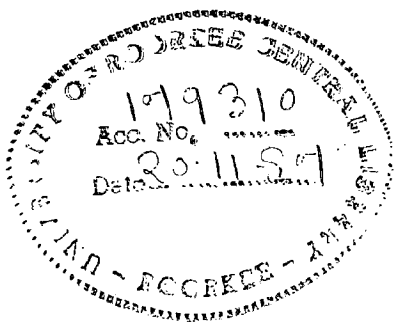
DESIGN AND IMPLEMENTATION OF A PACKAGE FOR PROCESS MIGRATION ON THE DECSYSTEM-2050

A DISSERTATION

submitted in partial fulfilment of the
requirements for the award of the degree
of
MASTER OF TECHNOLOGY
in
ELECTRONICS & COMMUNICATION ENGINEERING
(COMPUTER SCIENCE & TECHNOLOGY)

By

AJIT MALAVIYA



DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING
UNIVERSITY OF ROORKEE
ROORKEE-247 667 (INDIA)

April 1987

CANDIDATE'S DECLARATION

I hereby declare that the work which is being presented in the dissertation entitled, 'DESIGN AND IMPLEMENTATION OF A PACKAGE FOR PROCESS MIGRATION ON THE DECSYSTEM-2050', in partial fulfilment for the award of the degree of MASTER OF TECHNOLOGY in Electronics and Communication Engineering with specialization in COMPUTER SCIENCE AND TECHNOLOGY, submitted in the Department of Electronics and Communication Engineering, University of Roorkee, Roorkee, is an authentic record of my own work carried out for a period of about eight months, from August 1986 to March, 1987, under the supervision of Dr. (Mrs.) K.Garg, Reader, Department of Electronics and Communication engineering, University of Roorkee, Roorkee, India.

The matter embodied in this dissertation has not been submitted by me for the award of any other degree or diploma.

APRIL 27 ,1987 (AJIT MALAVIYA)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Dr.(Mrs.) K.Garg
Reader
Electronics and Communication
Engineering Department,
University of Roorkee,
Roorkee, U.P., INDIA.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Dr. (Mrs.) K.Garg, Reader, Department of Electronics and Communication Engineering, University of Roorkee, Roorkee, for her invaluable assistance. Her constant interest and excellent advice helped me to complete this dissertation successfully.

My thanks are due to my friends for their help and cooperation.

Finally, I am thankful to Mr. D.C.Bhardwaj for his excellent typing of this thesis.

AJIT MALAVIYA

TABLE OF CONTENTS

contd...

CHAPTER PAGE NO

ABSTRACT

A distributed system has to operate in a reliable and efficient manner in the presence of failures and unbalanced resource demands. Process Migration emerges as a possible solution to these problems.

In this dissertation an environment for supporting process migration on the DECSYSTEM-2050 has been implemented. Problems associated with process migration have been identified and solutions to some of these have been given.

CHAPTER - 1

INTRODUCTION

1.1 INTRODUCTION AND MOTIVATION

Computers have been becoming cheaper over the years, and so, using multiple computers in conjunction was an inevitable development. In the earliest stages, this was done on an adhoc basis, by connecting existing computers through simple interfaces and adding the controlling software as a separate utility, rather than as a part of the system. A simple network would, thus, have had a function-oriented protocol, such as a File Transfer Protocol (FTP) built on top of an interprocess.communication facility [1,2]. In fact, functionally, such a network would have been nothing more than a glorified telephone. Further developments brought about some systemisation and standardisation. The concept of expandibility of a network, its reliability and cost-effectiveness came in with time.

In the context of the present state of art, interconnected systems generally fall into three categories - multiprocessors, local area networks (LANs) and wide area or long-haul networks [3]. Multiprocessors consist of small computing units, very closely linked and synchronised, which are meant to function as a single machine. Wide area networks and interconnections of LANs. These two systems do not hold much relevance in the present discussion. Here we shall be

dealing mainly with systems based on local area networks.

Depending on the type of operating systems they support, one can classify networks as ordinary LANs or as Distributed Systems [3]. The term Distributed system is not very clearly defined, and has been used to refer to any thing from a multiprocessor to a wide area network. Throughout this dissertation, however, the term Distributed system will be used to refer to a loosely-coupled network of autonomous computers which supports a system wide or distributed operating system.

