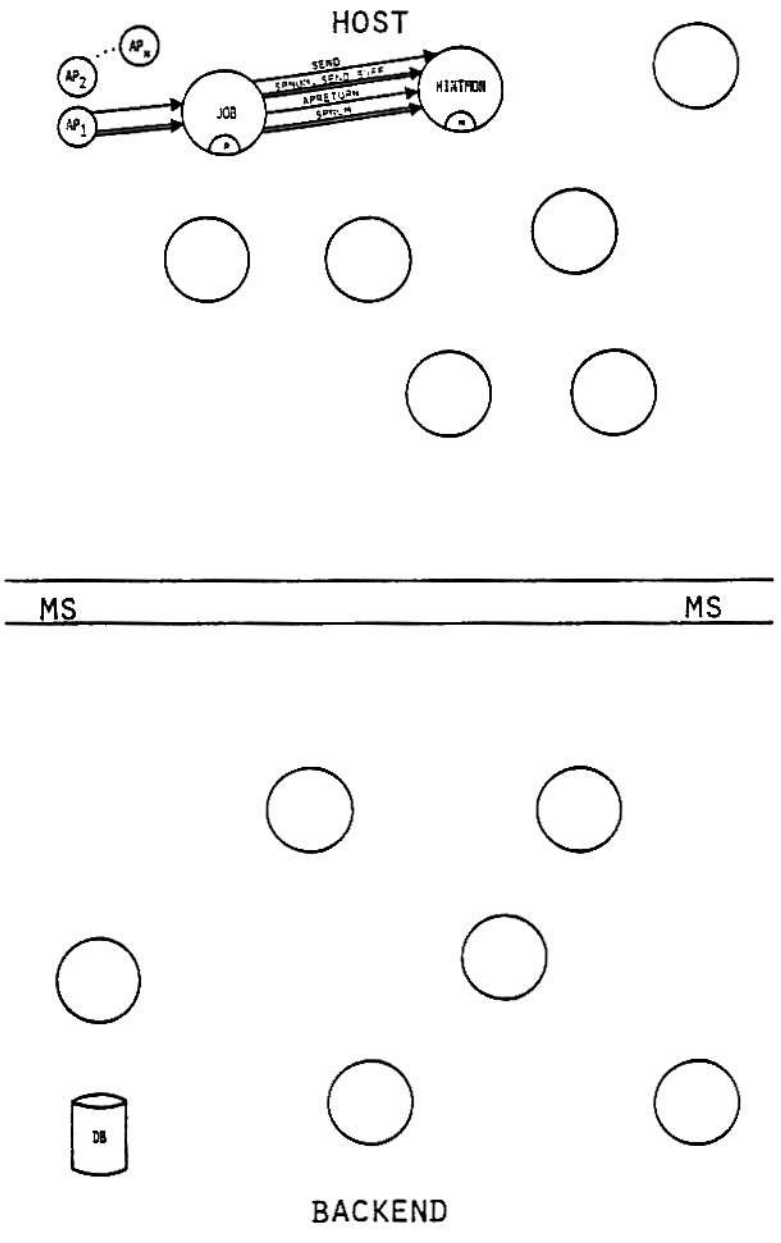


FIGURE 26
JOB PROCESS SENDING VARIABLES

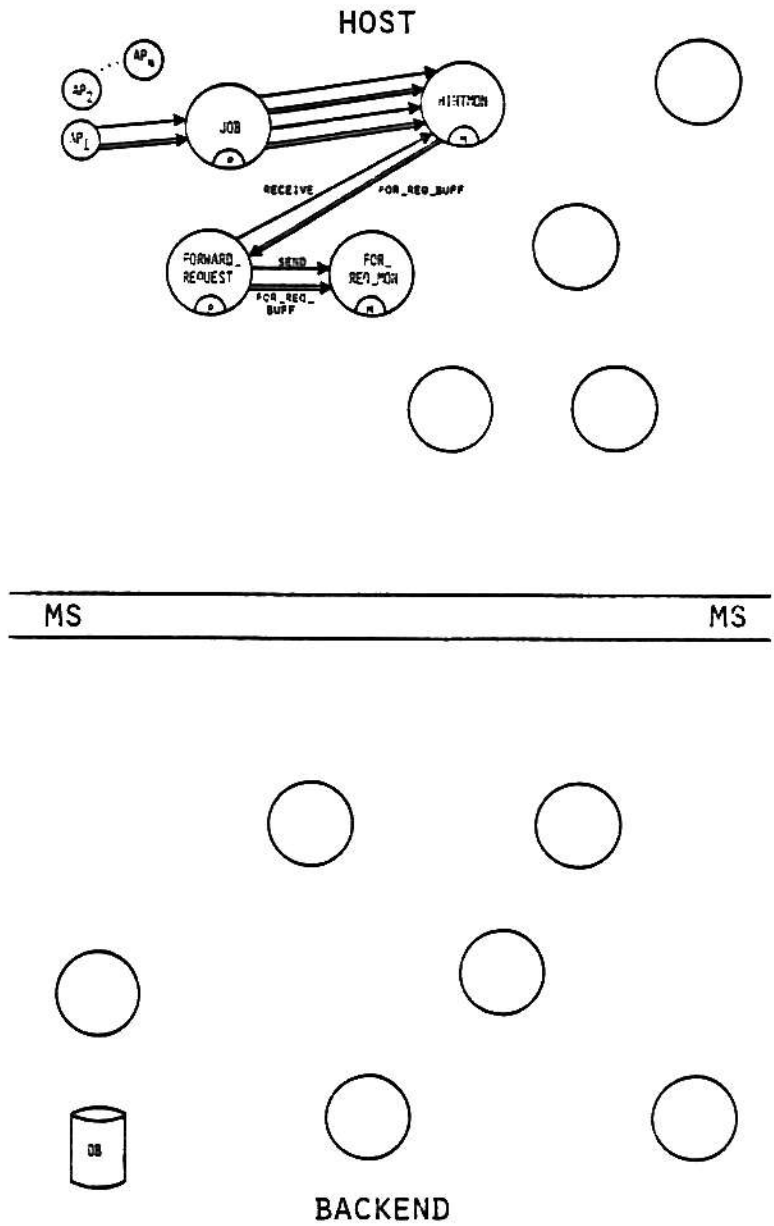


FIGURE 27
FORWARD_REQUEST PROCESS

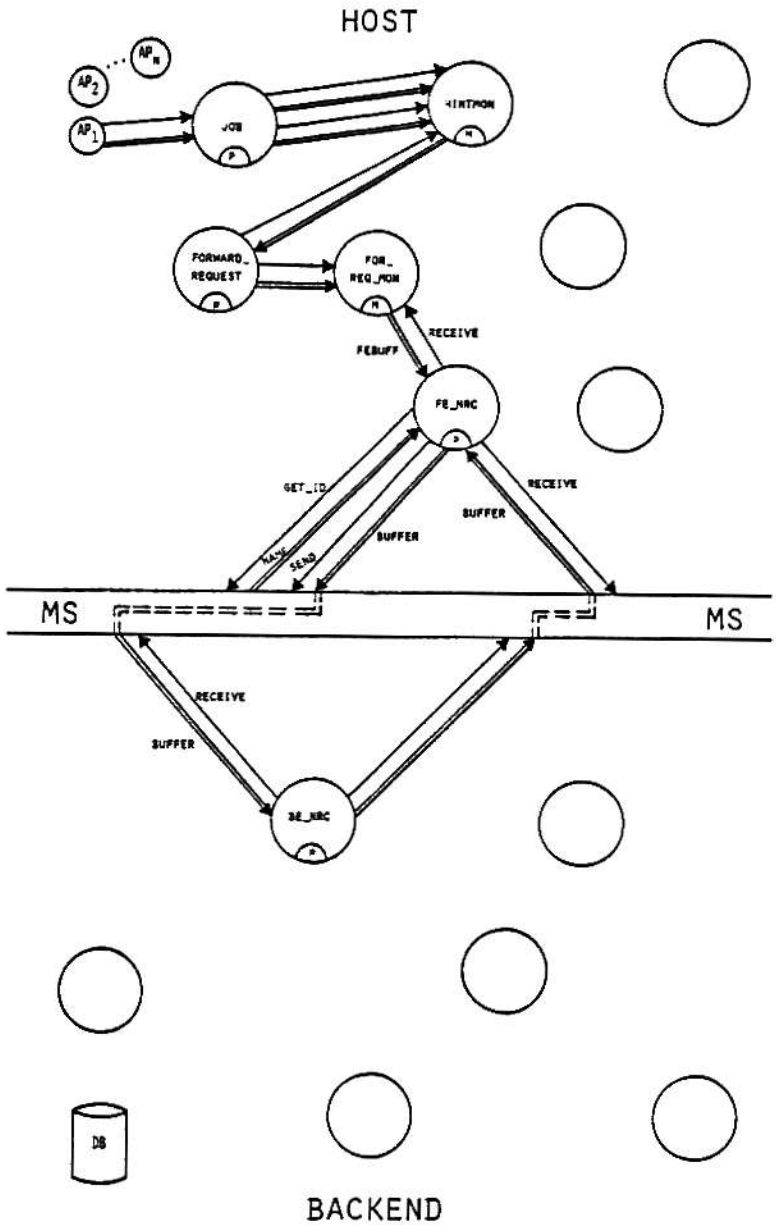


FIGURE 28
FE_NRC PROCESS

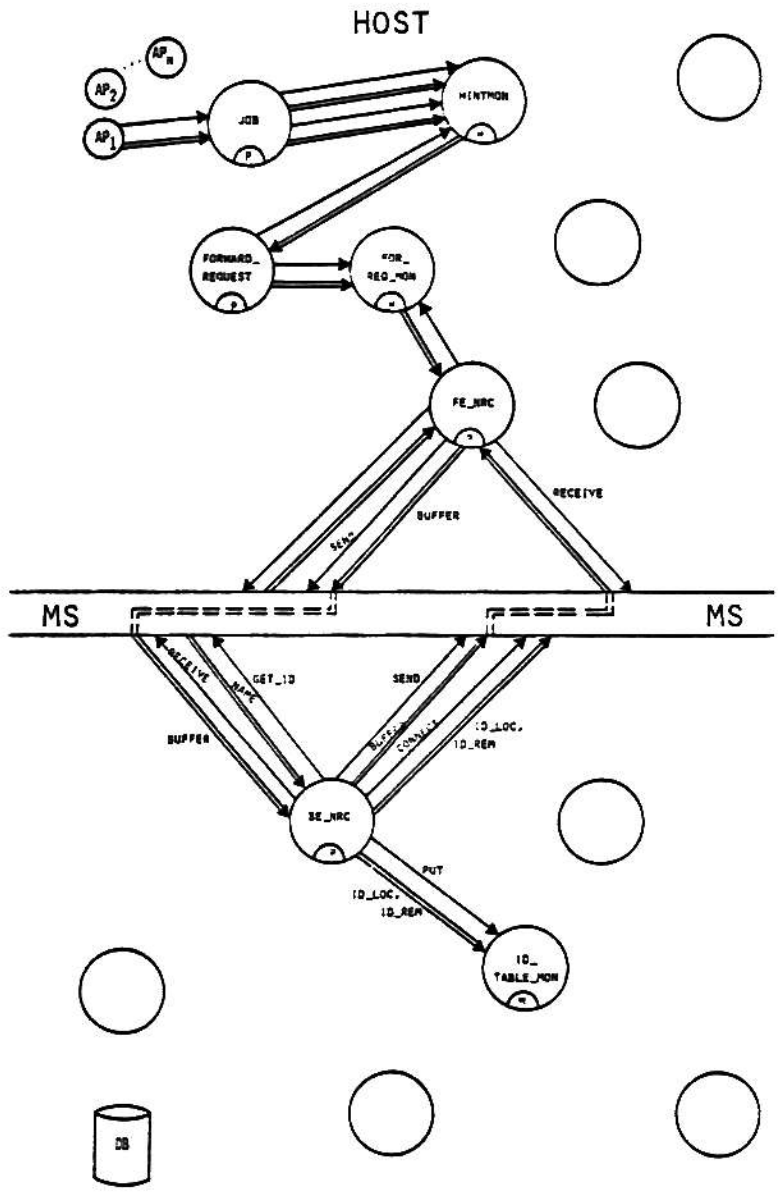


FIGURE 29

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_REQUET 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.