A loosely-coupled network is one where computers do not share memory. This characteristic is important to exclude multiprocessors from the definition, as they invariably have a tightly- coupled processor configuration. Some networks do however, provide means for computers to share memory, but not at the instruction level. That is, a computer cannot execute a code that is remote without first copying it into a local memory area.

Autonomous computer networks are those that do not have a master-slave relationship. For instance, a system based on the Intel 8086 and having an 8087 or 8089 co-processor could not be classified as a Distributed system because one processor is in complete control of the other. On the

other hand, systems connected hierarchically (like a tree) can form a Distributed system. if they are functionally independent.

System-wise operating system. refers to systems where services are requested by name and not by location [4]. Traditionally such an operating system is called a Distributed operating system .

Network will be used to refer to a computer network in the generic sense. In later chapters, the difference between a distributed system and a network will be brought out more clearly. Till then it would suffice to say that a distributed system. is a network with a large degree of cohesiveness and transparency.

As a comparatively recent development in the field of networking, Distributed systems have been the subject of a large amount of research. Most of this research is related to the reliability of systems and optimization of their use. This is particularly true when it comes to allocation of system resources among users. Resources of a system consist of memory, diskspace, processor time and information, primarily. Optimizing, or even controlling such a large amount of variables is no mean task, especially so, when the demand for resources do not follow a deterministic pattern. At a given time some

computers in a network might be heavily loaded while others are running relatively free. One or more computers might have failed, thereby reducing the amount of available resources, while not necessarily reducing the demand.

In such circumstances the concept of process migration becomes important. A process is the entity which causes actual computation to be carried out. It is the executing version of the whole or a part of a program. All resources are allocated to or used by a process. Migration of a process refers to changing the environment in which it is running. More specifically, it describes the movement of a process from one computer to another, without the loss of its context. Processes are made to migrate for the purpose of even-ing out the allocation of and demand for system resources.

Associated with process migration are the problems of when and where to migrate a process, how to migrate it, preserving its context [5], and ensuring that a change of environment does not affect its execution. Different distributed systems have approached these problems in different ways, adopting differing architectures and developing many process management techniques. Increasing the system efficiency without causing inconvenience to the user, has of course, been the underlying theme.

In this dissertation, a study of distributed systems has been made so as to bring out the characteristics of an operating system that aid in making process migration an asset. Various issues related to process management have been discussed. A suitable environment for migrating processes on the DECSYSTEM-2050 has been designed and implemented.

1.2 OBJECTIVES OF THE WORK UNDERTAKEN

The objectives of this dissertation are :

(i) To study the mechanism of process migration and the operating system environment suited to it.

(ii) To study the problems associated with process migration and possible solution to these problems.

(iii) To design a software package that generates partially the environment of a distributed system and to implement process migration on it.

1.3 ORGANISATION OF THE THESIS

After the introduction to the topic, which has already been presented, process Migration is studied in relation to the operating system environment, in chapter-2. The reason for preferring a Distributed Operating System (DOS) to a Network Operating system (NOS) is mentioned. Object-oriented software organisation is introduced and shown to be better suited for operating system design.

In Chapter-3, the mechanism of Process Migration and related issues have been presented. Here we have dealt with process management details like dynamic process scheduling and keeping track of migrating processes. Physical transportation of a process based on the DEMOS/MP implementation has been described.

Chapter-4 deals with the design of a package for implementing process migration on the DECSYSTEM-2050. After an overview of the facilities provided by the DECSYSTEM-2050, the package design has been explained in detail.

We conclude with a summary of the work done and the results achieved. Some suggestions as to what further work can be done on the subject has been given.

A list of references has been appended, detailing the various sources consulted in the preparation of the dissertation.

The program for the package is given as an Appendix.

CHAPTER - 2

PROCESS MIGRATION AND THE OPERATING ENVIRONMENT

2.1 INTRODUCTION: ADVANTAGES OF PROCESS MIGRATION

Before taking up any discussion concerning the 'how' or 'what' of process migration, it is necessary to justify its incorporation into a system. In this section our aim is to show how process migration goes towards enhancing the efficiency and reliability of a system.