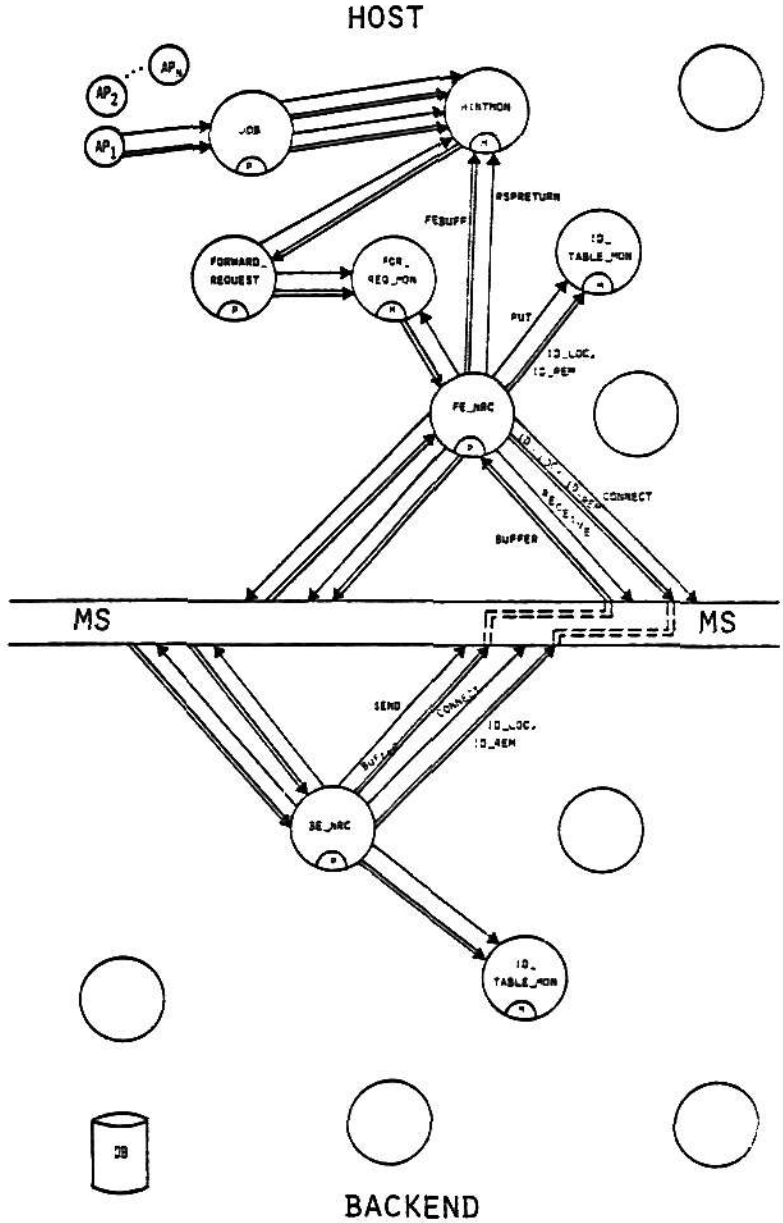


FIGURE 30

FE_NRC RETURNING TO HINTMON

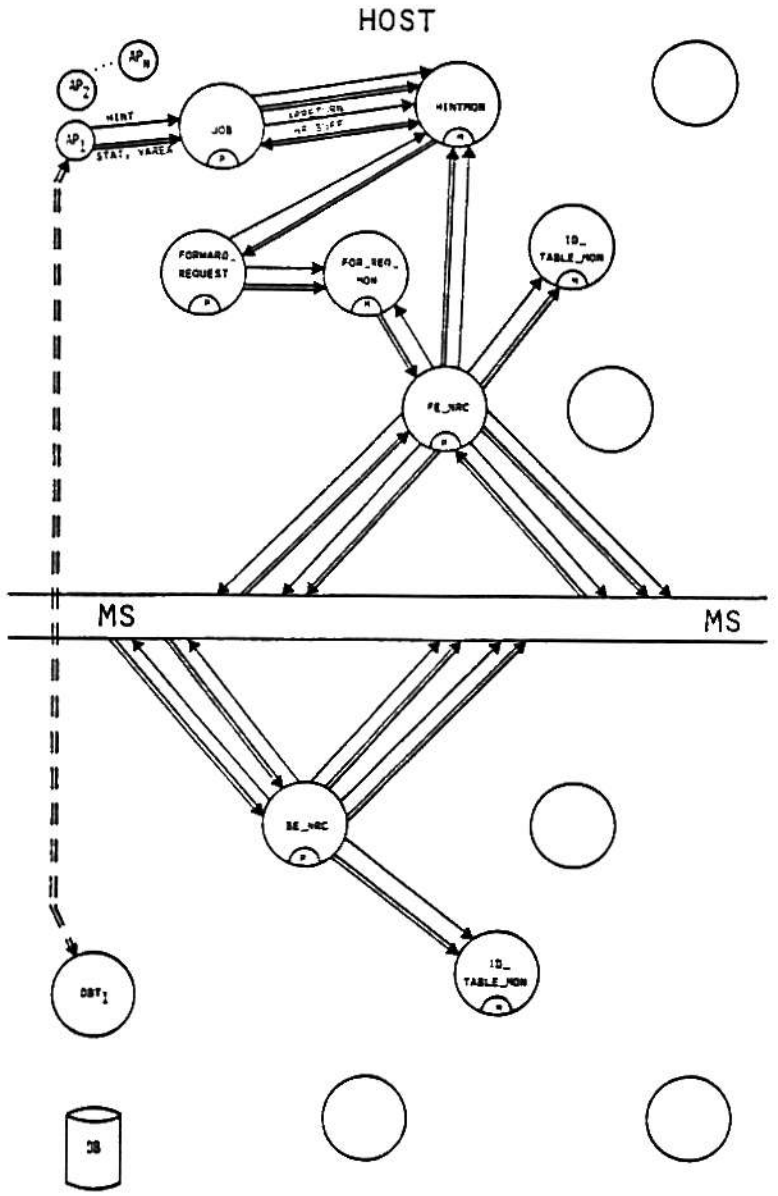


FIGURE 31

JOB PROCESS RETURNING

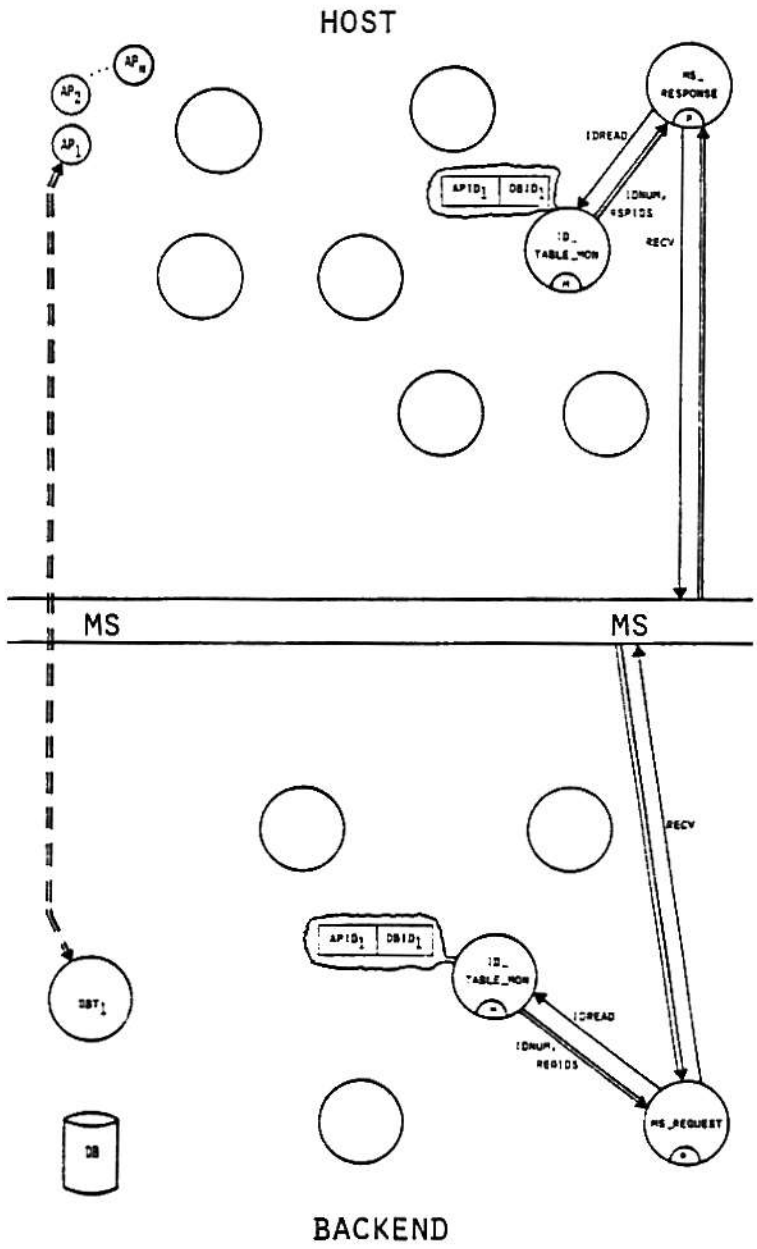


FIGURE 32

USER ENVELOPE AFTER SINON

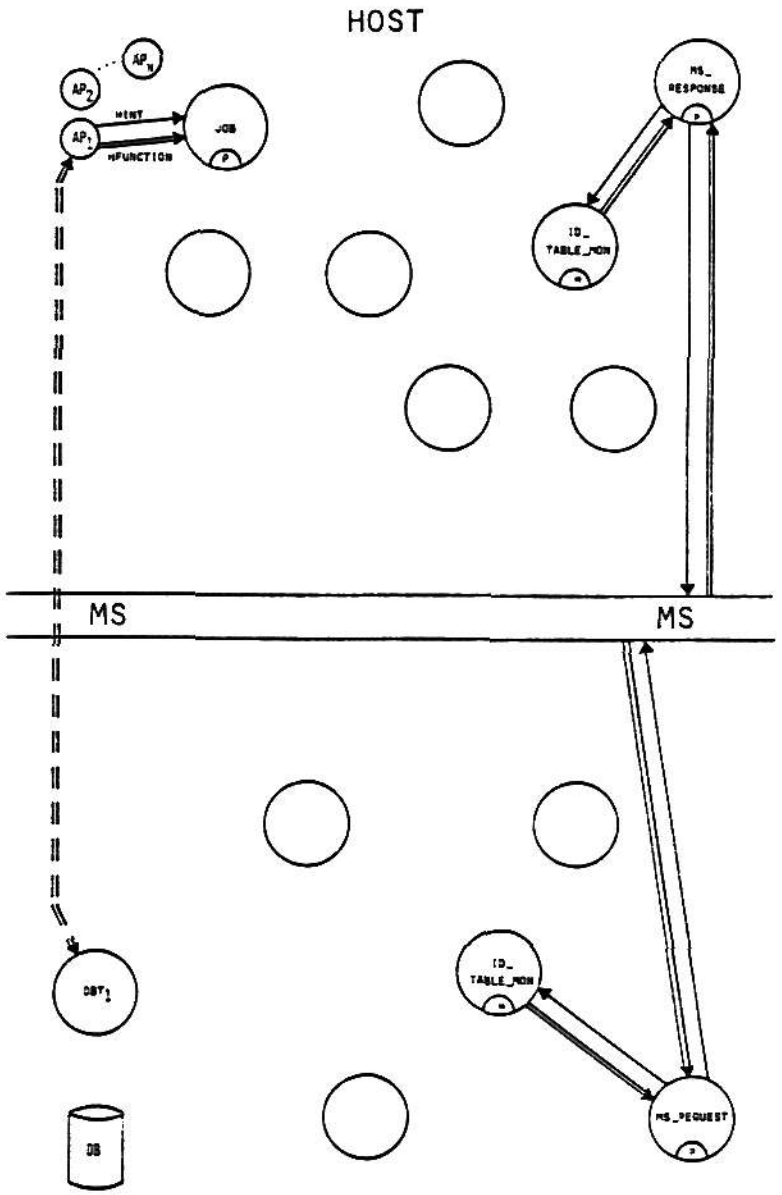


FIGURE 33

A DATA BASE REQUEST

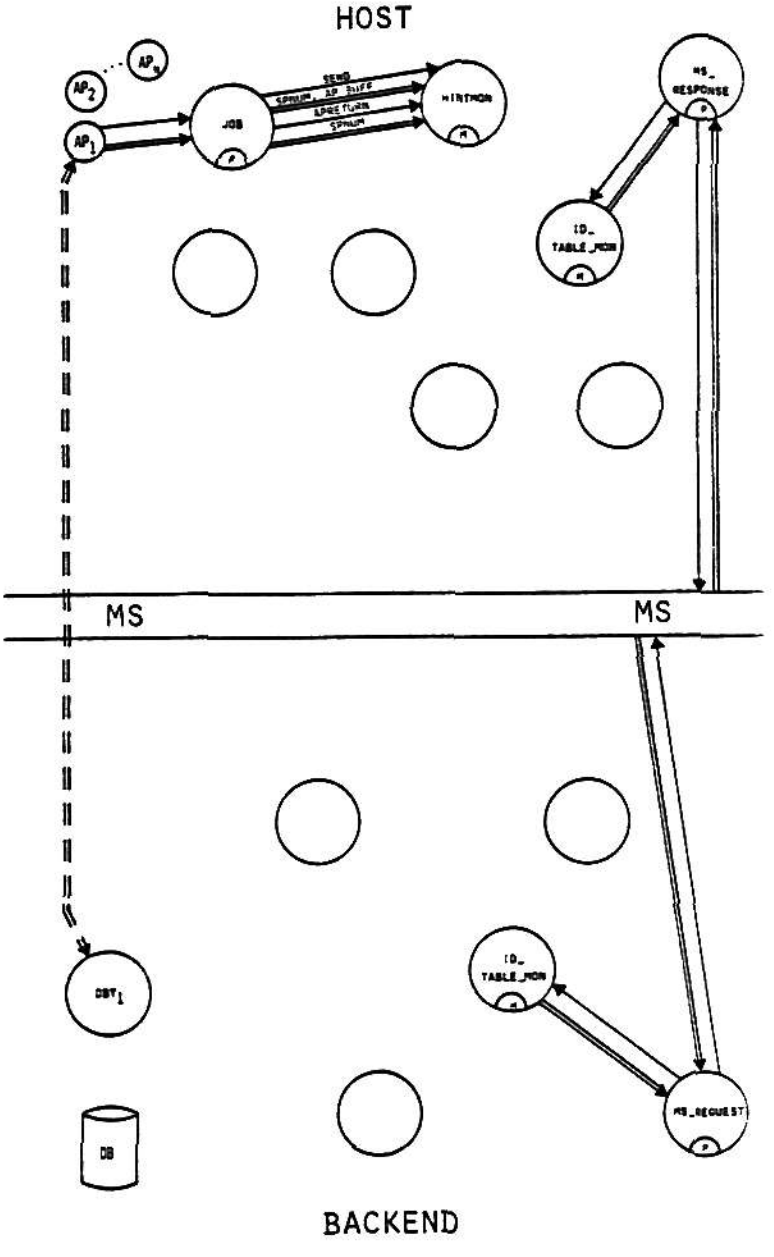


FIGURE 34
JOB SENDs TO HINTMON

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

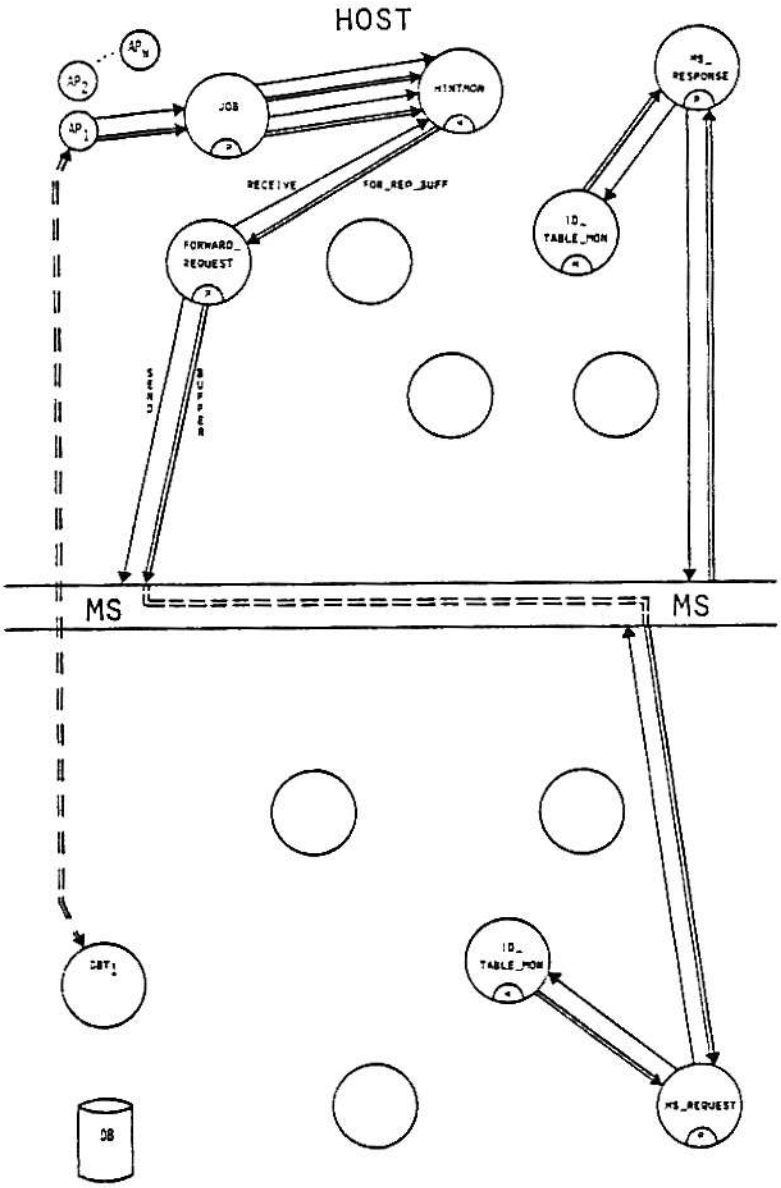


FIGURE 35
FORWARD_REQUEST PROCESS SENDS

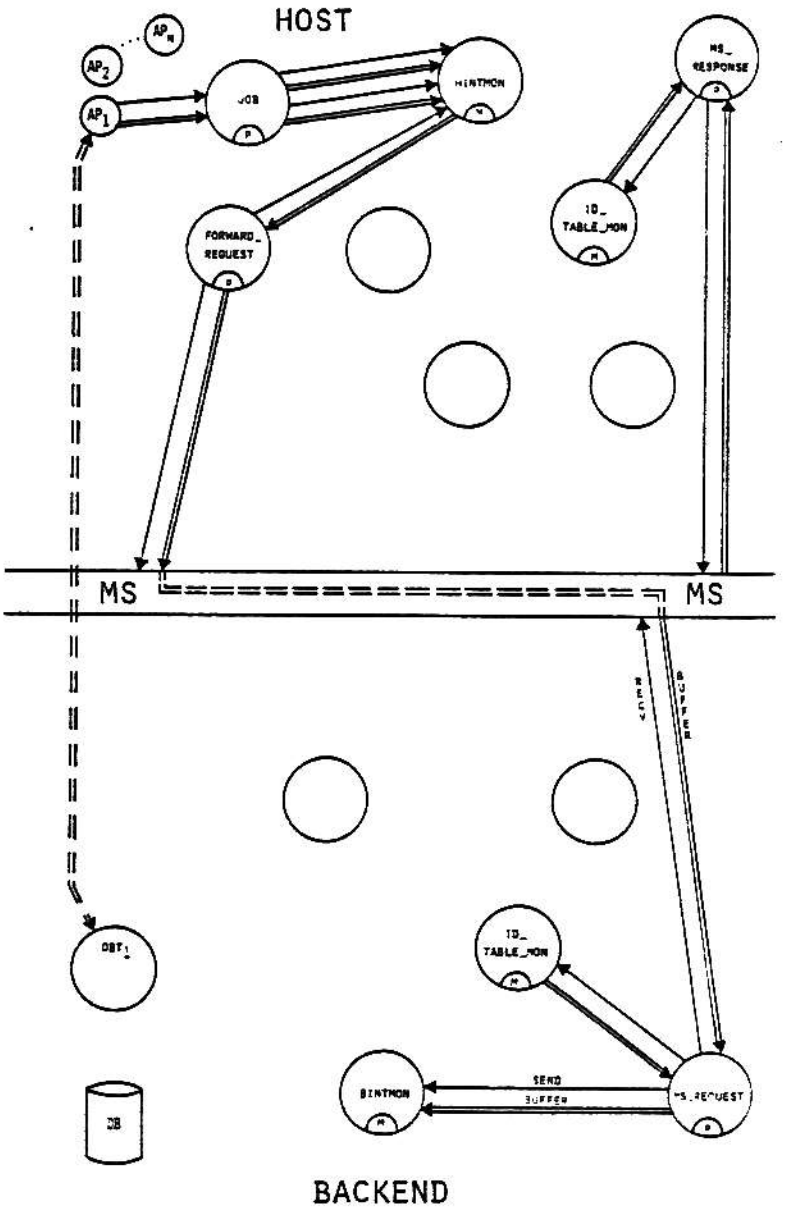


FIGURE 36
MS_REQUEST RECEIVEs

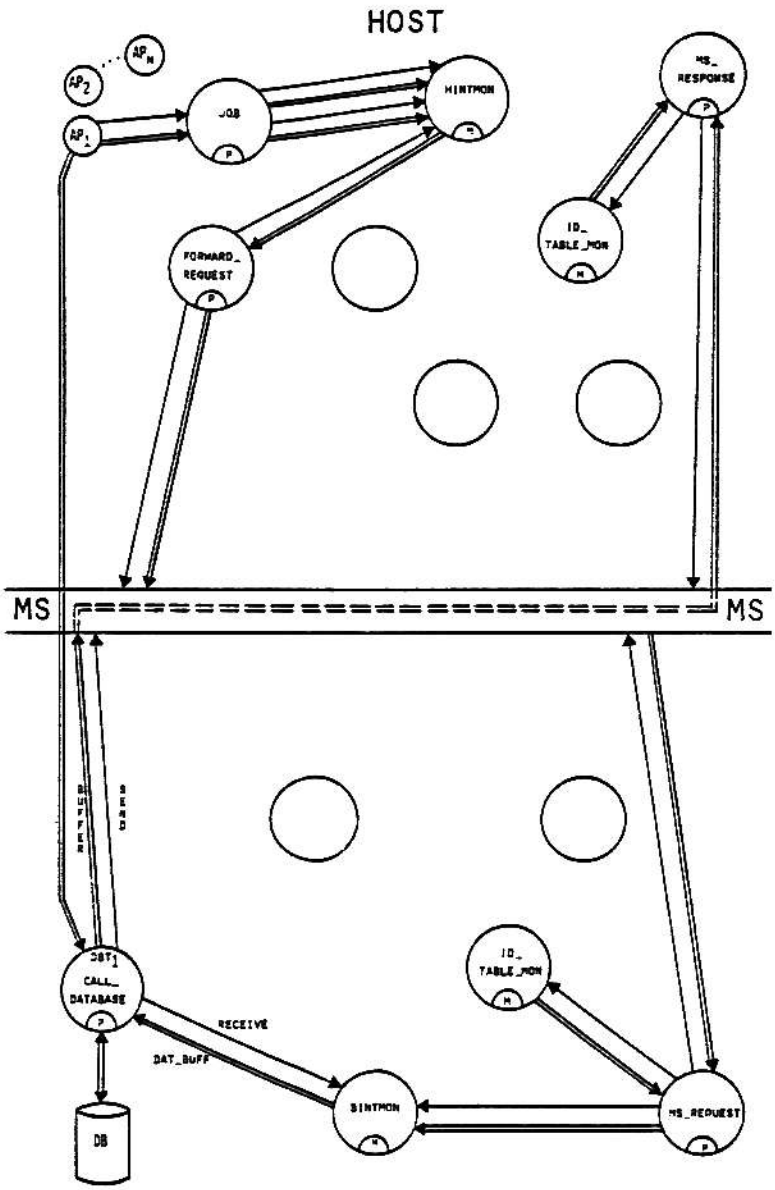


FIGURE 37

CALL_DATABASE PROCESS

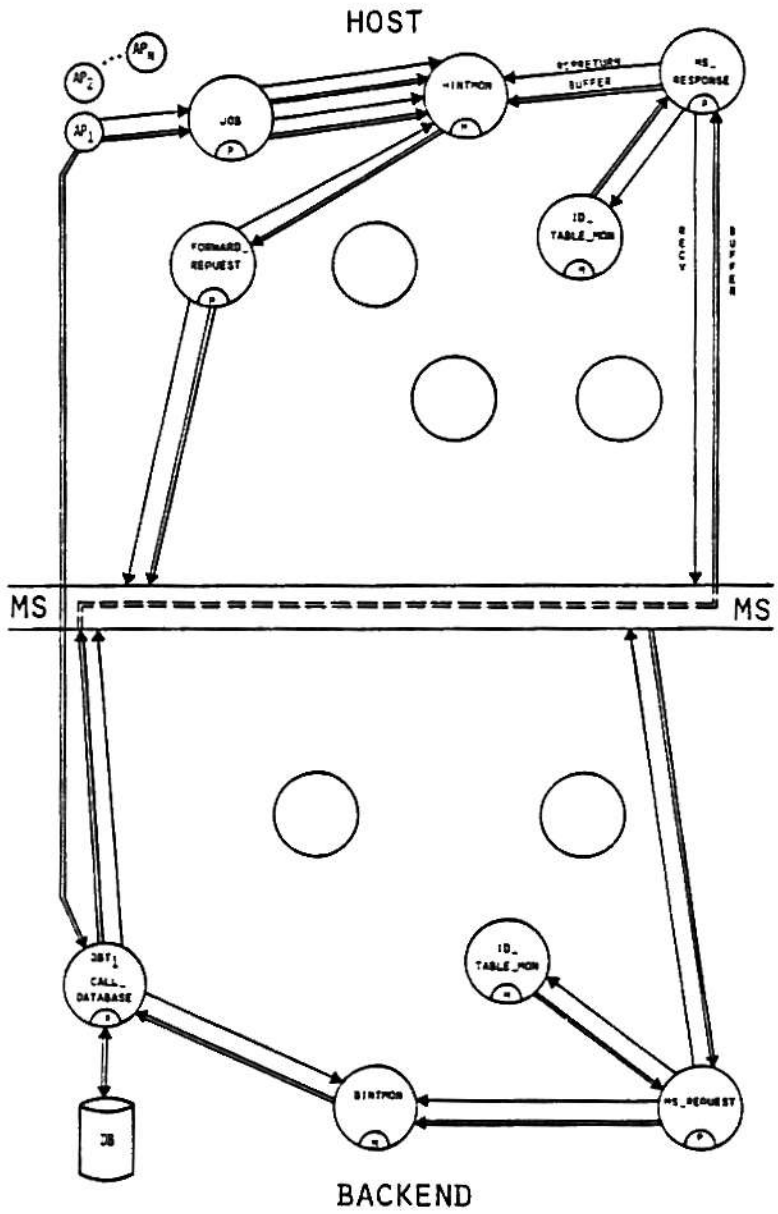


FIGURE 38

MS_RESPONSE RECEIVEs

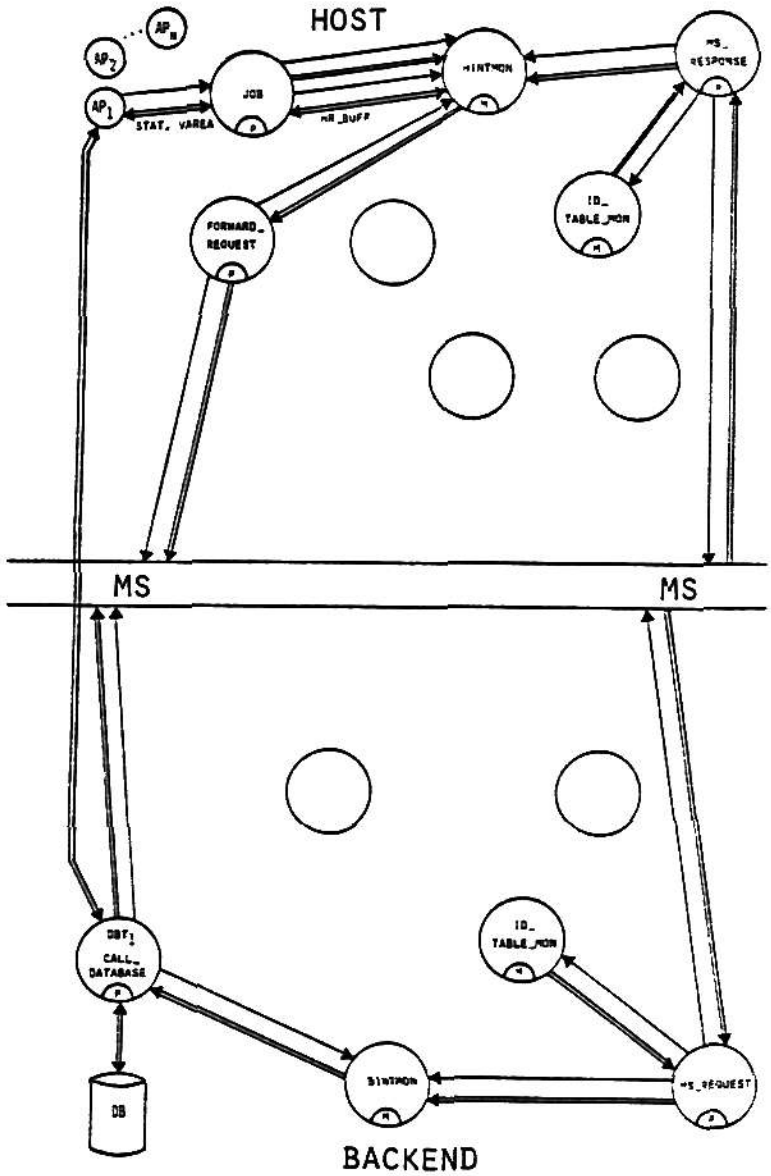


FIGURE 39

JOB PROCESS RETURNS TO AP

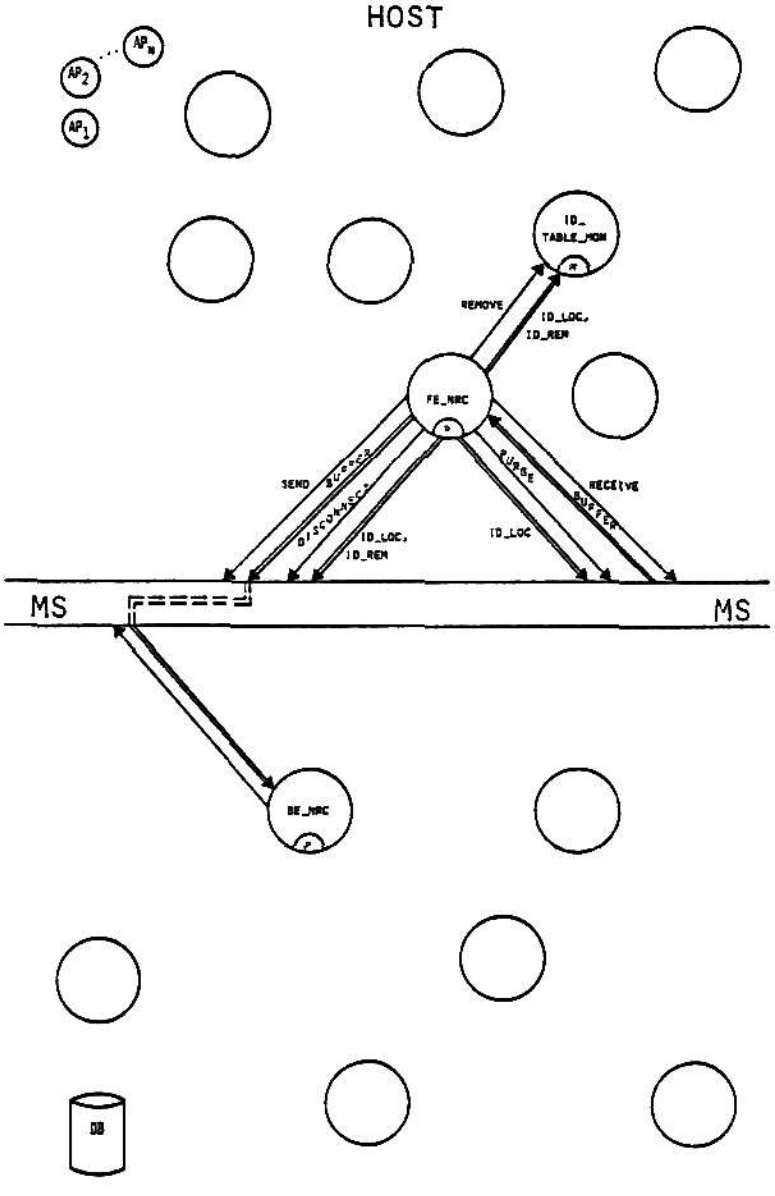


FIGURE 40

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 data base 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.

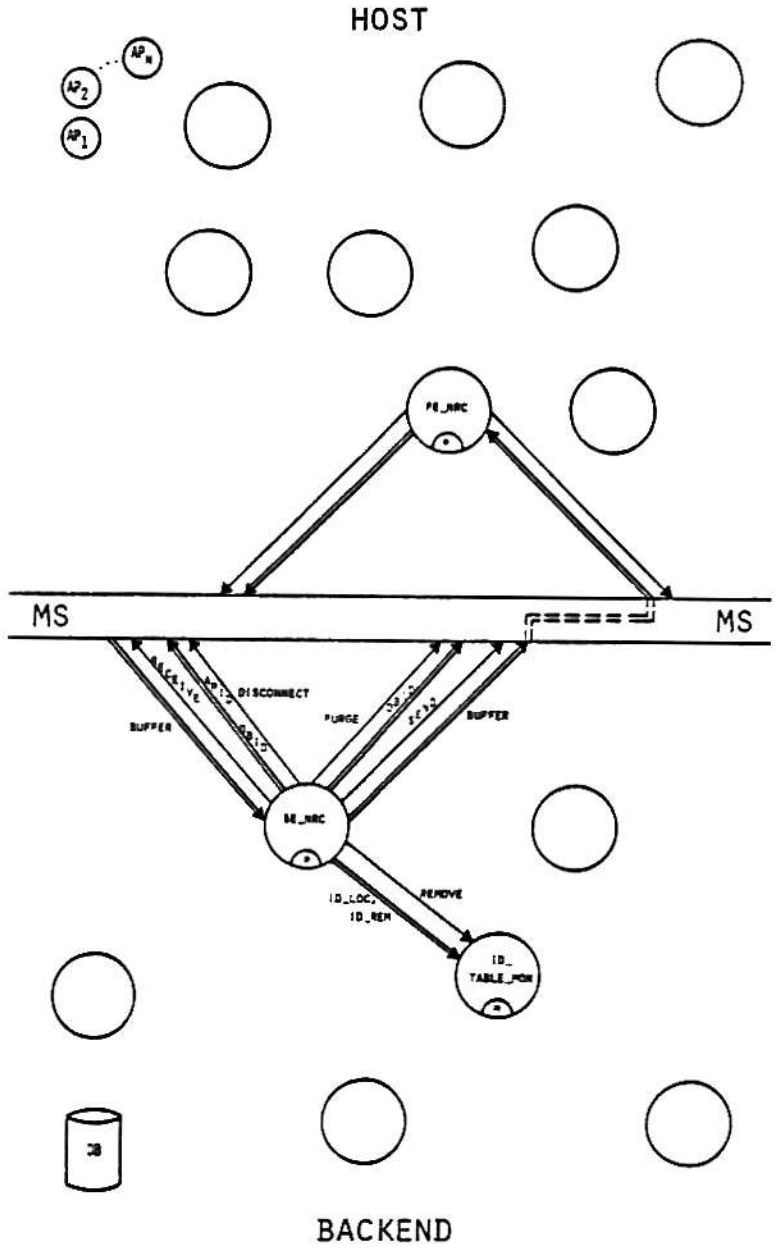


FIGURE 41
BE_NRC AT SINOF

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 SINOF 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 BINTMON 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.

HOURS		PHASE OF PROJECT
120		System Design
140		Walkthrough and Code
	20	Local Implementation Errors
	30	Test Without Message System
	30	Test Configuration
	60	Actual Test
	—	
140		Total Test
—		
400		

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 an 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 data base 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 truely multiple user operating system;
2. with multiple users, provide a Concurrent Pascal compatible, multiple threaded, data base management system so that multiple data base 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

1. Apfelbaum, Henry, Richard P. Case, J. Egil Juliussen, Bruce Shriver, Harold S. Stone, and James E. Thronton, "Computer System Organization: Problems of the 1980's," Computer, Volume 11, Number 9, September 1978, pp. 20-28.

2. Borgenson, Barry R., Garold S. Tjaden, and Merlin L. Hanson, "Mainframe Implementation with Off-the-Shelf LSI Modules," Computer, Volume 11, Number 7, July 1978, pp. 42-48.

3. Brinch Hansen, Per, The Architecture of Concurrent Programs, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

4. Denning, Peter J., "Operating Systems Principles for Data Flow Networks," Computer, Volume 11, Number 7, July 1978, pp. 86-96.

5. Enslow, Philip H. Jr., "What is a 'Distributed' Data Processing System," Computer, Volume 11, Number 1, January 1978, pp. 13-21.

6. Friedman, A. D., and Luea Simoncini, "The Effect of LSI Technology on the Theory of Modular Computer Design," Computer, Volume 11, Number 7, July 1978, pp. 60-67.

7. Gonzales, Mario J. Jr., "Workshop Report: Future Directions in Computer Architecture," Computer, Volume 11, Number 3, March 1978, pp. 54-62.

8. Hankley, William, "MIMICS Message System: Introduction and Users Guide," TR CS 78-03, Department of Computer Science, Kansas State University, Manhattan, Kansas 66506, January 1978.

9. Housh, Richard Dale, "A User Transparent Distributed Data Base Management System," Master's Report, Department of Computer Science, Kansas State University, Manhattan, Kansas 66506, 1978.

10. Jensen, E. Douglas, "The Honeywell Experimental Distributed Processor--An Overview," Computer, Volume 11, Number 1, January 1978, pp. 28-38.

11. Kartashev, Steven I., and Svetlana P. Kartashev, "LSI Modular Computers, Systems, and Networks," Computer, Volume 11, Number 7, July 1978, pp. 7-15.

12. Kartashev, Steven I., and Svetlana P. Kartashev, "Dynamic Architectures: Problems and Solutions," Computer, Volume 11, Number 7, July 1978, pp. 26-40.

13. Lipovski, G. Jack, and Keith L. Doty, "Developments and Directions in Computer Architecture," Computer, Volume 11, Number 8, August 1978, pp. 54-67.

14. Maryanski, Fred J., "A Survey of Developments in Distributed Data Base Management Systems," Computer, Volume 11, Number 2, February 1978, pp. 22-38.

15. Maryanski, Fred J. and Virgil E. Wallentine, "A Simulation Model of a Back-End Data Base Management System," Pittsburgh Conference on Modeling and Simulation, April 1976, pp. 243-248.

16. Maryanski, Fred J., Paul S. Fisher, Virgil E. Wallentine," A User-Transparent Mechanism for the Distribution of a CODASYL Data Base Management System," TR CS 76-22, Department of Computer Science, Kansas State University, Manhattan, Kansas 66506, December 1976.

17. Maryanski, Fred J., "Performance of Multi-Processor Back-End Data Base Systems," Conference on Information Sciences and Systems, April 1977, pp. 437-441.

18. Maryanski, Fred J., Virgil E. Wallentine, Paul S. Fisher, and Myron A. Calhoun, "Distributed Data Base Management Using Minicomputers," Minis Versus Mainframes, Infotech State-of-the Art Report, 1978, pp. 141-157.

19. Maryanski, Fred J., Paul S. Fisher, Richard D. Housh, and David A. Schmidt, "A Prototype Distributed DBMS," Hawaii International Conference on System Science, Vol. II, Jan. 1979, pp. 205-214.

20. Neal, David, "An Architectural Base for Concurrent Pascal," TR CS 76-19, Department of Computer Science, Kansas State University, Manhattan, Kansas 66506, July 1977.

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.

23. Peebles, Richard, and Eric Manning, "System Architecture for Distributed Data Management," Computer, Volume 11, Number 1, January 1978, pp. 40-47.

24. Shepherd, Mark Jr., "Distributed Computing Power: A Key to Productivity," Computer, Volume 10, Number 11, November 1977, pp. 66-77.

25. Siewiorek, D. P., D. E. Thomas, and D. L. Schufetter, "The Use of LSI Modules in Computer Structures: Trends and Limitations," Computer, Volume 11, Number 7, July 1978, pp. 16-25.

26. Stone, Harold S., and Shahid H. Bokhaii, "Control of Distributed Processes," Computer, Volume 11, Number 7, July 1978, pp. 97-106.

27. Wallentine, Virgil E., William J. Hankley, Gary G. Anderson, Myron A. Calhoun, and Fred J. Maryanski, "Progress Report on Functionally Distributed Computer Systems: Software and Systems Structure," TR CS 77-4, Department of Computer Science, Kansas State University, Manhattan, Kansas 66506, Dec. 1976.

28. Yao, S. B., Philip A. Bernstein, Nathan Goodman, Stewart A. Schuster, David W. Shipman, and Diane C. P. Smith, "Data-Base Systems," Computer, Volume 11, Number 9, September 1978, pp. 46-60.