Consider a system that does not support any type of process migration. The processes remain at the nodes on which they were created. It is quite possible that the resource requests be one sided, leading either to deadlock situations or a highly inefficient behaviour. Thus a process that requires a large amount of primary memory will either prevent other processes from being loaded or increase disk I/O by frequently demanding new pages (segments); depending upon the memory management scheme. Many computation-intensive processes scheduled on the same processor compete for processor time leaving the I/O devices relatively idle.

On the other extreme, running a large number of I/O bound processes leads to wastage of computing power, as the processes would be blocked for most of the time [6]. It would be infinitely more desirable for the demands to be more

evenly balanced so that the resources are properly utilized.

In network based systems, another vital issue is the traffic in the communication channels. Communicating processes residing on distant computers tend to increase this traffic. To reduce the communication load, the number of hops per message should be reduced. Alternatively, the processes can be scheduled on nodes between which the lines are under-utilized. All these cases require the processes to be dynamically distributed. Static scheduling, that is, at creation time only - would not be desirable as the system cannot have foreknowledge of the required resources [7].

Another factor supporting process migration is the sparse distribution of system processes in the network. In order to save memory space, most systems prefer to keep only a limited copy of system processes [8]. Depending on the requirements either the calling process or called process is made to migrate. It is possible for a utility routine to be divided into separate task forces running and scheduled concurrently, giving rise to the need for process migration.

Reliability is of prime importance in any distributed system, and is, in fact, one of the factors that led to their development. Instead of having all computing power localised- where a single failure can crash the whole system - it is

distributed over a number of autonomous systems. A failure, then, will just partially reduce the available resource, or in the worst case, deprive a small percentage of users of it. Failing systems can save their processes by migrating them elsewhere. This is particularly useful in situation where it is not possible to restart a process- like one which modifies an important database.

A very important advantage of having process migration is the support of a 'pool of processors' architecture [9]. In such a system, instead of the terminalsbeing controlled by a host computer, they are connected directly to a network through concentrators. Various computers are also connected to this network. The concept of local or remote resources is completely removed. When the user needs processor time, any processor in the computer bank can be assigned to him. Demands for devices can easily be met by migrating the calling process to a dedicated processor controlling the required device. In case of failure of a processor, its processes can be rescheduled among the other computers in the bank without the user being aware of it. This failure of a processor only reduces the available resources; providing a higher degree of reliability.

2.2 TRANSPARENCY IN THE PRESENCE OF PROCESS MIGRATION

2.2.1 Distributed Operating System versus Network Operating Systems

A distributed system has to provide a degree of transparency in its operation. That is, the user of the system must not be aware of the machine boundaries in the system. Coupled with process migration, this requirement imposes certain restrictions on the system design and layout; as a result of which it is preferable to have a Distributed Operating System (DOS) rather than a Network Operating System (NOS).

For instance, take the file system in a network. This file system is not globally mangged in an NOS, and so to access a remote file it is necessary to specify the machine or computer on which the required directory resides. In a system like the Newcastle Connection [10], which operates in an UNIX environment, the path through directories has to be specified. As no directory connects two computers, one has to assume a virtual superior directory (Fig.1). An OPENFILE request would take the form

open (" /../machine2/pathname",READ);

where '/..' is the virtual superior directory.