APPENDIX A

```
 1
 2 (NUMBER)
 3
 4 "PER BRINCH HANSEN                          * AS MODIFIED FOR THE
 5                                             *    INTERDATA 8/32
 6                                             *    UNDER OS/32-MT AT
 7  INFORMATION                                * DEPT. OF COMPUTER
 8  CALIFORNIA INSTITUTE OF TECHNOLOGY         *    SCIENCE, KANSAS
 9                                             *    STATE UNIVERSITY
10  UTILITY PROGRAMS FOR                       *
11  THE SOLO SYSTEM                            *
12                                             *
13  18 MAY 1975                                *       1 DEC 1976"
14
15
16 "############
17 #  PREFIX  #
18 ############"
19
20
21 CONST NL = '(:10:)';   FF = '(:12:)';   CR = '(:13:)';
22       EM = '(:25:)';
23
24 CONST PAGELENGTH = 512;
25 TYPE PAGE = ARRAY (.1..PAGELENGTH.) OF CHAR;
26
27 CONST LINELENGTH = 132;
28 TYPE LINE = ARRAY (.1..LINELENGTH.) OF CHAR;
29
30 CONST IDLENGTH = 12;
31 TYPE IDENTIFIER = ARRAY (.1..IDLENGTH.) OF CHAR;
32
33 TYPE FILE = 1..2;
34
35 TYPE FILEKIND = (EMPTY, SCRATCH, ASCII, SEQCODE, CONCODE);
36
37 TYPE FILEATTR = RECORD
38                   KIND: FILEKIND;
39                   ADDR: INTEGER;
40                   PROTECTED: BOOLEAN;
41                   NOTUSED: ARRAY (.1..5.) OF INTEGER
42                 END;
43
44 TYPE IODEVICE =
45   (TYPEDEVICE, DISKDEVICE, TAPEDEVICE, PRINTDEVICE,
46    CARDDEVICE);
47
48 TYPE IOOPERATION = (INPUT, OUTPUT, MOVE, CONTROL);
49
50 TYPE IOARG = (WRITEEOF, REWIND, UPSPACE, BACKSPACE);
51
52 TYPE IORESULT =
53   (COMPLETE, INTERVENTION, TRANSMISSION, FAILURE,
54    ENDFILE, ENDMEDIUM, STARTMEDIUM);
```

```
 55
 56 TYPE IOPARAM = RECORD
 57                  OPERATION: IOOPERATION;
 58                  STATUS: IORESULT;
 59                  ARG: IOARG
 60                END;
 61
 62 TYPE TASKKIND = (INPUTTASK, JOBTASK, OUTPUTTASK);
 63
 64 TYPE ARGTAG =
 65   (NILTYPE, BOOLTYPE, INTTYPE, IDTYPE, PTRTYPE);
 66
 67 TYPE POINTER = @BOOLEAN;
 68
 69 TYPE ARGTYPE = RECORD
 70                  CASE TAG: ARGTAG OF
 71                    NILTYPE, BOOLTYPE: (BOOL: BOOLEAN);
 72                    INTTYPE: (INT: INTEGER);
 73                    IDTYPE: (ID: IDENTIFIER);
 74                    PTRTYPE: (PTR: POINTER)
 75                END;
 76
 77 CONST MAXARG = 10;
 78 TYPE ARGLIST = ARRAY (.1..MAXARG.) OF ARGTYPE;
 79
 80 TYPE ARGSEQ = (INP, OUT);
 81
 82 TYPE PROGRESULT =
 83   (TERMINATED, OVERFLOW, POINTERERROR, RANGEERROR,
 84    VARIANTERROR, HEAPLIMIT, STACKLIMIT, CODELIMIT,
 85    TIMELIMIT, CALLERROR);
 86
 87 TYPE A1              = ARRAY (.1..6.) OF CHAR;
 88
 89 TYPE A2              = ARRAY (.1..4.) OF CHAR;
 90
 91 TYPE A3              = ARRAY (.1..6.) OF CHAR;
 92
 93 TYPE A4              = ARRAY (.1..26.) OF CHAR;
 94
 95 TYPE A5              = ARRAY (.1..20.) OF CHAR;
 96
 97 TYPE A6              = ARRAY (.1..10.) OF CHAR;
 98
 99 TYPE A7              = ARRAY (.1..6.) OF CHAR;
100
101 TYPE A8              = ARRAY (.1..6.) OF CHAR;
102
103 TYPE A9              = ARRAY (.1..10.) OF CHAR;
104
105 TYPE ACCEPTMSG       = ARRAY (.1..26.) OF CHAR;
106
107 TYPE DISPLAYMSG      = ARRAY (.1..38.) OF CHAR;
108
```

```
109 TYPE ENV_FUNCTION    = ARRAY (.1..6.) OF CHAR;
110
111 TYPE ENV_STAT        = ARRAY (.1..4.) OF CHAR;
112
113 TYPE ENV_FILE_NAME   = ARRAY (.1..4.) OF CHAR;
114
115 TYPE ENV_REFER       = ARRAY (.1..4.) OF CHAR;
116
117 TYPE ENV_LINKPATH    = ARRAY (.1..8.) OF CHAR;
118
119 TYPE ENV_CTRL_FIELD = ARRAY (.1..10.) OF CHAR;
120
121 TYPE ENV_VELEMENTS   = ARRAY (.1..60.) OF CHAR;
122
123 TYPE ENV_VAREA       = ARRAY (.1..82.) OF CHAR;
124
125 TYPE ENV_ENDP        = ARRAY (.1..4.) OF CHAR;
126
127 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 (.1..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 (.1..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 (.1..110.) OF CHAR
161        END;
162
```

```
163 PROCEDURE READ(VAR C: CHAR);
164 PROCEDURE WRITE(C: CHAR);
165
166 PROCEDURE OPEN(F: FILE; ID: IDENTIFIER;
167                 VAR FOUND: BOOLEAN);
168 PROCEDURE CLOSE(F: FILE);
169 PROCEDURE GET(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);
170 PROCEDURE PUT(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);
171 FUNCTION LENGTH(F: FILE): INTEGER;
172
173 PROCEDURE MARK(VAR TOP: INTEGER);
174 PROCEDURE RELEASE(TOP: INTEGER);
175
176 PROCEDURE IDENTIFY(HEADER: LINE);
177 PROCEDURE ACCEPT(VAR C: CHAR);
178 PROCEDURE DISPLAY(C: CHAR);
179
180 PROCEDURE READPAGE(VAR BLOCK: UNIV PAGE;
181                    VAR EOF: BOOLEAN);
182 PROCEDURE WRITEPAGE(BLOCK: UNIV PAGE; EOF: BOOLEAN);
183 PROCEDURE READLINE(VAR TEXT: UNIV LINE);
184 PROCEDURE WRITELINE(TEXT: UNIV LINE);
185 PROCEDURE READARG(S: ARGSEQ; VAR ARG: ARGTYPE);
186 PROCEDURE WRITEARG(S: ARGSEQ; ARG: ARGTYPE);
187
188 PROCEDURE LOOKUP(ID: IDENTIFIER; VAR ATTR: FILEATTR;
189                  VAR FOUND: BOOLEAN);
190
191 PROCEDURE IOTRANSFER
192   (DEVICE: IODEVICE; VAR PARAM: IOPARAM;
193    VAR BLOCK: UNIV PAGE);
194
195 PROCEDURE IOMOVE(DEVICE: IODEVICE; VAR PARAM: IOPARAM);
196
197 FUNCTION TASK: TASKKIND;
198
199 PROCEDURE RUN(ID: IDENTIFIER; VAR PARAM: ARGLIST;
200               VAR LINE: INTEGER;
201               VAR RESULT: PROGRESULT);
202
203 PROCEDURE HINT(HFUNCTION : ENV_FUNCTION;
204                 VAR STAT : ENV_STAT;
205                FILE_NAME : ENV_FILE_NAME;
206                    REFER : ENV_REFER;
207                 LINKPATH : ENV_LINKPATH;
208               CTRL_FIELD : ENV_CTRL_FIELD;
209                VELEMENTS : ENV_VELEMENTS;
210                VAR VAREA : ENV_VAREA;
211                     ENDP : ENV_ENDP);
212
213 PROGRAM P(VAR PARAM: ARGLIST);
214
215 VAR SINON,DATA1:ENV_FUNCTION;
216     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
248  BEGIN
249    SINOF := ' SINOF';
250    DUMMY1 := '     ';
251    DUMMY2 := '          ';
252    DUMMY3 := '           ';
253    DUMMY4 := '                                        ',
254                '                                          ';
255    ENDP := 'END.';
256    HINT (SINOF, STAT, DUMMY1, DUMMY1, DUMMY2,
257          DUMMY3, DUMMY4, SCHEMA, ENDP);
258          "SIGN OFF THE DATA BASE."
259  END;
260
261  PROCEDURE MTDISPLAY (DMSG : DISPLAYMSG);
262  VAR I : INTEGER; C : CHAR;
263
264  BEGIN
265    FOR I := 1 TO 38 DO
266        BEGIN
267          C := DMSG (.I.);
268          DISPLAY (C);
269        END;
270  END;
```

```
271
272 PROCEDURE MTACCEPT (ANUM : INTEGER;
273                      VAR AMSG : ACCEPTMSG);
274 VAR I,J,AMAX : INTEGER; C : CHAR;
275
276 BEGIN
277   AMAX := 26;
278   FOR I := 1 TO ANUM DO
279       BEGIN
280         ACCEPT (C);
281         AMSG (.I.) := C;
282       END;
283   FOR J := I + 1 TO AMAX DO
284       BEGIN
285         AMSG (.J.) :=' ';
286       END;
287 END;
288
289 PROCEDURE MASTER_TRANS (FCOMMAND : ENV_FUNCTION);
290 VAR MTAMSG, DUM1 : ACCEPTMSG; XNUMBER : A3;
291     NAME : A4; XADDRESS : A5; I,ANUM : INTEGER;
292     ENUMBER, TELEPHONE : A6; HIRE_DATE : A7;
293     BIRTH_DATE : A8; SOC_NUMB : A9;
294     D3,D4,D5,D6,D7,D8,D9 : DISPLAYMSG;
295     HFUNCTION,DATA1 : ENV_FUNCTION;
296     STAT,STATERR,DATA2 : ENV_STAT;
297     PEOP : ENV_FILE_NAME; DUMMY1 : ENV_REFER;
298     DUMMY2 : ENV_LINKPATH;
299     PEOP_DATA : ENV_VELEMENTS;
300     PEOP_RECORD : ENV_VAREA; ENDP : ENV_ENDP;
301     PRTLINE : PRINT_LINE; PEPIN : PEOP_IN;
302     H3A, H3B, H4A, H4B, H5A, H5B, H6A, H6B: DISPLAYMSG;
303     H7A, H7B, H8A, H8B, H9A, H9B: DISPLAYMSG;
304     DBRTN1, DBRTN2: DISPLAYMSG;
305
306 BEGIN
307   D3    := 'ENTER 6 DIGIT EMPLOYEE NUMBER.       (:10:)';
308   H3A   := '          C                          (:10:)';
309   H3B   := '  ****5*R                            (:10:)';
310   D4    := 'ENTER 25 CHARACTER NAME.             (:10:)';
311   H4A   := '          1    1    2    2C         (:10:)';
312   H4B   := '  ****5****0****5****0****5R         (:10:)';
313   D5    := 'ENTER 20 CHARACTER ADDRESS.          (:10:)';
314   H5A   := '          1    1    2C              (:10:)';
315   H5B   := '  ****5****0****5****0R              (:10:)';
316   D6    := 'ENTER 10 DIGIT TELEPHONE NUMBER.     (:10:)';
317   H6A   := '          1C                         (:10:)';
318   H6B   := '  ****5****0R                        (:10:)';
319   D7    := 'ENTER 6 DIGIT HIRE DATE.             (:10:)';
320   H7A   := '          C                          (:10:)';
321   H7B   := '  ****5*R                            (:10:)';
322   D8    := 'ENTER 6 DIGIT BIRTH DATE.            (:10:)';
323   H8A   := '          C                          (:10:)';
324   H8B   := '  ****5*R                            (:10:)';
```

```
325   D9    := 'ENTER 9 DIGIT SSN.                    (:10:)';
326   H9A   := '              C                       (:10:)';
327   H9B   := '   ****5****R                         (:10:)';
328   DBRTN1:= 'RETURNED STATUS --                     ';
329   DBRTN2:= 'RETURNED VARIABLE AREA --              ';
330   STATERR := '****';
331   ANUM:=1;
332   MTACCEPT(ANUM,DUM1);
333   MTDISPLAY (D3);
334   MTDISPLAY (H3A);
335   MTDISPLAY (H3B);
336   ANUM := 6;
337   MTACCEPT (ANUM, MTAMSG);
338   FOR I := 1 TO ANUM DO
339       XNUMBER (.I.) := MTAMSG (.I.);
340   ANUM:=1;
341   MTACCEPT(ANUM,DUM1);
342   MTDISPLAY (D4);
343   MTDISPLAY (H4A);
344   MTDISPLAY (H4B);
345   ANUM := 25;
346   MTACCEPT (ANUM, MTAMSG);
347   NAME(.1.):=' ';
348   FOR I := 1 TO ANUM DO
349       NAME (.I+1.) := MTAMSG (.I.);
350   ANUM:=1;
351   MTACCEPT(ANUM,DUM1);
352   MTDISPLAY (D5);
353   MTDISPLAY (H5A);
354   MTDISPLAY (H5B);
355   ANUM := 20;
356   MTACCEPT (ANUM, MTAMSG);
357   FOR I := 1 TO ANUM DO
358       XADDRESS (.I.) := MTAMSG (.I.);
359   ANUM:=1;
360   MTACCEPT(ANUM,DUM1);
361   MTDISPLAY (D6);
362   MTDISPLAY (H6A);
363   MTDISPLAY (H6B);
364   ANUM := 10;
365   MTACCEPT (ANUM, MTAMSG);
366   FOR I := 1 TO ANUM DO
367       TELEPHONE (.I.) := MTAMSG (.I.);
368   ANUM:=1;
369   MTACCEPT(ANUM,DUM1);
370   MTDISPLAY (D7);
371   MTDISPLAY (H7A);
372   MTDISPLAY (H7B);
373   ANUM := 6;
374   MTACCEPT (ANUM, MTAMSG);
375   FOR I := 1 TO ANUM DO
376       HIRE_DATE (.I.) := MTAMSG (.I.);
377   ANUM:=1;
378   MTACCEPT(ANUM,DUM1);
```

```
379   MTDISPLAY (D8);
380   MTDISPLAY (H8A);
381   MTDISPLAY (H8B);
382   ANUM := 6;
383   MTACCEPT (ANUM, MTAMSG);
384   FOR I := 1 TO ANUM DO
385       BIRTH_DATE (.I.) := MTAMSG (.I.);
386   ANUM:=1;
387   MTACCEPT(ANUM,DUM1);
388   MTDISPLAY (D9);
389   MTDISPLAY (H9A);
390   MTDISPLAY (H9B);
391   ANUM := 9;
392   MTACCEPT (ANUM, MTAMSG);
393   SOC_NUMB(.1.):=' ';
394   FOR I := 1 TO ANUM DO
395       SOC_NUMB(.I+1.) := MTAMSG (.I.);
396   WITH PEPIN DO
397     BEGIN
398       PEOP_NUM := XNUMBER;
399       FILLER1:='       ';
400       PEOP_NAM := NAME;
401       FILLER2:='       ';
402       PEOP_ADR := XADDRESS;
403       FILLER3:='       ';
404       PEOP_TEL := TELEPHONE;
405       FILLER4:='       ';
406       PEOP_HIR := HIRE_DATE;
407       FILLER5:='       ';
408       PEOP_BRT := BIRTH_DATE;
409       FILLER6:='       ';
410       PEOP_SOC := SOC_NUMB;
411     END;
412   RECDCNVT(PEPIN,OUTLINE);
413   OUTLINE(.132.):=NL;
414   FOR I:=1 TO 132 DO
415     BEGIN
416       C:=OUTLINE(.I.);
417       WRITE(C);
418     END;
419   FOR I:= 1 TO 4 DO
420     ENUMBER(.I.):=' ';
421   FOR I:= 1 TO 6 DO
422     ENUMBER(.I+4.):=XNUMBER(.I.);
423   HFUNCTION := FCOMMAND;
424   PEOP:='PEOP';
425   DUMMY1:='    ';
426   DUMMY2:='         ';
427   ENDP:='END.';
428   FOR I:= 1 TO 60 DO
429     PEOP_DATA(.I.):=OUTLINE(.I.);
430   HINT (HFUNCTION, STAT, PEOP, DUMMY1, DUMMY2,
431         ENUMBER, PEOP_DATA, PEOP_RECORD, ENDP);
432   FOR I:= 1 TO 38 DO
```

```
433     OUTLINE(.I.):=DBRTN1(.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.):=' ';
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.):= ' ';
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 PRTLINE DO
457               BEGIN
458                 FILLER1:='       ';
459                 DATA1:=HFUNCTION;
460                 FILLER2:='       ';
461                 DATA2:=STAT;
462               END;
463             CRT (PRTLINE);
464             RECDCNVT(PRTLINE,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:)';
```

```
487    H2B:= '   *R                                          (:10:)';
488    CLOSE:=FALSE;
489    REPEAT
490        MTDISPLAY (D1);
491        MTDISPLAY (H1A);
492        MTDISPLAY (H1B);
493        ANUM := 5;
494        MTACCEPT (ANUM, MTAMSG);
495        COMMAND(.1.):=' ';
496        FOR I := 1 TO ANUM DO
497            COMMAND (.I+1.) := MTAMSG (.I.);
498        ANUM:=1;
499        MTACCEPT(ANUM,DUM2);
500        MTDISPLAY (D2);
501        MTDISPLAY (H2A);
502        MTDISPLAY (H2B);
503        MTACCEPT (ANUM, MTAMSG);
504        FOR I:=1 TO 3 DO
505          BEGIN
506            NO_TIMES(.I.):=' ';
507          END;
508        FOR I := 1 TO ANUM DO
509            NO_TIMES (.I+3.) := MTAMSG (.I.);
510        WITH PRTCOMMAND DO
511          BEGIN
512            FILLER1:='       ';
513            COMM:=COMMAND;
514            FILLER2:='       ';
515            NUMB:=NO_TIMES;
516          END;
517        CRT (PRTCOMMAND);
518        RECDCNVT(PRTCOMMAND,OUTLINE);
519        OUTLINE(.132.):=NL;
520        FOR I:= 1 TO 132 DO
521          BEGIN
522            C:=OUTLINE(.I.);
523            WRITE(C);
524          END;
525        IF COMMAND =' CEASE'
526           THEN CLOSE:= TRUE
527           ELSE BEGIN
528                  I := 4;
529                  LIMIT:=ORD(NO_TIMES(.I.))-48;
530                  I:=1;
531                  REPEAT
532                      MASTER_TRANS (COMMAND);
533                      I := I + 1;
534                    UNTIL I > LIMIT;
535                  ANUM:=1;
536                  MTACCEPT(ANUM,DUM2);
537                END;
538      UNTIL CLOSE;
539    CLOSE_EM;
540  END;
```

```
541
542 "*****************************
543 *  SPASCAL USER'S PROGRAM   *
544 *****************************"
545
546 BEGIN
547   OUTARG.TAG:=IDTYPE;
548   OUTARG.ID:='PRINTER      ';
549   WRITEARG(OUT,OUTARG);
550   STATERR := '****';
551   SINON := ' SINON';
552   DUMMY1 := '     ';
553   DUMMY2 := '         ';
554   DUMMY3 := '           ';
555   DUMMY4 := '                                        ',
556             '                                          ';
557   ENDP := 'END.';
558   HINT (SINON, STAT, DUMMY1, DUMMY1, DUMMY2,
559         DUMMY3, DUMMY4, SCHEMA, ENDP);
560   IF STAT = STATERR
561      THEN GET_COMMAND
562      ELSE BEGIN
563             WITH PRTLINE DO
564               BEGIN
565                 FILLER1:='       ';
566                 DATA1:=' SINON';
567                 FILLER2:='       ';
568                 DATA2:=STAT;
569               END;
570             CRT (PRTLINE);
571             RECDCNVT(PRTLINE,OUTLINE);
572             OUTLINE(.132.):=NL;
573             FOR I:= 1 TO 132 DO
574               BEGIN
575                 C:=OUTLINE(.I.);
576                 WRITE(C);
577               END;
578             CLOSE_EM;
579           END;
580   OUTLINE(.1.):=EM;
581   OUTLINE(.2.):=NL;
582   FOR I:= 1 TO 2 DO
583     BEGIN
584       C:=OUTLINE(.I.);
585       WRITE(C);
586     END;
587   READARG(OUT,OUTARG);
588 END.
```

APPENDIX B

```
 1 (NUMBER, ERRORS ONLY)
 2
 3
 4
 5 " PER BRINCH HANSEN
 6
 7   INFORMATION SCIENCE
 8   CALIFORNIA INSTITUTE OF TECHNOLOGY
 9
10   THE SOLO SYSTEM
11   8 JUNE 1975"
12
13 "DAVID NEAL
14
15  DEPARTMENT OF COMPUTER SCIENCE
16  KANSAS STATE UNIVERSITY
17
18  MODIFICATIONS OF THE SOLO SYSTEM FOR USE
19  UNDER INTERDATA'S OS/32-MT AND FOR THE USE
20  OF TWO VIRTUAL DISKS
21
22  OCTOBER 15, 1977"
23
24
25 "##############
26 #  IO TYPES   #
27 ##############"
28
29
30 TYPE IODEVICE =
31   (TYPEDEVICE, PRIVATEDISK, TAPEDEVICE, PRINTDEVICE,
32    CARDDEVICE, TAPE2DEVICE, ITAMDEVICE, VARDEVICE,
33    SYSTEMDISK);        "KSU"
34
35 TYPE IOARG =
36   (WRITEEOF, REWIND, UPSPACE, BACKSPACE);
37
38 TYPE IOOPERATION = (INPUT, OUTPUT, MOVE, CONTROL);
39
40 TYPE IORESULT = (COMPLETE,INTERVENTION,TRANSMISSION,
41          FAILURE, ENDFILE, ENDMEDIUM, STARTMEDIUM);
42
43
44 TYPE IOPARAM = RECORD
45                  OPERATION: IOOPERATION;
46                  STATUS: IORESULT;
47                  ARG: IOARG
48                END;
49
50 CONST LINELENGTH = 132;
51 TYPE LINE = ARRAY [1..LINELENGTH] OF CHAR;
52
53 CONST PAGELENGTH = 512;
54 TYPE PAGE = ARRAY [1..PAGELENGTH] OF CHAR;
```

```
 55
 56 TYPE OKTYPE = INTEGER;
 57
 58 TYPE NOT_DONE_TYPE = INTEGER;
 59
 60
 61 CONST NL = '(:10:)'; CR = '(:13:)';     NULL = '(:0:)';
 62       FF = '(:12:)'; EM = '(:25:)';
 63
 64
 65 "############################
 66 #  PROCESSQUEUE AND FIFO   #
 67 ############################"
 68
 69
 70 CONST PROCESSCOUNT = 18;
 71 TYPE PROCESSQUEUE = ARRAY [1..PROCESSCOUNT] OF QUEUE;
 72
 73
 74 TYPE FIFO =
 75 CLASS(LIMIT: INTEGER);
 76
 77 VAR HEAD, TAIL, LENGTH: INTEGER;
 78
 79 FUNCTION ENTRY ARRIVAL: INTEGER;
 80 BEGIN
 81   ARRIVAL:= TAIL;
 82   TAIL:= TAIL MOD LIMIT + 1;
 83   LENGTH:= LENGTH + 1;
 84 END;
 85
 86 FUNCTION ENTRY DEPARTURE: INTEGER;
 87 BEGIN
 88   DEPARTURE:= HEAD;
 89   HEAD:= HEAD MOD LIMIT + 1;
 90   LENGTH:= LENGTH - 1;
 91 END;
 92
 93 FUNCTION ENTRY EMPTY: BOOLEAN;
 94 BEGIN EMPTY:= (LENGTH = 0) END;
 95
 96 FUNCTION ENTRY FULL: BOOLEAN;
 97 BEGIN FULL:= (LENGTH = LIMIT) END;
 98
 99 BEGIN HEAD:= 1; TAIL:= 1; LENGTH:= 0 END;
100
101
102 "#############
103 #  RESOURCE   #
104 #############"
105
106
107 TYPE RESOURCE =
108 MONITOR
```

```
109
110 VAR FREE: BOOLEAN; Q: PROCESSQUEUE; NEXT: FIFO;
111
112 PROCEDURE ENTRY REQUEST;
113 BEGIN
114   IF FREE THEN FREE:= FALSE
115     ELSE DELAY(Q[NEXT.ARRIVAL]);
116 END;
117
118 PROCEDURE ENTRY RELEASE;
119 BEGIN
120   IF NEXT.EMPTY THEN FREE:= TRUE
121     ELSE CONTINUE(Q[NEXT.DEPARTURE]);
122 END;
123
124 BEGIN FREE:= TRUE; INIT NEXT(PROCESSCOUNT) END;
125
126
127 "##################
128 #  TYPERESOURCE   #
129 ##################"
130
131
132 TYPE TYPERESOURCE =
133 MONITOR
134
135 VAR FREE: BOOLEAN; Q: PROCESSQUEUE; NEXT: FIFO;
136     HEADER: LINE;
137
138 PROCEDURE ENTRY REQUEST(TEXT: LINE;
139                         VAR CHANGED: BOOLEAN);
140 BEGIN
141   IF FREE THEN FREE:= FALSE
142     ELSE DELAY(Q[NEXT.ARRIVAL]);
143   CHANGED:= (HEADER <> TEXT);
144   HEADER:= TEXT;
145 END;
146
147 PROCEDURE ENTRY RELEASE;
148 BEGIN
149   IF NEXT.EMPTY THEN FREE:= TRUE
150     ELSE CONTINUE(Q[NEXT.DEPARTURE]);
151 END;
152
153 BEGIN
154   FREE:= TRUE; HEADER[1]:= NL;
155   INIT NEXT(PROCESSCOUNT);
156 END;
157
158
159 "################
160 #  TYPEWRITER   #
161 ################"
162
```

```
163
164 TYPE TYPEWRITER =
165 CLASS(DEVICE: IODEVICE);
166
167 CONST LINELIMIT = 81;
168   CANCELCHAR = '\'; "KSU - IMPLEMENTED BY"
169                     "OS/32-MT DRIVERS"
170   CANCELLINE = '#'; "KSU - IMPLEMENTED BY"
171                     "OS/32-MT DRIVERS"
172
173 PROCEDURE WRITECHAR(X: CHAR);
174 VAR PARAM: IOPARAM; C: CHAR;
175 BEGIN
176   PARAM.OPERATION:= OUTPUT;
177   C:= X;
178   IO(C, PARAM, DEVICE);
179 END;
180
181 PROCEDURE ENTRY WRITE(TEXT: LINE);
182 VAR PARAM: IOPARAM;
183     I: INTEGER; C: CHAR;
184 BEGIN
185   PARAM.OPERATION:= OUTPUT;
186   I:= 0;
187   REPEAT
188     I:= I + 1; C:= TEXT[I];
189     IO(C, PARAM, DEVICE);
190   UNTIL (C = NL) OR (I = LINELIMIT);
191   IF C <> NL THEN WRITECHAR(NL);
192 END;
193
194 PROCEDURE ENTRY READ(VAR TEXT: LINE);
195 CONST BEL = '(:7:)';
196 VAR PARAM: IOPARAM;
197     I: INTEGER; C: CHAR;
198 BEGIN
199   WRITECHAR(BEL);
200   PARAM.OPERATION:= INPUT;
201   I:= 0;
202   REPEAT
203     IO(C, PARAM, DEVICE);
204         "CODE TO CANCEL INPUT LINES"
205         "AND CHARACTERS DELETED - KSU"
206      I:= I + 1; TEXT[I]:= C;
207   UNTIL (C = NL) OR (I = LINELIMIT);
208   IF C <> NL THEN
209   BEGIN
210     WRITECHAR(NL);
211     TEXT[LINELIMIT + 1]:= NL;
212   END;
213 END;
214
215 BEGIN END;
216
```

```
217
218 "##############
219 #  TERMINAL   #
220 ##############"
221
222
223 TYPE TERMINAL =
224 CLASS(ACCESS: TYPERESOURCE);
225
226 VAR UNIT: TYPEWRITER;
227
228 PROCEDURE ENTRY READ(HEADER: LINE; VAR TEXT: LINE);
229 VAR CHANGED: BOOLEAN;
230 BEGIN
231   ACCESS.REQUEST(HEADER, CHANGED);
232   IF CHANGED THEN UNIT.WRITE(HEADER);
233   UNIT.READ(TEXT);
234   ACCESS.RELEASE;
235 END;
236
237 PROCEDURE ENTRY WRITE(HEADER, TEXT: LINE);
238 VAR CHANGED: BOOLEAN;
239 BEGIN
240   ACCESS.REQUEST(HEADER, CHANGED);
241   IF CHANGED THEN UNIT.WRITE(HEADER);
242   UNIT.WRITE(TEXT);
243   ACCESS.RELEASE;
244 END;
245
246 BEGIN INIT UNIT(TYPEDEVICE) END;
247
248
249 "####################
250 #  TERMINALSTREAM   #
251 ####################"
252
253
254 TYPE TERMINALSTREAM =
255 CLASS(OPERATOR: TERMINAL);
256
257 CONST LINELIMIT = 80;
258
259 VAR HEADER: LINE; ENDINPUT: BOOLEAN;
260     INP, OUT: RECORD COUNT: INTEGER; TEXT: LINE END;
261
262 PROCEDURE INITIALIZE(TEXT: LINE);
263 BEGIN
264   HEADER:= TEXT;
265   ENDINPUT:= TRUE;
266   OUT.COUNT:= 0;
267 END;
268
269 PROCEDURE ENTRY READ(VAR C: CHAR);
270 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
309 TYPE DISK =
310 CLASS(TYPEUSE: TYPERESOURCE);
311
312 "DISKDEVICE ADDED TO ALL PROCEDURES TO ALLOW       - KSU
313     THE USE OF MULTIPLE VIRTUAL DISKS              - KSU"
314
315 VAR OPERATOR: TERMINAL;
316
317 FUNCTION XLTESTAT(STATUS: UNIV INTEGER): INTEGER;
318 BEGIN
319   XLTESTAT:= STATUS;
320 END;
321
322 PROCEDURE TRANSFER(COMMAND: IOOPERATION;
323   PAGEADDR: UNIV IOARG; VAR BLOCK: PAGE;
324   DISKDEVICE: IODEVICE);
```

```
325
326     "STATUS CHANGES MADE FOR OS/32-MT FOIBLES     - KSU"
327
328 VAR PARAM: IOPARAM; RESPONSE: LINE;
329 BEGIN
330   WITH PARAM, OPERATOR DO
331   BEGIN
332     OPERATION:= COMMAND;
333     ARG:= PAGEADDR;
334     IO(BLOCK, PARAM, DISKDEVICE);
335     IF STATUS = ENDFILE THEN                    "KSU"
336       STATUS:= COMPLETE;                        "KSU"
337     WHILE STATUS <> COMPLETE DO
338     BEGIN
339       RESPONSE[1]:= CHR(XLTESTAT(STATUS) + ORD('0'));
340       RESPONSE[2]:= NL;
341       WRITE('DISK:  ERROR(:10:) ', RESPONSE);
342       READ('PUSH RETURN(:10:)', RESPONSE);
343       IO(BLOCK, PARAM, DISKDEVICE);
344     END;
345   END;
346 END;
347
348 PROCEDURE ENTRY READ(PAGEADDR: INTEGER;
349                      VAR BLOCK: UNIV PAGE;
350                      DISKDEVICE: IODEVICE);
351 BEGIN TRANSFER(INPUT, PAGEADDR, BLOCK, DISKDEVICE) END;
352
353 PROCEDURE ENTRY WRITE(PAGEADDR: INTEGER;
354                       VAR BLOCK: UNIV PAGE;
355                       DISKDEVICE: IODEVICE);
356 BEGIN TRANSFER(OUTPUT, PAGEADDR, BLOCK, DISKDEVICE); END;
357
358 BEGIN INIT OPERATOR(TYPEUSE) END;
359
360
361 "############################
362 #  FILEMAP AND DISKFILE  #
363 ############################"
364
365
366 CONST MAPLENGTH = 255;
367 TYPE FILEMAP = RECORD
368                  FILELENGTH: INTEGER;
369                  PAGESET: ARRAY [1..MAPLENGTH] OF INTEGER
370                END;
371
372
373 TYPE DISKFILE =
374 CLASS(TYPEUSE: TYPERESOURCE);
375
376 "IODEVICE ADDED TO PROCEDURES TO ALLOW THE USE     - KSU
377        OF MULTIPLE VIRTUAL DISKS                   - KSU"
378
```

```
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 "########################
424 #  CATALOG STRUCTURE   #
425 ########################"
426
427
428 CONST IDLENGTH = 12;
429 TYPE IDENTIFIER = ARRAY [1..IDLENGTH] OF CHAR;
430
431 TYPE FILEKIND =
432   (EMPTY, SCRATCH, ASCII, SEQCODE, CONCODE);
```

```
433
434
435 TYPE FILEATTR = RECORD
436                    KIND: FILEKIND;
437                    ADDR: INTEGER;
438                    PROTECTED: BOOLEAN;
439                    NOTUSED: ARRAY [1..5] OF INTEGER
440                  END;
441
442 TYPE CATENTRY = RECORD
443                    ID: IDENTIFIER;
444                    ATTR: FILEATTR;
445                    KEY, SEARCHLENGTH: INTEGER
446                  END;
447
448 CONST CATPAGELENGTH = 16;
449 TYPE CATPAGE = ARRAY [1..CATPAGELENGTH] OF CATENTRY;
450
451 CONST CATADDR = 154;
452
453
454 "###############
455 #  DISKTABLE  #
456 ###############"
457
458
459 TYPE DISKTABLE =
460 CLASS(TYPEUSE: TYPERESOURCE;
461       CATADDR: INTEGER);
462
463 "DISKDEVICE PERMANENT VARIABLE AND DISK PARAMETER  - KSU"
464 "ADDED TO ALLOW FOR USE OF MULTIPLE VIRTUAL DISKS  - KSU"
465
466 VAR FILE: DISKFILE;
467     PAGENO: INTEGER; DISKDEVICE: IODEVICE;
468     BLOCK: CATPAGE;
469
470 ENTRY FLENGTH: INTEGER;
471
472 PROCEDURE ENTRY READ(I: INTEGER; VAR ELEM: CATENTRY;
473                      DISK: IODEVICE);
474 VAR INDEX: INTEGER;
475 BEGIN
476   IF DISK <> DISKDEVICE THEN
477     WITH FILE DO        "KSU - ADDED TO SWITCH BETWEEN"
478                         "VIRTUAL DISKS"
479     BEGIN
480       CLOSE;
481       FILE.OPEN(CATADDR, DISK);
482       FLENGTH:= FILE.LENGTH * CATPAGELENGTH;
483       PAGENO:= 0;
484       DISKDEVICE:= DISK;
485     END;
486   INDEX:= (I - 1) DIV CATPAGELENGTH + 1;
```

```
487    IF PAGENO <> INDEX THEN
488    BEGIN
489      PAGENO:= INDEX;
490      FILE.READ(PAGENO, BLOCK, DISK);
491    END;
492    ELEM:= BLOCK[(I - 1) MOD CATPAGELENGTH + 1];
493  END;
494
495  BEGIN
496    INIT FILE(TYPEUSE);
497    DISKDEVICE:= PRIVATEDISK;
498    FILE.OPEN(CATADDR, DISKDEVICE);
499    FLENGTH:= FILE.LENGTH * CATPAGELENGTH;
500    PAGENO:= 0;
501  END;
502
503
504  "################
505  #  DISKCATALOG  #
506  ################"
507
508
509  TYPE DISKCATALOG =
510  MONITOR(TYPEUSE: TYPERESOURCE;
511          DISKUSE: RESOURCE; CATADDR: INTEGER);
512
513  "THE DISK PARAMETER HAS BEEN ADDED TO ALLOW FOR    - KSU
514        THE USE OF MULTIPLE VIRTUAL DISKS            - KSU"
515
516  VAR TABLE: DISKTABLE;
517
518  FUNCTION HASH(ID: IDENTIFIER): INTEGER;
519
520  "THE VALIDITY OF THE HASH FUNCTION DEPENDS ON  - KSU"
521  "THE FACT THAT ALL VIRTUAL DISKS HAVE THE SAME - KSU"
522  "CATALOG SIZE                                  - KSU"
523  VAR KEY, I: INTEGER; C: CHAR;
524  BEGIN
525    KEY:= 1; I:= 0;
526    REPEAT
527      I:= I + 1; C:= ID[I];
528      IF C <> ' ' THEN
529        KEY:= KEY * ORD(C) MOD TABLE.FLENGTH + 1;
530    UNTIL (C = ' ') OR (I = IDLENGTH);
531    HASH:= KEY;
532  END;
533
534  PROCEDURE ENTRY LOOKUP
535  (ID: IDENTIFIER; VAR ATTR: FILEATTR;
536   VAR FOUND: BOOLEAN; DISK: IODEVICE);
537
538  VAR KEY, MORE, INDEX: INTEGER; ELEM: CATENTRY;
539  BEGIN
540    DISKUSE.REQUEST;
```

```
541    KEY:= HASH(ID);
542    TABLE.READ(KEY, ELEM, DISK);
543    MORE:= ELEM.SEARCHLENGTH;
544    INDEX:= KEY; FOUND:= FALSE;
545    WHILE NOT FOUND & (MORE > 0) DO
546    BEGIN
547      TABLE.READ(INDEX, ELEM, DISK);
548      IF ELEM.ID = ID THEN
549      BEGIN ATTR:= ELEM.ATTR; FOUND:= TRUE END
550      ELSE
551      BEGIN
552        IF ELEM.KEY = KEY THEN MORE:= MORE - 1;
553        INDEX:= INDEX MOD TABLE.FLENGTH + 1;
554      END;
555    END;
556    DISKUSE.RELEASE;
557  END;
558
559  BEGIN INIT TABLE(TYPEUSE, CATADDR) END;
560
561
562  "##############
563  #  DATAFILE   #
564  ##############"
565
566
567  TYPE DATAFILE =
568  CLASS(TYPEUSE: TYPERESOURCE; DISKUSE: RESOURCE;
569        CATALOG: DISKCATALOG);
570
571  VAR FILE: DISKFILE; OPENED: BOOLEAN;
572
573  ENTRY LENGTH: INTEGER;
574
575  PROCEDURE ENTRY OPEN(ID: IDENTIFIER; VAR FOUND: BOOLEAN);
576  VAR ATTR: FILEATTR;
577  BEGIN
578    CATALOG.LOOKUP(ID, ATTR, FOUND, PRIVATEDISK);
579    IF FOUND THEN
580    BEGIN
581      DISKUSE.REQUEST;
582      FILE.OPEN(ATTR.ADDR, PRIVATEDISK);
583      LENGTH:= FILE.LENGTH;
584      DISKUSE.RELEASE;
585    END;
586    OPENED:= FOUND;
587  END;
588
589  PROCEDURE ENTRY CLOSE;
590  BEGIN
591    FILE.CLOSE;
592    LENGTH:= 0;
593    OPENED:= FALSE;
594  END;
```

```
595
596 PROCEDURE ENTRY READ(PAGENO: INTEGER;
597                       VAR BLOCK: UNIV PAGE);
598 BEGIN
599   IF OPENED THEN
600   BEGIN
601     DISKUSE.REQUEST;
602     FILE.READ(PAGENO, BLOCK, PRIVATEDISK);
603     DISKUSE.RELEASE;
604   END;
605 END;
606
607 PROCEDURE ENTRY WRITE(PAGENO: INTEGER;
608                        VAR BLOCK: UNIV PAGE);
609 BEGIN
610   IF OPENED THEN
611   BEGIN
612     DISKUSE.REQUEST;
613     FILE.WRITE(PAGENO, BLOCK, PRIVATEDISK);
614     DISKUSE.RELEASE;
615   END;
616 END;
617
618 BEGIN
619   INIT FILE(TYPEUSE);
620   LENGTH:= 0;
621   OPENED:= FALSE;
622 END;
623
624
625 "############################
626 #  PROGSTORE AND PROGFILE  #
627 ############################"
628
629
630 TYPE PROGSTATE = (READY, NOTFOUND, NOTSEQ, TOOBIG);
631
632 CONST STORELENGTH1 = 40;
633 TYPE PROGSTORE1 =
634   ARRAY [1..STORELENGTH1] OF PAGE;
635
636
637 TYPE PROGFILE1 =
638 CLASS(TYPEUSE: TYPERESOURCE; DISKUSE: RESOURCE;
639       CATALOG: DISKCATALOG);
640
641 "PROGRAM FILES ARE FOUND BY FIRST SEARCHING THE    - KSU"
642 "PRIVATE DISK, IF NOT FOUND THERE, THE SYSTEM DISK - KSU"
643 "IS THEN SEARCHED                                  - KSU"
644
645
646 VAR FILE: DISKFILE;
647
648 ENTRY STORE: PROGSTORE1;
```

```
649
650 PROCEDURE ENTRY OPEN(ID: IDENTIFIER;
651                        VAR STATE: PROGSTATE);
652 VAR ATTR: FILEATTR; FOUND: BOOLEAN; PAGENO: INTEGER;
653     DISK: IODEVICE;
654 BEGIN
655   CATALOG.LOOKUP(ID, ATTR, FOUND, PRIVATEDISK);
656   IF FOUND THEN DISK:= PRIVATEDISK
657   ELSE BEGIN
658     CATALOG.LOOKUP(ID, ATTR, FOUND, SYSTEMDISK);
659     DISK:= SYSTEMDISK;
660   END;
661   WITH DISKUSE, FILE, ATTR DO
662   IF NOT FOUND THEN
663     STATE:= NOTFOUND ELSE
664   IF KIND <> SEQCODE 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: TYPERESOURCE; 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, PROGRESULT, RUNTIME);


TYPE PROGSTACK =
MONITOR

CONST STACKLENGTH = 5;

VAR STACK: ARRAY [1..STACKLENGTH] OF
           RECORD
             PROGID: IDENTIFIER;
```

```
757                HEAPADDR: INTEGER
758              END;
759       TOP: 0..STACKLENGTH;
760
761 FUNCTION ENTRY SPACE: BOOLEAN;
762 BEGIN SPACE:= (TOP < STACKLENGTH) END;
763
764 FUNCTION ENTRY ANY: BOOLEAN;
765 BEGIN ANY:= (TOP > 0) END;
766
767 PROCEDURE ENTRY PUSH(ID: IDENTIFIER);
768 BEGIN
769   IF TOP < STACKLENGTH THEN
770   BEGIN
771     TOP:= TOP + 1;
772     WITH STACK[TOP] DO
773     BEGIN
774       PROGID:= ID;
775       HEAPADDR:= ATTRIBUTE(HEAPTOP);
776     END;
777   END;
778 END;
779
780 PROCEDURE ENTRY POP
781   (VAR LINE, RESULT: UNIV INTEGER);
782 CONST TERMINATED = 0;
783 BEGIN
784   LINE:= ATTRIBUTE(PROGLINE);
785   RESULT:= ATTRIBUTE(PROGRESULT);
786   IF RESULT <> TERMINATED THEN
787     SETHEAP(STACK[TOP].HEAPADDR);
788   TOP:= TOP - 1;
789 END;
790
791 PROCEDURE ENTRY GET(VAR ID: IDENTIFIER);
792 BEGIN
793   IF TOP > 0 THEN ID:= STACK[TOP].PROGID;
794 END;
795
796 BEGIN TOP:= 0 END;
797
798
799 "##########################
800 #  TASKKIND AND ARGTYPE  #
801 ##########################"
802
803
804 TYPE TASKKIND = (INPUTTASK, JOBTASK, OUTPUTTASK);
805
806 TYPE ARGTAG =
807   (NILTYPE, BOOLTYPE, INTTYPE, IDTYPE, PTRTYPE);
808
809      ARGTYPE = RECORD
810                  TAG: ARGTAG;
```

```
811                   ARG: IDENTIFIER
812                 END;
813
814 CONST MAXARG = 10;
815 TYPE ARGLIST = ARRAY [1..MAXARG] OF ARGTYPE;
816
817 TYPE ARGSEQ = (INP, OUT);
818
819
820 "###############
821 #  ARGBUFFER  #
822 ###############"
823
824
825 TYPE ARGBUFFER =
826 MONITOR
827 VAR BUFFER: ARGTYPE; FULL: BOOLEAN;
828     SENDER, RECEIVER: QUEUE;
829
830 PROCEDURE ENTRY READ(VAR ARG: ARGTYPE);
831 BEGIN
832   IF NOT FULL THEN DELAY(RECEIVER);
833   ARG:= BUFFER; FULL:= FALSE;
834   CONTINUE(SENDER);
835 END;
836
837 PROCEDURE ENTRY WRITE(ARG: ARGTYPE);
838 BEGIN
839   IF FULL THEN DELAY(SENDER);
840   BUFFER:= ARG; FULL:= TRUE;