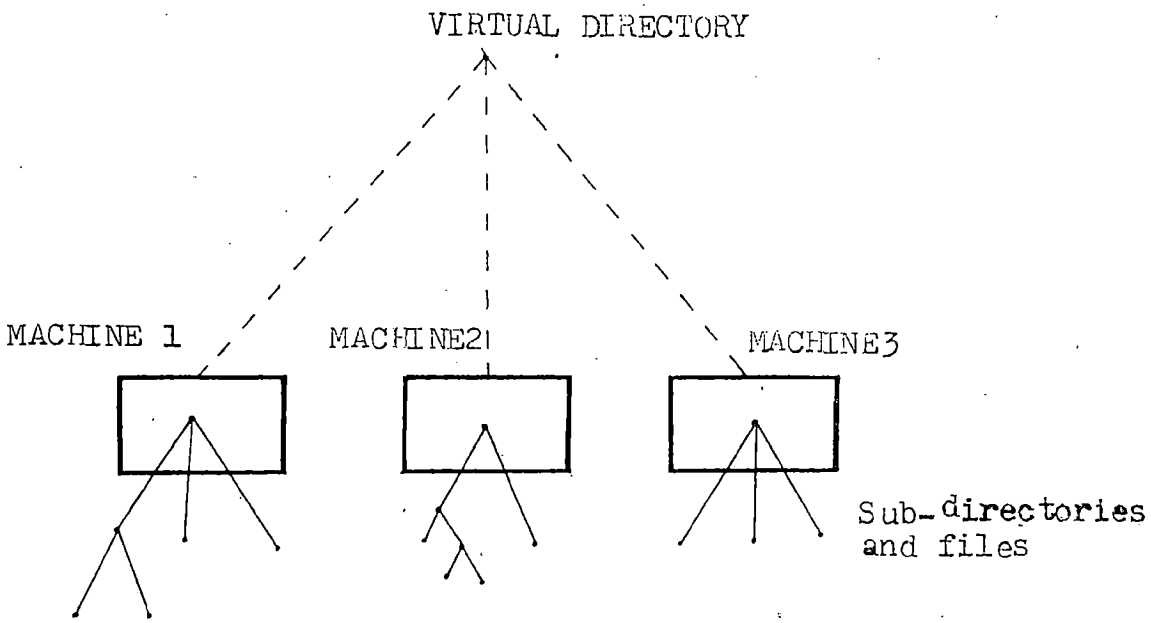


Fig. 1(a) Virtual Directory in an NOS

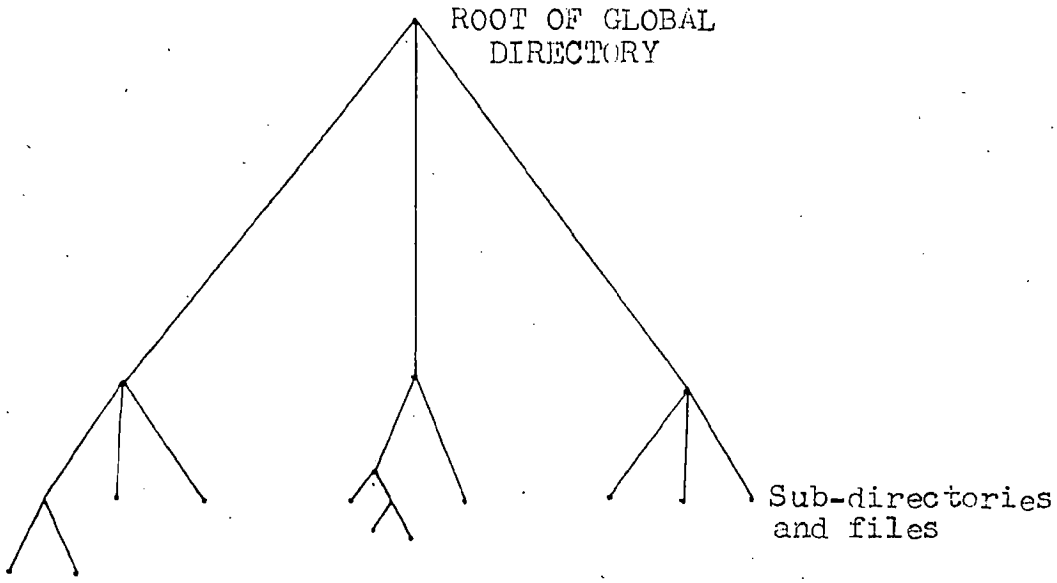


Fig. 1(b) Global Directory in a DOS

Migrating processes face a severe problem stemming from such a file management system. On the initial open request, the system assignsa local logical number to the file being accessed. This number is unique only in the host machine boundary. If the process is taken to another computer, the file number it uses refers to an entirely different file. To solve this problem, the system has to intercept each request of a migrated process and re-direct it to the creating node, resulting in a large amount of overhead.

A DOS would be having a global file name space [1,11]. Logical numbers assigned to a file anywhere in the system are unique, and independent of the caller's or called locations. It is not necessary to specify the location of a directory when requesting a file. On LOCUS [11], another UNIX based system, the OPENFILE would be

```
open("/etc/passwd",READONLY);
```

where 'etc' is a global system directory.

Another area where machine boundaries become visible is the CREATEPROCESS system call [1]. In an NOS the process will be created on the host computer unless another node has been specified, and a remote login performed. A DOS does not require any remote login, and the new process is automatically scheduled in the best suited environment.