841   CONTINUE(RECEIVER);
842 END;
843
844 BEGIN FULL:= FALSE END;
845
846
847 "################
848 #  LINEBUFFER  #
849 ################"
850
851
852 TYPE LINEBUFFER =
853 MONITOR
854 VAR BUFFER: LINE; FULL: BOOLEAN;
855     SENDER, RECEIVER: QUEUE;
856
857 PROCEDURE ENTRY READ(VAR TEXT: LINE);
858 BEGIN
859   IF NOT FULL THEN DELAY(RECEIVER);
860   TEXT:= BUFFER; FULL:= FALSE;
861   CONTINUE(SENDER);
862 END;
863
864 PROCEDURE ENTRY WRITE(TEXT: LINE);
```

```
865 BEGIN
866   IF FULL THEN DELAY(SENDER);
867   BUFFER:= TEXT; FULL:= TRUE;
868   CONTINUE(RECEIVER);
869 END;
870
871 BEGIN FULL:= FALSE END;
872
873
874 "################
875 #  PAGEBUFFER  #
876 ################"
877
878
879 TYPE PAGEBUFFER =
880 MONITOR
881 VAR BUFFER: PAGE; LAST, FULL: BOOLEAN;
882     SENDER, RECEIVER: QUEUE;
883
884 PROCEDURE ENTRY READ(VAR TEXT: PAGE; VAR EOF: BOOLEAN);
885 BEGIN
886   IF NOT FULL THEN DELAY(RECEIVER);
887   TEXT:= BUFFER; EOF:= LAST; FULL:= FALSE;
888   CONTINUE(SENDER);
889 END;
890
891 PROCEDURE ENTRY WRITE(TEXT: PAGE; EOF: BOOLEAN);
892 BEGIN
893   IF FULL THEN DELAY(SENDER);
894   BUFFER:= TEXT; LAST:= EOF; FULL:= TRUE;
895   CONTINUE(RECEIVER);
896 END;
897
898 BEGIN FULL:= FALSE END;
899
900
901 "################
902 #  CHARSTREAM  #
903 ################"
904
905
906 TYPE CHARSTREAM =
907 CLASS(BUFFER: PAGEBUFFER);
908
909 VAR TEXT: PAGE; COUNT: INTEGER; EOF: BOOLEAN;
910
911
912 PROCEDURE ENTRY READ(VAR C: CHAR);
913 BEGIN
914   IF COUNT = PAGELENGTH THEN
915   BEGIN
916     BUFFER.READ(TEXT, EOF);
917     COUNT:= 0;
918   END;
```

```
919    COUNT:= COUNT + 1;
920    C:= TEXT[COUNT];
921    IF C = EM THEN
922    BEGIN
923      WHILE NOT EOF DO BUFFER.READ(TEXT, EOF);
924      COUNT:= PAGELENGTH;
925    END;
926  END;
927
928  PROCEDURE ENTRY INITREAD;
929  BEGIN COUNT:= PAGELENGTH END;
930
931  PROCEDURE ENTRY WRITE(C: CHAR);
932  BEGIN
933    COUNT:= COUNT + 1;
934    TEXT[COUNT]:= C;
935    IF (COUNT = PAGELENGTH) OR (C = EM) THEN
936    BEGIN
937      BUFFER.WRITE(TEXT, FALSE); COUNT:= 0;
938      IF C = EM THEN BUFFER.WRITE(TEXT, TRUE);
939    END;
940  END;
941
942  PROCEDURE ENTRY INITWRITE;
943  BEGIN COUNT:= 0 END;
944
945  BEGIN END;
946
947  CONST APCNT = 2;
948
949  CONST NOIDCNT = 2;
950        "NUMBER OF SEQUENTIAL PROGRAMS TO BE DELAYED
951         BECAUSE THERE ARE NO AVAILABLE DBIDS."
952
953  "*****************************
954  *  UNIVERSAL KERNEL TYPES   *
955  *****************************"
956
957  TYPE PAGE_BUFFER = ARRAY [1..512] OF CHAR;
958
959  TYPE ERRSET = SET OF 0..127;
960
961  TYPE SUBCHAR = SET OF '(:0:)'..'(:127:)';
962
963  TYPE DISPLAY_MESSAGES = (COMMAND_ERROR,ERROR,ENTER_DEST_ID,
964                  ENTER_SOURCE_ID,ENTER_TEXT,
965                  ENTER_DATA_LENGTH,NI);
966
967  TYPE INPUTLINE = ARRAY [1..80] OF CHAR;
968
969  TYPE PHY_ID = ARRAY [1..4] OF CHAR;
970
971  TYPE LOG_ID = ARRAY [1..12] OF CHAR;
972
```

```
 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              SCB : USER_SCB_RECORD;
 997           MSCONTENTS : PAGE_BUFFER;
 998            HIT, FULL : BOOLEAN;
 999      SENDER,RECEIVER : QUEUE;
1000          I, J, RCODE : INTEGER;
1001        GET_ID_ARRAY : IDARRAY;
1002   HOST_ID, TEMPNAME : PHY_ID;
1003             COMMAND : LOG_ID;
1004
1005 PROCEDURE ENTRY CONNECT(ID_REM : PHY_ID;
1006                         ID_LOC : PHY_ID;
1007                     VAR RCODE : INTEGER);
1008 BEGIN
1009   WITH SCB DO
1010    BEGIN
1011       REMOTE_ID := ID_REM;
1012       LOCAL_ID := ID_LOC;
1013       IN_CODE := 3;
1014       OUT_CODE := 0;
1015       COMMAND := 0;
1016       COMMAND_LEN := 0;
1017       DATA_LEN := 0;
1018       SYSQUE(-2,RCODE,SCB,SCB);
1019       RCODE := OUT_CODE;
1020    END;
1021 END;
1022
1023 PROCEDURE ENTRY DISCONNECT(ID_REM : PHY_ID;
1024                            ID_LOC : PHY_ID;
1025                        VAR RCODE : INTEGER);
1026 BEGIN
```

```
1027    WITH SCB DO
1028     BEGIN
1029        REMOTE_ID := ID_REM;
1030        LOCAL_ID := ID_LOC;
1031        IN_CODE := 4;
1032        COMMAND := 1;
1033        OUT_CODE := 0;
1034        SYSQUE(-2,RCODE,SCB,SCB);
1035        RCODE := OUT_CODE;
1036     END;
1037 END;
1038
1039 PROCEDURE ENTRY SEND(ID_REM : PHY_ID;
1040                      ID_LOC : PHY_ID;
1041                  VAR BUFFER : PAGE_BUFFER;
1042                      LENGTH : INTEGER;
1043                   VAR RCODE : INTEGER);
1044 BEGIN
1045   WITH SCB DO
1046    BEGIN
1047       REMOTE_ID := ID_REM;
1048       LOCAL_ID := ID_LOC;
1049       IN_CODE := 60;
1050       COMMAND := 2;
1051       COMMAND_LEN := 0;
1052       DATA_LEN := LENGTH;
1053       OUT_CODE := 0;
1054       SYSQUE(-2,RCODE,BUFFER,SCB);
1055       RCODE := OUT_CODE;
1056    END;
1057 END;
1058
1059 PROCEDURE ENTRY RECEIVE(ID_REM : PHY_ID;
1060                         ID_LOC : PHY_ID;
1061                     VAR BUFFER : PAGE_BUFFER;
1062                     VAR LENGTH : INTEGER;
1063                      VAR RCODE : INTEGER);
1064 BEGIN
1065   WITH SCB DO
1066    BEGIN
1067       REMOTE_ID := ID_REM;
1068       LOCAL_ID := ID_LOC;
1069       IN_CODE := 63;
1070       COMMAND := 3;
1071       OUT_CODE := 0;
1072       DATA_LEN := LENGTH;
1073       COMMAND_LEN := 128;
1074       SYSQUE(-2,RCODE,BUFFER,SCB);
1075       LENGTH := DATA_LEN;
1076       RCODE := OUT_CODE;
1077    END;
1078 END;
1079
1080 PROCEDURE ENTRY GET_ID (VAR IDNAME:PHY_ID );
```

```
1081 BEGIN
1082   I:= I+1;
1083   GET_ID_ARRAY(.I.):=TEMPNAME;
1084   IDNAME:=TEMPNAME;
1085   TEMPNAME(.4.):=CHR(ORD(TEMPNAME(.4.))+1);
1086           "CMTP:=CMTP+1"
1087 END;
1088
1089 PROCEDURE ENTRY PURGE (PURGENAME:PHY_ID);
1090 BEGIN
1091   J:= 0;
1092   REPEAT
1093       J:=J+1;
1094       IF PURGENAME = GET_ID_ARRAY(.J.)
1095          THEN HIT:=TRUE;
1096     UNTIL HIT;
1097   IF J < I
1098      THEN REPEAT
1099                GET_ID_ARRAY(.J.):=GET_ID_ARRAY(.J+1.);
1100                J:=J+1;
1101             UNTIL J=I;
1102   I:= I-1;
1103 END;
1104
1105 BEGIN
1106   FULL:= FALSE;
1107   HOST_ID:= '    ';
1108   FOR J:= 1 TO 4 DO
1109     GET_ID_ARRAY(.J.):= '    ';
1110   I:= 0;
1111   SYSQUE(0,RCODE,HOST_ID,COMMAND);
1112   TEMPNAME:=HOST_ID;
1113 END;
1114
1115
1116 TYPE ENV_FUNCTION   = ARRAY (.1..6.) OF CHAR;
1117
1118 TYPE ENV_STAT       = ARRAY (.1..4.) OF CHAR;
1119
1120 TYPE ENV_FILE_NAME  = ARRAY (.1..4.) OF CHAR;
1121
1122 TYPE ENV_REFER      = ARRAY (.1..4.) OF CHAR;
1123
1124 TYPE ENV_LINKPATH   = ARRAY (.1..8.) OF CHAR;
1125
1126 TYPE ENV_CTRL_FIELD = ARRAY (.1..10.) OF CHAR;
1127
1128 TYPE ENV_VELEMENTS  = ARRAY (.1..60.) OF CHAR;
1129
1130 TYPE ENV_VAREA      = ARRAY (.1..82.) OF CHAR;
1131
1132 TYPE ENV_ENDP       = ARRAY (.1..4.) OF CHAR;
1133
1134 TYPE BOOLARRAY      = ARRAY (.1..APCNT.) OF BOOLEAN;
```

```
1135
1136 TYPE SPQUEUE         = ARRAY (.1..NOIDCNT.) OF QUEUE;
1137
1138 TYPE DBCMDS          = ARRAY (.1..38.) OF CHAR;
1139
1140 TYPE WORK_BUFF =
1141        RECORD
1142             BFUNCTION : ENV_FUNCTION;
1143                 BSTAT : ENV_STAT;
1144            BFILE_NAME : ENV_FILE_NAME;
1145                BREFER : ENV_REFER;
1146             BLINKPATH : ENV_LINKPATH;
1147           BCTRL_FIELD : ENV_CTRL_FIELD;
1148            BVELEMENTS : ENV_VELEMENTS;
1149                BVAREA : ENV_VAREA;
1150                 BENDP : ENV_ENDP;
1151                  APID : PHY_ID;
1152                  DBID : PHY_ID;
1153                   ACK : BOOLEAN;
1154                  SPID : INTEGER;
1155                FILLER : ARRAY (.1..318.) OF CHAR
1156        END;
1157
1158 TYPE ID_PAIR = ARRAY (.1..APCNT.) OF
1159                   RECORD
1160                     APID : PHY_ID;
1161                     DBID : PHY_ID
1162                   END;
1163
1164 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
1172 TYPE BUFFARRAY   = ARRAY (.1..APCNT.) OF BEBUFF;
1173
1174 TYPE ERRMESSAGE  = ARRAY (.1..24.) OF CHAR;
1175
1176 TYPE BEQUEUE     = ARRAY (.1..APCNT.) OF QUEUE;
1177
1178
1179 "******************
1180 *  HINT MONITOR  *
1181 ******************"
1182
1183 TYPE HINTMON = MONITOR
1184 VAR                 I, CNT : INTEGER;
1185             SPBOOL,RTBOOL : BOOLARRAY;
1186                     AP,SP : SPQUEUE;
1187                    SPNEXT : FIFO;
1188                FULL,RFULL : BOOLEAN;
```

```
1189     APSENDER,FRPRECEIVER : QUEUE;
1190             BUFF,RETBUFF : WORK_BUFF;
1191                 RESPONSE : QUEUE;
1192
1193 PROCEDURE ENTRY SEND (SPNUM : INTEGER;
1194                     AP_BUFF : WORK_BUFF);
1195 BEGIN
1196   IF (CNT > APCNT) "NO ROOM FOR MORE APS"
1197      AND NOT SPBOOL (.SPNUM.) "THE AP HAS NOT BEEN"
1198                               "SERVICED"
1199      THEN DELAY (SP(.SPNEXT.ARRIVAL.));
1200   IF FULL
1201      THEN DELAY (APSENDER);
1202   IF NOT SPBOOL (.SPNUM.)
1203      THEN BEGIN "FIRST TIME FOR THE AP REQUEST"
1204             SPBOOL (.SPNUM.) := TRUE;
1205             CNT := CNT + 1;
1206           END;
1207   BUFF:=AP_BUFF;
1208   BUFF.SPID := SPNUM;
1209   BUFF.ACK:=FALSE;
1210   FULL := TRUE;
1211   CONTINUE (FRPRECEIVER);
1212 END;
1213
1214 PROCEDURE ENTRY RECEIVE (VAR HINT_BUFF : WORK_BUFF);
1215 BEGIN
1216   IF NOT FULL
1217      THEN DELAY (FRPRECEIVER);
1218   HINT_BUFF := BUFF;
1219   FULL := FALSE;
1220   CONTINUE (APSENDER);
1221 END;
1222
1223 PROCEDURE ENTRY APRETURN (SPNUM : INTEGER;
1224                       VAR HR_BUFF : WORK_BUFF);
1225 BEGIN
1226   IF NOT RTBOOL (.SPNUM.) "DELAY AN AP IN THE PREFIX"
1227      THEN DELAY (AP(.SPNUM.)); "UNTIL A RSP"
1228   HR_BUFF := RETBUFF;
1229   RTBOOL (.SPNUM.) := FALSE;
1230   IF HR_BUFF.ACK = TRUE
1231      THEN BEGIN
1232             CONTINUE (SP(.SPNEXT.DEPARTURE.));
1233             "CONTINUE AN AP WAITING AT THE"
1234             "PREFIX BECAUSE THERE WAS NO ROOM"
1235             SPBOOL (.SPNUM.) := FALSE;
1236             CNT := CNT - 1;
1237           END;
1238   RFULL := FALSE;
1239   CONTINUE (RESPONSE);
1240 END;
1241
1242 PROCEDURE ENTRY RSPRETURN (RSPBUFF : WORK_BUFF);
```

```
1243 BEGIN
1244   IF FULL
1245      THEN DELAY (RESPONSE);
1246   RETBUFF := RSPBUFF;
1247   RTBOOL (.RETBUFF.SPID.) := TRUE;
1248   RFULL := TRUE;
1249   CONTINUE (AP(.RETBUFF.SPID.));
1250 END;
1251
1252 BEGIN CNT := 1;
1253       FULL := FALSE;
1254       RFULL := FALSE;
1255       FOR I:= 1 TO APCNT DO
1256         BEGIN
1257           SPBOOL(.I.):=FALSE;
1258           SPBOOL(.I.):=FALSE;
1259         END;
1260       INIT SPNEXT (NOIDCNT);
1261 END;
1262
1263
1264 "**********************
1265 *  FORWARD MONITOR   *
1266 **********************"
1267
1268 TYPE FOR_REQ_MON = MONITOR
1269 VAR            FRCONTENTS : WORK_BUFF;
1270                       FULL : BOOLEAN;
1271     NRCRECEIVER,FRSENDER : QUEUE;
1272
1273 PROCEDURE ENTRY SEND (HINT_BUFF : WORK_BUFF);
1274 BEGIN
1275   IF FULL
1276      THEN DELAY (FRSENDER);
1277   FRCONTENTS := HINT_BUFF;
1278   FULL := TRUE;
1279   CONTINUE (NRCRECEIVER);
1280 END;
1281
1282 PROCEDURE ENTRY RECEIVE (VAR RECVBUFF : WORK_BUFF);
1283 BEGIN
1284   IF NOT FULL
1285      THEN DELAY (NRCRECEIVER);
1286   RECVBUFF := FRCONTENTS;
1287   FULL := FALSE;
1288   CONTINUE (FRSENDER);
1289 END;
1290
1291 BEGIN
1292   FULL := FALSE;
1293 END;
1294
1295 "**********************
1296 *  FORWARD PROCESS   *
```

```
1297 **********************"
1298
1299 TYPE FORWARD_REQUEST = PROCESS (FEHINT : HINTMON;
1300                                   FRMON : FOR_REQ_MON;
1301                                TYPEUSE : TYPERESOURCE;
1302                                    MS  : MESSAGE_SYSTEM);
1303 VAR  FOR_REQ_BUFF : WORK_BUFF;
1304     ID_REM,ID_LOC : PHY_ID;
1305            BUFFER : PAGE_BUFFER;
1306      LENGTH,RCODE : INTEGER;
1307                 I : INTEGER;
1308               CRT : TERMINAL;
1309            CRTMSG : TERMINALSTREAM;
1310                 C : CHAR;
1311
1312 PROCEDURE WORK_BUFF_CONVERT(SENDBUFF:UNIV PAGE_BUFFER;
1313                             VAR RETBUFF:PAGE_BUFFER);
1314 BEGIN
1315   RETBUFF:=SENDBUFF;
1316 END;
1317
1318 BEGIN
1319   INIT
1320      CRT (TYPEUSE), CRTMSG (CRT);
1321   LENGTH := 512;
1322   RCODE := 19;  "NO EVENTS QUEUED"
1323   CYCLE
1324     FEHINT.RECEIVE (FOR_REQ_BUFF);
1325     CRT.WRITE('FORWARD_REQUEST_PROCESS: (:10:)',
1326               'RECEIVED FROM HINTMON. (:10:)');
1327         IF (FOR_REQ_BUFF.BFUNCTION = ' SINON') OR
1328            (FOR_REQ_BUFF.BFUNCTION = ' SINOF')
1329            THEN FRMON.SEND (FOR_REQ_BUFF)
1330            ELSE BEGIN
1331                   ID_REM := FOR_REQ_BUFF.DBID;
1332                   ID_LOC := FOR_REQ_BUFF.APID;
1333                   WORK_BUFF_CONVERT(FOR_REQ_BUFF, BUFFER);
1334                   CRT.WRITE
1335                       ('FORWARD_REQUEST_PROCESS: (:10:)',
1336                        'SENT TO MESSAGE SYSTEM.(:10:)');
1337                   MS.SEND (ID_REM,
1338                            ID_LOC,
1339                            BUFFER,
1340                            LENGTH,
1341                            RCODE);
1342                 "CHECK RCODE"
1343                 END;
1344   END;
1345 END;
1346
1347 "**********************
1348 *  ID TABLE MONITOR  *
1349 **********************"
1350
```

```
1351 TYPE ID_TABLE_MON = MONITOR (TYPEUSE : TYPERESOURCE);
1352 VAR             HIT : BOOLEAN;
1353             IDTABLE : ID_PAIR;
1354         I,J,K,IDNUM : INTEGER;
1355                   C : CHAR;
1356                 CRT : TERMINAL;
1357            CRTERROR : TERMINALSTREAM;
1358
1359 PROCEDURE ENTRY PUT (BE_APID,BE_DBID : PHY_ID);
1360 BEGIN
1361   I := I + 1;
1362   WITH IDTABLE(.I.) DO
1363     BEGIN
1364       APID := BE_APID;
1365       DBID := BE_DBID;
1366     END;
1367 END;
1368
1369 PROCEDURE ENTRY REMOVE (BE_APID,BE_DBID : PHY_ID);
1370 BEGIN
1371   J := 0;
1372   REPEAT
1373       J := J + 1;
1374       IF IDTABLE(.J.).APID = BE_APID
1375          THEN HIT:= TRUE;
1376     UNTIL (HIT) OR (J = I);
1377   IF HIT
1378      THEN BEGIN
1379             IF J=I
1380                THEN BEGIN
1381                       IDTABLE(.J.).APID:= '    ';
1382                       IDTABLE(.J.).DBID:= '    ';
1383                     END
1384                ELSE BEGIN
1385                       FOR K:= J TO I DO
1386                         BEGIN
1387                           IDTABLE(.K.).APID :=
1388                             IDTABLE(.K+1.).APID;
1389                           IDTABLE(.K.).DBID :=
1390                             IDTABLE(.K+1.).DBID;
1391                         END;
1392                     END;
1393             I := I - 1;
1394             HIT:= FALSE;
1395           END
1396      ELSE CRT.WRITE('ID_TABLE_MON:(:10:)',
1397                     'REMOVE ERROR IN ID_TABLE.(:10:)');
1398 END;
1399
1400 PROCEDURE ENTRY IDREAD (VAR IDNUM : INTEGER;
1401                         VAR REQIDS : ID_PAIR);
1402 BEGIN
1403   IDNUM := I;
1404   REQIDS := IDTABLE;
```

```
1405 END;
1406
1407 BEGIN
1408   HIT:=FALSE;
1409   FOR I := 1 TO APCNT DO
1410     BEGIN
1411       WITH IDTABLE(.I.) DO
1412         BEGIN
1413           APID := '    ';
1414           DBID := '    ';
1415         END;
1416     END;
1417   I := 0;
1418   J := 0;
1419   K := 0;
1420   IDNUM := 0;
1421   INIT
1422     CRT (TYPEUSE), CRTERROR (CRT);
1423 END;
1424
1425 "*****************************
1426 * NRC DIRECTORY MONITOR     *
1427 *****************************"
1428
1429 TYPE DIRECTORY = MONITOR
1430 VAR FRONTNRC : PHY_ID;
1431     BACKNRC  : PHY_ID;
1432
1433 PROCEDURE ENTRY RECEIVE (VAR FENRC : PHY_ID;
1434                          VAR BENRC : PHY_ID);
1435 BEGIN
1436   FENRC := FRONTNRC;
1437   BENRC := BACKNRC;
1438 END;
1439
1440 BEGIN
1441   FRONTNRC := 'ABMF';
1442   BACKNRC := 'ABAB';
1443 END;
1444
1445 "***********************
1446 *  FRONT_END CONNECT *
1447 ***********************"
1448
1449 TYPE FE_CONN_PROCESS =
1450      PROCESS (NRCDIRECTORY : DIRECTORY;
1451                         MS : MESSAGE_SYSTEM;
1452                   TYPEUSE : TYPERESOURCE);
1453 VAR ID_LOC,ID_REM : PHY_ID;
1454          RCODE, I : INTEGER;
1455          CONN_OK : BOOLEAN;
1456              CRT : TERMINAL;
1457               OK : OKTYPE;
1458
```

```
1459 BEGIN
1460   RCODE:= 19; "NO EVENTS QUEUED"
1461   I:= 0;
1462   OK:= 0;
1463   CONN_OK:= FALSE;
1464   INIT
1465     CRT(TYPEUSE);
1466   NRCDIRECTORY.RECEIVE(ID_LOC,ID_REM);