Network and Distributed operating systemshave different implementation schemes. An NOS is developed by adding software on a native operating system, while a DOS is designed from scratch, in a manner that supports a global management scheme. This difference often gets reflected in synchronisation signals like those used for interrupts and interprocess communication, making it uneconomical if not infeasible to support process mig - ration transparently in a Network Operating System [1].

2.2.2 Object- Oriented Software Organisation

That a network has a DOS does not necessarily ensure that it can handle process migration easily. Here we introduce an Object-Oriented system which has been used successfully in systems like EDEN [7] and AMEOBA [12]. It is not that a system not organised in this fashion cannot efficiently support process migration, but that transparency is inherent, rather than forced, in a system that is (organised in this manner).

In an object-oriented organisation a software segment and its related databases together form a single entity called an object. Any reference to the database can be made only through the relevant object, a message being the initiator of an objects activity. An object is the smallest entity that can be moved from one kernel to

another. A major advantage resulting from this software abstraction is that of consistency. The whole object, that is, the program and database are referred to by a single location independent name. It is easy to standardise the interface between objects. A user's request for the invocation of a remote request is the same as one invoking a local object, providing a high degree of transparency. Contrast this with a procedure calling system where a request sent to a remote node will be interpreted by the kernel, compiled into possibly a chain of procedure calls, executed, the result collected into a message and then returned.

Another advantage of using an object oriented organisation is that of an inherent global protection scheme. It is usual to have a capability based addressing scheme [7, 12] in such systems. Every object in the system is referred to by a logical entity called a port. Accessibility of the port address is itself the right to use that port and hence that object [12]. In a migrating process, the port address or the capability is a part of its context and will thus be present in the new environment also. A procedure-call system like a DECSYSTEM-2050 specifies a process 's capability by what is given to it by the creating process and not from its context [13].

The port addressing scheme is exactly similar to the link based interprocess communication method preferred by distributed systems [7,11,12]. A link is a unidirectional logical communication channel. The 'link owner' is the process that creates or requests for the creation of the link, and can receive a message sent on it. Any process that has the link number is a 'link holder' and can send messages on it. Possession of the link number is implicitly the capability to use that link.

As links and ports are very similar in their characteristics, processes can easily be treated as objects having a system-wide identifier, and thus giving uniformity to the operating systems organisation.

CHAPTER - 3

DESIGN ISSUES RELATED TO PROCESS MIGRATION

Uptil now, we have discussed the environment in which a process can, or cannot, migrate, and what causes it to do so. This chapter deals with the mechanism of actually transporting a process and the decision and book keeping issues.

3.1 DYNAMIC PROCESS SCHEDULING

The first step towards migrating a process is deciding which process to migrate, and where. Such a decision has to be taken with the whole system in consideration. Thus there has to be some mechanism by which information about process status can be collected and processed globally. We shall be presenting a possible algorithm for achieving this. Later an algorithm for choosing a migratable process and its destination will also be given. Both of these algorithms provide sub-optimal solutions, as going in for an optimal distribution is uneconomical and time taking [14].

3.1.1 Measuring Processor Load

The information collecting algorithm used by MOS[14], a Multi Computer Operating system is quite simple and can be used to arrive at a sub-optimal solutions.

In a system with 'n' computers (nodes) each one maintains a load vector 'L' of size 'ℓ'. The first entry of the vector L(0) holds a measure of the local processor load and the remaining the load of an arbitrary subset of nodes. This subset does not remain the same, but changes with time. A unit of time 't' is chosen for load balancing considerations. Every 't' seconds the following steps are taken by a processor.

STEP 1 - update the value of its own load vector.

STEP 2 - choose a random number $1 \leq i \leq n$.

Step 3 - send the first half of its vector L to node i.

If this node receives a load vector from some other node, then

STEP 4 - merge the received vector L_R with its own vector L according to the mapping (Fig.2).

$L(j) \rightarrow L(2j),\ 1 \leq j \leq \ell/2-1$

$L_R(j) \rightarrow L(2j+1),\ 0 \leq j \leq \ell/2-1$

While this algorithm does not provide every node with the latest update, its has its own distinct advantages. A comparatively small amount of message passing is required, equal to the number of nodes in the systems. For a complete update the number of messages would be n(n-1). The message size is also considerably smaller, as the value of ℓ is kept lower than n [14]. By choosing the values of l and t properly, it is possible to make the algorithm decently efficient. Sometimes there may be duplication of vector entries in a load vector, but this is unavoidable and permitted.