1467   REPEAT
1468       MS.CONNECT(ID_LOC,
1469                  ID_REM,
1470                  RCODE);
1471       IF RCODE=OK
1472          THEN CONN_OK:= TRUE
1473          ELSE I:= I+1;
1474     UNTIL (CONN_OK) OR (I=5);
1475   IF NOT CONN_OK
1476      THEN CRT.WRITE
1477               ('FE_CONN_PROCESS: (:10:)',
1478                'FE COULD NOT CONNECT WITH BE.(:10:)')
1479      ELSE IF I=5 THEN CRT.WRITE
1480                           ('FE_CONN_PROCESS: (:10:)',
1481                            'CONNECT TIME OUT.(:10:)')
1482                  ELSE CRT.WRITE
1483                           ('FE_CONN_PROCESS: (:10:)',
1484                            'FE CONNECTED WITH BE.(:10:)');
1485 END;
1486
1487 "*********************
1488 * BACK_END CONNECT *
1489 *********************"
1490
1491 TYPE BE_CONN_PROCESS =
1492      PROCESS (NRCDIRECTORY : DIRECTORY;
1493                         MS : MESSAGE_SYSTEM;
1494                    TYPEUSE : TYPERESOURCE);
1495 VAR ID_LOC,ID_REM : PHY_ID;
1496          RCODE, I : INTEGER;
1497           CONN_OK : BOOLEAN;
1498               CRT : TERMINAL;
1499                OK : OKTYPE;
1500
1501 BEGIN
1502   RCODE:= 19; "NO EVENTS QUEUED"
1503   I:= 0;
1504   CONN_OK:= FALSE;
1505   OK:= 0;
1506   INIT
1507     CRT(TYPEUSE);
1508   NRCDIRECTORY.RECEIVE(ID_REM,ID_LOC);
1509   REPEAT
1510       MS.CONNECT(ID_LOC,
1511                  ID_REM,
1512                  RCODE);
```

```
1513        IF RCODE=OK
1514           THEN CONN_OK:= TRUE
1515           ELSE I:= I+1;
1516      UNTIL (CONN_OK) OR (I=5);
1517    IF NOT CONN_OK
1518       THEN CRT.WRITE
1519                ('BE_CONN_PROCESS: (:10:)',
1520                 'BE COULD NOT CONNECT WITH FE.(:10:)')
1521       ELSE IF I=5 THEN CRT.WRITE
1522                                ('BE_CONN_PROCESS: (:10:)',
1523                                 'CONNECT TIME OUT.(:10:)')
1524                    ELSE CRT.WRITE
1525                                ('BE_CONN_PROCESS: (:10:)',
1526                                 'BE CONNECTED WITH FE.(:10:)');
1527 END;
1528 "********************
1529 *  FRONT-END NRC  *
1530 ********************"
1531
1532 TYPE FE_NRC = PROCESS (FRMON : FOR_REQ_MON;
1533                        FEHINT : HINTMON;
1534                          FIDT : ID_TABLE_MON;
1535                   NRCDIRECTORY : DIRECTORY ;
1536                       TYPEUSE : TYPERESOURCE;
1537                            MS  : MESSAGE_SYSTEM);
1538 VAR  RETWORKBUFF : WORK_BUFF;
1539      FEBUFF : WORK_BUFF;
1540      BUFFER : PAGE_BUFFER;
1541     TMP_REM : PHY_ID;
1542      ID_REM : PHY_ID;
1543     TMP_LOC : PHY_ID;
1544      ID_LOC : PHY_ID;
1545        TACK : BOOLEAN;
1546      LENGTH : INTEGER;
1547     I,RCODE : INTEGER;
1548          OK : OKTYPE;
1549        NAME : PHY_ID;
1550       FENRC : PHY_ID;
1551       BENRC : PHY_ID;
1552           C : CHAR;
1553         CRT : TERMINAL;
1554    CRTERROR : TERMINALSTREAM;
1555
1556 PROCEDURE WORK_BUFF_CONVERT(SENDBUFF:UNIV PAGE_BUFFER;
1557                             VAR RETBUFF:PAGE_BUFFER);
1558 BEGIN
1559   RETBUFF:=SENDBUFF;
1560 END;
1561
1562 PROCEDURE PAGE_BUFF_CONVERT (BUFF : UNIV WORK_BUFF;
1563                              VAR RETWORK : WORK_BUFF);
1564 BEGIN
1565   RETWORK := BUFF;
1566 END;
```

```
1567
1568 BEGIN
1569   INIT
1570     CRT(TYPEUSE), CRTERROR(CRT);
1571   OK := 0;
1572   TACK:=FALSE;
1573   LENGTH:=512;
1574   I:=0;
1575   RCODE:=19; "NO EVENTS QUEUED"
1576   CYCLE
1577     FRMON.RECEIVE (FEBUFF);
1578     CRT.WRITE('FE_NRC PROCESS:(:10:)',
1579               'RECEIVED FROM FOR_REQ_MON(:10:)');
1580     IF FEBUFF.BFUNCTION = ' SINON'
1581        THEN BEGIN
1582               "CALL DIRECTORY FOR
1583                FE AND BE NRCS"
1584               NRCDIRECTORY.RECEIVE(FENRC,BENRC);
1585               ID_LOC := FENRC;
1586               ID_REM := BENRC;
1587               MS.GET_ID (NAME);
1588               FEBUFF.APID := NAME;
1589               WORK_BUFF_CONVERT(FEBUFF,BUFFER);
1590               MS.SEND (ID_REM,
1591                        ID_LOC,
1592                        BUFFER,
1593                        LENGTH,
1594                         RCODE);
1595               "CHECK RCODE"
1596               MS.RECEIVE (ID_REM,
1597                           ID_LOC,
1598                           BUFFER,
1599                           LENGTH,
1600                           RCODE);
1601               "CHECK RCODE"
1602               TMP_REM := ID_REM;
1603               TMP_LOC := ID_LOC;
1604               PAGE_BUFF_CONVERT(BUFFER,RETWORKBUFF);
1605               ID_REM := RETWORKBUFF.DBID;
1606               ID_LOC := RETWORKBUFF.APID;
1607               MS.CONNECT (ID_REM,
1608                           ID_LOC,
1609                           RCODE);
1610               "CHECK RCODE"
1611               FIDT.PUT (ID_LOC,ID_REM);
1612               FEBUFF.APID := ID_LOC;
1613               FEBUFF.DBID := ID_REM;
1614               CRT.WRITE('FE_NRC PROCESS:(:10:)',
1615                         'FE SINON COMMAND.(:10:)');
1616            END
1617        ELSE IF FEBUFF.BFUNCTION = ' SINOF'
1618                 THEN BEGIN
1619               WORK_BUFF_CONVERT(FEBUFF,BUFFER);
1620               "CALL DIRECTORY FOR
```

```
1621                FE AND BE NRCS"
1622              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_CONVERT(BUFFER,RETWORKBUFF);
1634              ID_REM := RETWORKBUFF.DBID;
1635              ID_LOC := RETWORKBUFF.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_CONVERT(BUFFER,RETWORKBUFF);
1652              RETWORKBUFF.ACK := TACK;
1653              FEBUFF := RETWORKBUFF;
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
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;
1676           RSPIDS : ID_PAIR;
1677          RSPBUFF : WORK_BUFF;
1678           ID_REM : PHY_ID;
1679           ID_LOC : PHY_ID;
1680                C : CHAR;
1681              CRT : TERMINAL;
1682         CRTERROR : TERMINALSTREAM;
1683           BUFFER : PAGE_BUFFER;
1684           LENGTH : INTEGER;
1685            RCODE : INTEGER;
1686               OK : OKTYPE;
1687         NOT_DONE : NOT_DONE_TYPE;
1688
1689 PROCEDURE WORK_BUFF_CONVERT(SENDBUFF:UNIV PAGE_BUFFER;
1690                             VAR RETBUFF:PAGE_BUFFER);
1691 BEGIN
1692   RETBUFF:=SENDBUFF;
1693 END;
1694
1695 PROCEDURE PAGE_BUFF_CONVERT(BUFF : UNIV WORK_BUFF;
1696                                VAR RETWORK : WORK_BUFF);
1697 BEGIN
1698   RETWORK:= BUFF;
1699 END;
1700
1701 BEGIN
1702   INIT
1703     CRT (TYPEUSE), CRTERROR (CRT);
1704   MSG := FALSE;
1705   OK := 0;
1706   NOT_DONE := 28;
1707   I:=0;
1708   J:=0;
1709   IDNUM:=0;
1710   LENGTH:=512;
1711   RCODE:=19; "NO EVENTS QUEUED"
1712   CYCLE
1713     IDNUM := 0;
1714     REPEAT
1715         FIDT.IDREAD (IDNUM,RSPIDS);
1716       UNTIL IDNUM > 0;
1717     CRT.WRITE('MS_RESPONSE PROCESS: (:10:)',
1718               'RECEIVED FROM FE ID_TABLE_MON. (:10:)');
1719       I := 1;
1720       REPEAT
1721           WITH RSPIDS (.I.) DO
1722             BEGIN
1723               ID_LOC := APID;
1724               ID_REM := DBID;
1725             END;
1726           FOR J:= 1 TO 5 DO
1727             WAIT;
1728           MS.RECEIVE (ID_REM,
```

```
1729                        ID_LOC,
1730                        BUFFER,
1731                        LENGTH,
1732                        RCODE);
1733           IF RCODE = OK
1734              THEN BEGIN
1735                     MSG:=TRUE;
1736                     CRT.WRITE
1737                         ('MS_RESPONSE PROCESS: (:10:)',
1738                          'RECEIVED FROM MESSAGE',
1739                             'SYSTEM.(:10:)');
1740                   END
1741              ELSE IF RCODE = NOT_DONE
1742                      THEN I := I + 1
1743                      ELSE CRT.WRITE
1744                               ('MS_RESPONSE',
1745                                   'PROCESS: (:10:)',
1746                                'MESSAGE SYSTEM RECEIVE',
1747                                   'ERROR.(:10:)');
1748           WAIT;
1749         UNTIL (I=IDNUM) OR (MSG);
1750     IF MSG
1751        THEN BEGIN
1752               PAGE_BUFF_CONVERT(BUFFER,RSPBUFF);
1753               FEHINT.RSPRETURN (RSPBUFF);
1754               MSG:= FALSE;
1755             END;
1756   END;
1757 END;
1758
1759 "******************
1760 *  BACK-END NRC  *
1761 ******************"
1762
1763 TYPE BE_NRC = PROCESS (BIDT : ID_TABLE_MON;
1764                 NRCDIRECTORY : DIRECTORY;
1765                      TYPEUSE : TYPERESOURCE;
1766                         MS   : MESSAGE_SYSTEM);
1767 VAR            BE_NRC_BUFF : WORK_BUFF;
1768     TMP_ID_REM,TMP_ID_LOC : PHY_ID;
1769                  MSG,TACK : BOOLEAN;
1770              ID_REM,ID_LOC : PHY_ID;
1771                     BUFFER : PAGE_BUFFER;
1772            I,LENGTH,RCODE : INTEGER;
1773                         OK : OKTYPE;
1774                       NAME : PHY_ID;
1775              FENRC, BENRC : PHY_ID;
1776                          C : CHAR;
1777                        CRT : TERMINAL;
1778                   CRTERROR : TERMINALSTREAM;
1779
1780 PROCEDURE WORK_BUFF_CONVERT(SENDBUFF:UNIV PAGE_BUFFER;
1781                             VAR RETBUFF:PAGE_BUFFER);
1782 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 BFUNCTION = ' 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,
```

```
1837                                    RCODE);
1838                        "CHECK RCODE"
1839                        BIDT.PUT(APID,DBID);
1840                        CRT.WRITE('BE_NRC PROCESS:(:10:)',
1841                                  'BE SINON COMMAND.(:10:)');
1842                    END
1843                ELSE IF BFUNCTION = ' SINOF'
1844                        THEN BEGIN
1845                                ID_REM := APID;
1846                                ID_LOC := DBID;
1847                                MS.DISCONNECT (ID_REM,
1848                                               ID_LOC,
1849                                               RCODE);
1850                                IF RCODE = OK
1851                                   THEN TACK := TRUE
1852                                   ELSE TACK := FALSE;
1853                                BIDT.REMOVE (APID, DBID);
1854                                MS.PURGE (DBID);
1855                                IF RCODE <> OK
1856                                   THEN TACK := FALSE;
1857                                ID_REM := TMP_ID_REM;
1858                                ID_LOC := TMP_ID_LOC;
1859                                ACK := TACK;
1860                                WORK_BUFF_CONVERT(BE_NRC_BUFF,
1861                                                  BUFFER);
1862                                MS.SEND (ID_REM,
1863                                         ID_LOC,
1864                                         BUFFER,
1865                                         LENGTH,
1866                                         RCODE);
1867                                "CHECK RCODE"
1868                                CRT.WRITE
1869                                    ('BE_NRC PROCESS:(:10:)',
1870                                     'BE SINOF COMMAND.(:10:)');
1871                            END
1872                        ELSE CRT.WRITE
1873                                 ('BE_NRC PROCESS:(:10:)',
1874                                  'RECEIVE ERROR IN',
1875                                    'BE_NRC(:10:)');
1876          END;
1877       MSG:=FALSE;
1878       TACK:= FALSE;
1879     END;
1880 END;
1881
1882
1883 "******************
1884 *   BINT MONITOR   *
1885 ******************"
1886
1887 TYPE BINTMON = MONITOR
1888 VAR    BUSY,FULL : BOOLEAN;
1889                Q : BEQUEUE;
1890       NEXT,BNEXT : FIFO;
```

```
1891         CONTENTS : BUFFARRAY;
1892
1893 PROCEDURE ENTRY SEND (BERB : BEBUFF);
1894 BEGIN
1895   CONTENTS (.BNEXT.ARRIVAL.) := BERB;
1896   FULL := TRUE;
1897   CONTINUE (Q(.NEXT.DEPARTURE.));
1898 END;
1899
1900 PROCEDURE ENTRY RECEIVE (VAR DAT_BUFF : BEBUFF);
1901 BEGIN
1902   IF NOT FULL
1903      THEN DELAY (Q(.NEXT.ARRIVAL.));
1904   DAT_BUFF := CONTENTS (.BNEXT.DEPARTURE.);
1905   IF BNEXT.EMPTY
1906      THEN FULL := FALSE;
1907 END;
1908
1909 BEGIN
1910   FULL := FALSE; BUSY := FALSE;
1911   INIT NEXT (APCNT), BNEXT (APCNT);
1912 END;
1913
1914 "**********************
1915 *  REQUEST PROCESS   *
1916 **********************"
1917
1918 TYPE MS_REQUEST = PROCESS (BEBINT : BINTMON;
1919                            BIDT : ID_TABLE_MON;
1920                               MS   : MESSAGE_SYSTEM;
1921                           TYPEUSE : TYPERESOURCE);
1922 VAR    I,J,IDNUM : INTEGER;
1923               MSG : BOOLEAN;
1924           REQIDS : ID_PAIR;
1925           DBBUFF : BEBUFF;
1926           ID_REM : PHY_ID;
1927           ID_LOC : PHY_ID;
1928                C : CHAR;
1929              CRT : TERMINAL;
1930         CRTERROR : TERMINALSTREAM;
1931               OK : OKTYPE;
1932         NOT_DONE : NOT_DONE_TYPE;
1933           BUFFER : PAGE_BUFFER;
1934    LENGTH,RCODE : INTEGER;
1935
1936 BEGIN
1937   INIT
1938     CRT (TYPEUSE), CRTERROR (CRT);
1939   MSG := FALSE;
1940   OK := 0;
1941   NOT_DONE := 28;
1942   I:=0;
1943   J:=0;
1944   IDNUM:=0;
```

```
1945    LENGTH:= 512;
1946    CYCLE
1947      IDNUM := 0;
1948      REPEAT
1949          BIDT.IDREAD (IDNUM,REQIDS);
1950        UNTIL IDNUM > 0;
1951      CRT.WRITE('MS_REQUEST PROCESS:(:10:)',
1952                'RECEIVED FROM BE ID_TABLE_MON. (:10:)');
1953        I := 1;
1954        REPEAT
1955            WITH REQIDS (.I.) DO
1956              BEGIN
1957                ID_REM := APID;
1958                ID_LOC := DBID;
1959              END;
1960            FOR J:= 1 TO 5 DO
1961              WAIT;
1962            MS.RECEIVE (ID_REM,
1963                        ID_LOC,
1964                        BUFFER,
1965                        LENGTH,
1966                        RCODE);
1967            IF RCODE = OK
1968               THEN BEGIN
1969                      MSG:=TRUE;
1970                      CRT.WRITE
1971                          ('MS_REQUEST PROCESS:(:10:)',
1972                           'RECEIVED FROM MESSAGE',
1973                              'SYSTEM.(:10:)');
1974                    END
1975               ELSE IF RCODE = NOT_DONE
1976                       THEN I := I + 1
1977                       ELSE CRT.WRITE
1978                                ('MS_REQUEST PROCESS:(:10:)',
1979                                 'MESSAGE SYSTEM RECEIVE',
1980                                    'ERROR.(:10:)');
1981          WAIT;
1982        UNTIL (I = IDNUM) OR (MSG);
1983        IF MSG
1984           THEN
1985             BEGIN
1986               WITH DBBUFF DO
1987                 BEGIN
1988                   BE_ID_REM := ID_REM;
1989                   BE_ID_LOC := ID_LOC;
1990                   BE_BUFFER := BUFFER;
1991                   BE_LENGTH := LENGTH;
1992                   BE_RCODE := RCODE;
1993                 END;
1994               BEBINT.SEND (DBBUFF);
1995               MSG:= FALSE;
1996             END;
1997  END;
1998 END;
```

```
1999
2000 "****************************
2001 *  CALL 'DATBAS' 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,DBELEM,DBAREA,DBRSP : DBCMDS;
2017                   C : CHAR;
2018                 CRT : TERMINAL;
2019              CRTMSG : 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 RETWORK : WORK_BUFF);
2028 BEGIN
2029   RETWORK:=BUFF;
2030 END;
2031
2032 BEGIN
2033   INIT
2034     CRT (TYPEUSE), CRTMSG (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   DBELEM := ' 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     BEBINT.RECEIVE (DAT_BUFF);
2051     CRT.WRITE('CALL_DATABASE PROCESS: (:10:)',
2052               'RECEIVED FROM BINTMON. (:10:)');
```

```
2053      WITH DAT_BUFF DO
2054        BEGIN
2055          PAGE_BUFF_CONVERT(BE_BUFFER,DBCMD_BUFF);
2056          WITH DBCMD_BUFF DO
2057            BEGIN
2058              "CALL 'DATBAS' USING
2059                 BFUNCTION BSTAT BFILE_NAME
2060                 BREFER BLINKPATH BCTRL_FIELD
2061                 BVELEMENTS BVAREA;"
2062              FOR I := 1 TO 38 DO
2063                DBLINE (.I.) := DBFUNC (.I.);
2064              FOR I := 1 TO 6 DO
2065                DBLINE (.I + 38.) := BFUNCTION (.I.);
2066              FOR I := 45 TO 130 DO
2067                DBLINE (.I.) := ' ';
2068              DB_BUFFER.WRITE (DBLINE);
2069              BSTAT:= '****';
2070              FOR I := 1 TO 38 DO
2071                DBLINE (.I.) := DBSTAT (.I.);
2072              FOR I := 1 TO 4 DO
2073                DBLINE (.I + 38.) := BSTAT (.I.);
2074              FOR I := 43 TO 130 DO
2075                DBLINE (.I.) := ' ';
2076              DB_BUFFER.WRITE (DBLINE);
2077              FOR I := 1 TO 38 DO
2078                DBLINE (.I.) := DBFNAM (.I.);
2079              FOR I := 1 TO 4 DO
2080                DBLINE (.I + 38.) := BFILE_NAME (.I.);
2081              FOR I := 43 TO 130 DO
2082                DBLINE (.I.) := ' ';
2083              DB_BUFFER.WRITE (DBLINE);
2084              FOR I := 1 TO 38 DO
2085                DBLINE (.I.) := DBREF (.I.);
2086              FOR I := 1 TO 4 DO
2087                DBLINE (.I + 38.) := BREFER (.I.);
2088              FOR I := 43 TO 130 DO
2089                DBLINE (.I.) := ' ';
2090              DB_BUFFER.WRITE (DBLINE);
2091              FOR I := 1 TO 38 DO
2092                DBLINE (.I.) := DBLNK (.I.);
2093              FOR I := 1 TO 8 DO
2094                DBLINE (.I + 38.) := BLINKPATH (.I.);
2095              FOR I := 47 TO 130 DO
2096                DBLINE (.I.) := ' ';
2097              DB_BUFFER.WRITE (DBLINE);
2098              FOR I := 1 TO 38 DO
2099                DBLINE (.I.) := DBCTR (.I.);
2100              FOR I := 1 TO 10 DO
2101                DBLINE (.I + 38.) := BCTRL_FIELD (.I.);
2102              FOR I := 49 TO 130 DO
2103                DBLINE (.I.) := ' ';
2104              DB_BUFFER.WRITE (DBLINE);
2105              FOR I := 1 TO 38 DO
2106                DBLINE (.I.) := DBELEM (.I.);
```

```
2107              FOR I := 1 TO 60 DO
2108                DBLINE (.I + 38.) := BVELEMENTS (.I.);
2109              FOR I := 99 TO 130 DO
2110                DBLINE (.I.) := ' ';
2111              DB_BUFFER.WRITE (DBLINE);
2112              FOR I:= 1 TO 36 DO
2113                BVAREA(.I.):=DBRSP(.I.);
2114              FOR I:= 1 TO 46 DO
2115                BVAREA(.I+36.):=' ';
2116              FOR I := 1 TO 38 DO
2117                DBLINE (.I.) := DBAREA (.I.);
2118              FOR I := 1 TO 82 DO
2119                DBLINE (.I + 38.) := BVAREA(.I.);
2120              FOR I := 121 TO 130 DO
2121                DBLINE (.I.) := ' ';
2122              DB_BUFFER.WRITE (DBLINE);
2123            END;
2124          ID_REM := BE_ID_REM;
2125          ID_LOC := BE_ID_LOC;
2126          WORK_BUFF_CONVERT(DBCMD_BUFF,BUFFER);
2127          LENGTH := BE_LENGTH;
2128        END;
2129      MS.SEND (ID_REM,
2130               ID_LOC,
2131               BUFFER,
2132               LENGTH,
2133               RCODE);
2134      "CHECK RCODE"
2135    END;
2136 END;
2137
2138
2139 "################
2140 #  JOBPROCESS  #
2141 ################"
2142
2143
2144 TYPE JOBPROCESS =
2145 PROCESS
2146   (TYPEUSE: TYPERESOURCE; DISKUSE: RESOURCE;
2147    CATALOG: DISKCATALOG;
2148    INBUFFER, OUTBUFFER: PAGEBUFFER;
2149    INREQUEST, INRESPONSE,
2150    OUTREQUEST, OUTRESPONSE: ARGBUFFER;
2151    STACK: PROGSTACK; FEHINT : HINTMON);
2152
2153 "PROGRAM DATA SPACE = " +9000
2154 "KSU - ALLOWS MAX DATA ADDRESSABILITY"
2155
2156 CONST MAXFILE = 2;
2157 TYPE FILE = 1..MAXFILE;
2158
2159 VAR
2160
```

```
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, WRITEARG,
2175    LOOKUP, IOTRANSFER, 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);
```

```
2215 END;
2216
2217 PROCEDURE ENTRY PUT(F: FILE; P: INTEGER;
2218                      VAR BLOCK: PAGE);
2219 BEGIN FILES[F].WRITE(P, BLOCK) END;
2220
2221 FUNCTION ENTRY LENGTH(F: FILE): INTEGER;
2222 BEGIN LENGTH:= FILES[F].LENGTH END;
2223
2224 PROCEDURE ENTRY MARK(VAR TOP: INTEGER);
2225 BEGIN TOP:= ATTRIBUTE(HEAPTOP) END;
2226
2227 PROCEDURE ENTRY RELEASE(TOP: INTEGER);
2228 BEGIN SETHEAP(TOP) END;
2229
2230 PROCEDURE ENTRY IDENTIFY(HEADER: LINE);
2231 BEGIN OPSTREAM.RESET(HEADER) END;
2232
2233 PROCEDURE ENTRY ACCEPT(VAR C: CHAR);
2234 BEGIN OPSTREAM.READ(C) END;
2235
2236 PROCEDURE ENTRY DISPLAY(C: CHAR);
2237 BEGIN OPSTREAM.WRITE(C) END;
2238
2239 PROCEDURE ENTRY READPAGE(VAR BLOCK: PAGE;
2240                           VAR EOF: BOOLEAN);
2241 BEGIN INBUFFER.READ(BLOCK, EOF) END;
2242
2243 PROCEDURE ENTRY WRITEPAGE(BLOCK: PAGE; EOF: BOOLEAN);
2244 BEGIN OUTBUFFER.WRITE(BLOCK, EOF) END;
2245
2246 PROCEDURE ENTRY READLINE(VAR TEXT: LINE);
2247 BEGIN END;
2248
2249 PROCEDURE ENTRY WRITELINE(TEXT: LINE);
2250 BEGIN END;
2251
2252 PROCEDURE ENTRY READARG(S: ARGSEQ; VAR ARG: ARGTYPE);
2253 BEGIN
2254   IF S = INP THEN INRESPONSE.READ(ARG)
2255              ELSE OUTRESPONSE.READ(ARG);
2256 END;
2257
2258 PROCEDURE ENTRY WRITEARG(S: ARGSEQ; ARG: ARGTYPE);
2259 BEGIN
2260   IF S = INP THEN INREQUEST.WRITE(ARG)
2261              ELSE OUTREQUEST.WRITE(ARG);
2262 END;
2263
2264
2265
2266 PROCEDURE ENTRY LOOKUP
2267 (ID: IDENTIFIER; VAR ATTR: FILEATTR; VAR FOUND: BOOLEAN);
2268
```

```
2269 "CATALOG IS SEARCHED FOR PRIVATE DISK FOLLOWED  - KSU"
2270 "BY SYSTEM DISK ONLY SEQCODE FILES ARE FOUND ON - KSU"
2271 "THE SYSTEM DISK                               - KSU"
2272
2273 BEGIN
2274   CATALOG.LOOKUP(ID, ATTR, FOUND, PRIVATEDISK);
2275   IF NOT FOUND THEN BEGIN
2276     CATALOG.LOOKUP(ID, ATTR, FOUND, SYSTEMDISK);
2277     FOUND:= FOUND AND (ATTR.KIND = SEQCODE);
2278   END;
2279 END;
2280
2281 PROCEDURE ENTRY IOTRANSFER
2282 (DEVICE: IODEVICE; VAR PARAM: IOPARAM; VAR BLOCK: PAGE);
2283 BEGIN
2284   IF (DEVICE = PRIVATEDISK) OR (DEVICE = SYSTEMDISK)  "KSU"
2285   THEN
2286   BEGIN
2287     DISKUSE.REQUEST;
2288     IO(BLOCK, PARAM, DEVICE);
2289     WITH PARAM DO
2290       IF STATUS = ENDFILE THEN                "KSU"
2291         STATUS:= COMPLETE;                    "KSU"
2292     DISKUSE.RELEASE;
2293   END ELSE
2294     IO(BLOCK, PARAM, DEVICE);
2295 END;
2296
2297 PROCEDURE ENTRY IOMOVE(DEVICE: IODEVICE;
2298                        VAR PARAM: IOPARAM);
2299 BEGIN IO(PARAM, PARAM, DEVICE) END;
2300
2301 FUNCTION ENTRY TASK: TASKKIND;
2302 BEGIN TASK:= JOBTASK END;
2303
2304 PROCEDURE ENTRY RUN
2305 (ID: IDENTIFIER; VAR PARAM: ARGLIST;
2306  VAR LINE: INTEGER; VAR RESULT: RESULTTYPE);
2307 BEGIN CALL(ID, PARAM, LINE, RESULT) END;
2308
2309 PROCEDURE ENTRY HINT(HFUNCTION : ENV_FUNCTION;
2310                      VAR STAT : ENV_STAT;
2311                    FILE_NAME : ENV_FILE_NAME;
2312                        REFER : ENV_REFER;
2313                     LINKPATH : ENV_LINKPATH;
2314                   CTRL_FIELD : ENV_CTRL_FIELD;
2315                    VELEMENTS : ENV_VELEMENTS;
2316                    VAR VAREA : ENV_VAREA;
2317                         ENDP : ENV_ENDP);
2318 VAR SPNUM : INTEGER;
2319 BEGIN
2320   SPNUM := 1;
2321   WITH SEND_BUFF DO
2322     BEGIN
```

```
2323        BFUNCTION := HFUNCTION;
2324        BFILE_NAME := FILE_NAME;
2325        BREFER := REFER;
2326        BLINKPATH := LINKPATH;
2327        BCTRL_FIELD := CTRL_FIELD;
2328        BVELEMENTS := VELEMENTS;
2329        BENDP := ENDP;
2330      END;
2331    FEHINT.SEND (SPNUM, SEND_BUFF);
2332    FEHINT.APRETURN (SPNUM, HR_BUFF);
2333    STAT := HR_BUFF.BSTAT;
2334    VAREA := HR_BUFF.BVAREA;
2335    SEND_BUFF:= HR_BUFF;
2336  END;
2337
2338  PROCEDURE INITIALIZE;
2339  VAR I: INTEGER; PARAM: ARGLIST;
2340      LINE: INTEGER; RESULT: RESULTTYPE;
2341  BEGIN
2342    INIT OPERATOR(TYPEUSE), OPSTREAM(OPERATOR),
2343      INSTREAM(INBUFFER), OUTSTREAM(OUTBUFFER);
2344    INSTREAM.INITREAD; OUTSTREAM.INITWRITE;
2345    FOR I:= 1 TO MAXFILE DO
2346      INIT FILES[I](TYPEUSE, DISKUSE, CATALOG);
2347    INIT CODE(TYPEUSE, DISKUSE, CATALOG);
2348    WITH PARAM[2] DO
2349    BEGIN TAG:= IDTYPE; ARG:= 'CONSOLE      ' END;
2350    CALL('DO          ', PARAM, LINE, RESULT);
2351    OPERATOR.WRITE('JOBPROCESS:(:10:)',
2352                   'TERMINATED (:10:)');
2353  END;
2354
2355  BEGIN INITIALIZE END;
2356
2357
2358  "###############
2359  #  IOPROCESS  #
2360  ###############"
2361
2362
2363  TYPE IOPROCESS =
2364  PROCESS
2365    (TYPEUSE: TYPERESOURCE; DISKUSE: RESOURCE;
2366     CATALOG: DISKCATALOG; SLOWIO: LINEBUFFER;
2367     BUFFER: PAGEBUFFER; REQUEST, RESPONSE: ARGBUFFER;
2368     STACK: PROGSTACK; IOTASK: TASKKIND);
2369
2370  "PROGRAM DATA SPACE = " +2000
2371
2372  TYPE FILE = 1..1;
2373
2374  VAR
2375
2376  OPERATOR: TERMINAL; OPSTREAM: TERMINALSTREAM;
```

```
2377
2378 IOSTREAM: CHARSTREAM; IOFILE: DATAFILE;
2379
2380 CODE: PROGFILE2;
2381
2382 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, IOTRANSFER, IOMOVE, TASK, RUN;
2387
2388 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;
```

```
2431 BEGIN LENGTH:= IOFILE.LENGTH END;
2432
2433 PROCEDURE ENTRY MARK(VAR TOP: INTEGER);
2434 BEGIN TOP:= ATTRIBUTE(HEAPTOP) END;
2435
2436 PROCEDURE ENTRY RELEASE(TOP: INTEGER);
2437 BEGIN SETHEAP(TOP) END;
2438
2439 PROCEDURE ENTRY IDENTIFY(HEADER: LINE);
2440 BEGIN OPSTREAM.RESET(HEADER) END;
2441
2442 PROCEDURE ENTRY ACCEPT(VAR C: CHAR);
2443 BEGIN OPSTREAM.READ(C) END;
2444
2445 PROCEDURE ENTRY DISPLAY(C: CHAR);
2446 BEGIN OPSTREAM.WRITE(C) END;
2447
2448 PROCEDURE ENTRY READPAGE(VAR BLOCK: PAGE;
2449                           VAR EOF: BOOLEAN);
2450 BEGIN BUFFER.READ(BLOCK, EOF) END;
2451
2452 PROCEDURE ENTRY WRITEPAGE(BLOCK: PAGE; EOF: BOOLEAN);
2453 BEGIN BUFFER.WRITE(BLOCK, EOF) END;
2454
2455 PROCEDURE ENTRY READLINE(VAR TEXT: LINE);
2456 BEGIN SLOWIO.READ(TEXT) END;
2457
2458 PROCEDURE ENTRY WRITELINE(TEXT: LINE);
2459 BEGIN SLOWIO.WRITE(TEXT) END;
2460
2461 PROCEDURE ENTRY READARG(S: ARGSEQ; VAR ARG: ARGTYPE);
2462 BEGIN REQUEST.READ(ARG) END;
2463
2464 PROCEDURE ENTRY WRITEARG(S: ARGSEQ; ARG: ARGTYPE);
2465 BEGIN RESPONSE.WRITE(ARG) END;
2466
2467 PROCEDURE ENTRY LOOKUP
2468 (ID: IDENTIFIER; VAR ATTR: FILEATTR; VAR FOUND: BOOLEAN);
2469
2470 "PRIVATE DISK IS SEARCHED FIRST, FOLLOWED        - KSU"
2471 "BY SYSTEM DISK ONLY SEQCODE FILES ARE FOUND ON - KSU"
2472 "THE SYSTEM DISK                                - KSU"
2473 BEGIN
2474   CATALOG.LOOKUP(ID, ATTR, FOUND, PRIVATEDISK);
2475   IF NOT FOUND THEN BEGIN
2476     CATALOG.LOOKUP(ID, ATTR, FOUND, SYSTEMDISK);
2477     FOUND:= FOUND AND (ATTR.KIND = SEQCODE);
2478   END;
2479 END;
2480
2481 PROCEDURE ENTRY IOTRANSFER
2482 (DEVICE: IODEVICE; VAR PARAM: IOPARAM; VAR BLOCK: PAGE);
2483 BEGIN
2484   IF (DEVICE = PRIVATEDISK) OR (DEVICE = SYSTEMDISK)  "KSU"
```

```
2485    THEN
2486    BEGIN
2487      DISKUSE.REQUEST;
2488      IO(BLOCK, PARAM, DEVICE);
2489      WITH PARAM DO                                  "KSU"
2490        IF STATUS = ENDFILE THEN
2491           STATUS:= COMPLETE;                        "KSU"
2492      DISKUSE.RELEASE;
2493    END ELSE
2494      IO(BLOCK, PARAM, DEVICE);
2495  END;
2496
2497  PROCEDURE ENTRY IOMOVE(DEVICE: IODEVICE;
2498                         VAR PARAM: IOPARAM);
2499  BEGIN IO(PARAM, PARAM, DEVICE) END;
2500
2501  FUNCTION ENTRY TASK: TASKKIND;
2502  BEGIN TASK:= IOTASK END;
2503
2504  PROCEDURE ENTRY RUN
2505  (ID: IDENTIFIER; VAR PARAM: ARGLIST;
2506   VAR LINE: INTEGER; VAR RESULT: RESULTTYPE);
2507  BEGIN CALL(ID, PARAM, LINE, RESULT) END;
2508
2509
2510  PROCEDURE INITIALIZE;
2511  VAR PARAM: ARGLIST; LINE: INTEGER; RESULT: RESULTTYPE;
2512  BEGIN
2513    INIT OPERATOR(TYPEUSE), OPSTREAM(OPERATOR),
2514      IOSTREAM(BUFFER),
2515      IOFILE(TYPEUSE, DISKUSE, CATALOG),
2516      CODE(TYPEUSE, DISKUSE, CATALOG);
2517    IF IOTASK = INPUTTASK THEN IOSTREAM.INITWRITE
2518                          ELSE IOSTREAM.INITREAD;
2519    CALL('IO          ', PARAM, LINE, RESULT);
2520    OPERATOR.WRITE('IOPROCESS: (:10:)',
2521                   'TERMINATED (:10:)');
2522  END;
2523
2524  BEGIN INITIALIZE END;
2525
2526
2527  "################
2528  #  CARDPROCESS  #
2529  ################"
2530
2531
2532  TYPE CARDPROCESS =
2533  PROCESS
2534    (TYPEUSE: TYPERESOURCE; BUFFER: LINEBUFFER);
2535
2536  "CARDSPROCESS HAS BEEN MODIFIED FOR THE USE OF A  - KSU"
2537  "VIRTUAL READER ENDFILE OR ENDMEDIUM CAUSES THE   - KSU"
2538  "PROCESS TO TERMINATE                             - KSU"
```

```
2539
2540 VAR OPERATOR: TERMINAL; TEXT: LINE;
2541     PARAM: IOPARAM; OK, ENDF: BOOLEAN;
2542 BEGIN
2543   INIT OPERATOR(TYPEUSE);
2544   PARAM.OPERATION:= INPUT;
2545   ENDF:= FALSE;
2546   REPEAT
2547     REPEAT
2548       IO(TEXT, PARAM, CARDDEVICE);
2549       CASE PARAM.STATUS OF
2550         COMPLETE:
2551           OK:= TRUE;
2552         INTERVENTION, FAILURE:
2553           BEGIN OK:= FALSE; WAIT END;
2554         ENDFILE, ENDMEDIUM:          "KSU - SUPPORTS"
2555                                      "VIRTUAL READER"
2556           BEGIN OK:= TRUE; ENDF:= TRUE; END;
2557         TRANSMISSION:
2558           BEGIN
2559             OPERATOR.WRITE('CARDS: (:10:)',
2560                            'ERROR(:10:)');
2561             OK:= FALSE;
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 "####################
2573 #  PRINTERPROCESS  #
2574 ####################"
2575
2576
2577 TYPE PRINTERPROCESS =
2578 PROCESS
2579   (TYPEUSE: TYPERESOURCE; BUFFER: LINEBUFFER);
2580
2581 VAR OPERATOR: TERMINAL; PARAM: IOPARAM;
2582     TEXT: LINE;
2583
2584 BEGIN
2585   INIT OPERATOR(TYPEUSE);
2586   PARAM.OPERATION:= OUTPUT;
2587   CYCLE
2588     BUFFER.READ(TEXT);
2589     IO(TEXT, PARAM, PRINTDEVICE);
2590     IF PARAM.STATUS <> COMPLETE THEN
2591     BEGIN
2592       OPERATOR.WRITE('PRINTER: (:10:)',
```

```
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 "###################
2604 #  LOADERPROCESS  #
2605 ###################"
2606
2607 TYPE LOADERPROCESS=
2608 PROCESS(DISKUSE: RESOURCE);
2609
2610 CONST SOLOADDR = 24;
2611 VAR PARAM: IOPARAM;
2612
2613 PROCEDURE INITIALIZE(PAGENO: UNIV IOARG);
2614 BEGIN
2615   WITH PARAM DO
2616   BEGIN
2617     OPERATION:= CONTROL;
2618     ARG:= PAGENO;
2619   END;
2620 END;
2621
2622 BEGIN
2623   INITIALIZE(SOLOADDR);
2624   "AWAIT BEL SIGNAL"
2625   IO(PARAM, PARAM, TYPEDEVICE);
2626   "LOAD SOLO SYSTEM"
2627   DISKUSE.REQUEST;
2628   IO(PARAM, PARAM, PRIVATEDISK);
2629   DISKUSE.RELEASE;
2630 END;
2631
2632 "####################
2633 #  INITIAL PROCESS  #
2634 ####################"
2635
2636
2637 VAR
2638
2639 TYPEUSE: TYPERESOURCE;
2640
2641 DISKUSE: RESOURCE; CATALOG: DISKCATALOG;
2642
2643 INBUFFER, OUTBUFFER: PAGEBUFFER;
2644
2645 CARDBUFFER, PRINTERBUFFER: LINEBUFFER;
2646
```

```
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 FNP : FE_NRC;
2671 MRESP : MS_RESPONSE;
2672 FCONNP: 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 BCONNP: 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     FCONNP(NRCDIRECTORY, MS, TYPEUSE);
2699     OPERATOR.WRITE('INIT : (:10:)',
2700                    'FE_CONN_PROCESS(:10:)');
```

```
2701    INIT
2702      FRP(FEHINT, FRMON, TYPEUSE, MS),
2703      FNP(FRMON, FEHINT, FIDT, NRCDIRECTORY,
2704          TYPEUSE, MS);
2705      OPERATOR.WRITE('INIT : (:10:)','FNP(:10:)');
2706    INIT
2707      MRESP(FEHINT, FIDT, MS, TYPEUSE),
2708      BCONNP(NRCDIRECTORY, MS, TYPEUSE);
2709      OPERATOR.WRITE('INIT : (:10:)',
2710                     'BE_CONN_PROCESS(:10:)');
2711    INIT
2712      BEBINT, BIDT (TYPEUSE);
2713      OPERATOR.WRITE('INIT : (:10:)','BIDT(:10:) ');
2714    INIT
2715      MRQSTP(BEBINT, BIDT, MS, TYPEUSE),
2716      BNP(BIDT, NRCDIRECTORY, TYPEUSE, MS);
2717      OPERATOR.WRITE('INIT : (:10:)','BNP(:10:)');
2718    INIT
2719      CDP(BEBINT, MS, TYPEUSE, PRINTERBUFFER),
2720      READER(TYPEUSE, CARDBUFFER),
2721      WRITER(TYPEUSE, PRINTERBUFFER),
2722      PRODUCER(TYPEUSE, DISKUSE, CATALOG, CARDBUFFER,
2723        INBUFFER, INREQUEST, INRESPONSE,
2724        INSTACK, INPUTTASK),
2725      CONSUMER(TYPEUSE, DISKUSE, CATALOG, PRINTERBUFFER,
2726        OUTBUFFER, OUTREQUEST, OUTRESPONSE,
2727        OUTSTACK, OUTPUTTASK),
2728      MASTER(TYPEUSE, DISKUSE, CATALOG,
2729        INBUFFER, OUTBUFFER,
2730        INREQUEST, INRESPONSE,
2731        OUTREQUEST, OUTRESPONSE,
2732        JOBSTACK, FEHINT),
2733      WATCHDOG(DISKUSE);
2734      OPERATOR.WRITE('INIT : (:10:)',
2735                     'WATCHDOG(:10:) ');
2736  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.