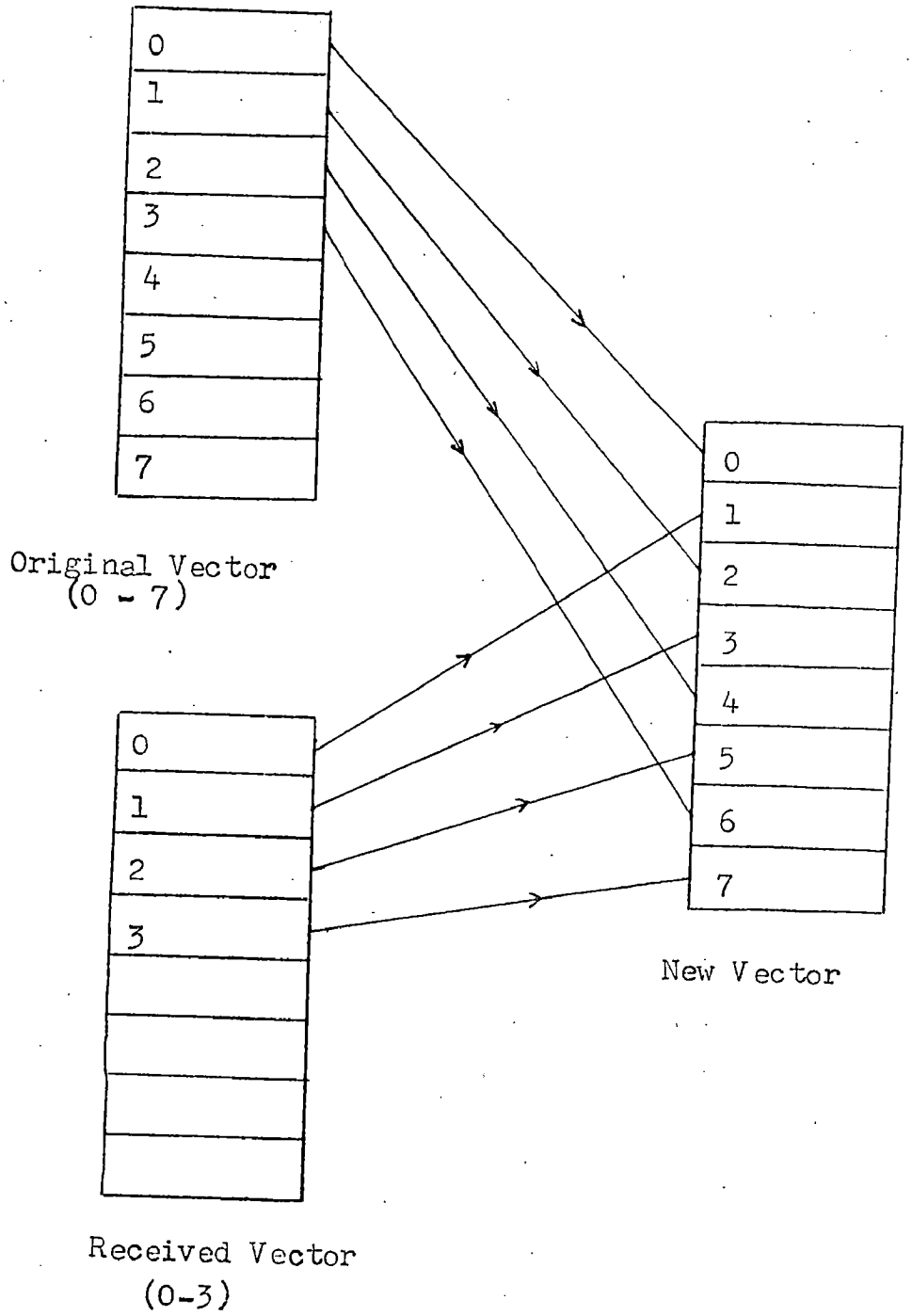


Fig. 2 Mapping Of Load Vectors

3.1.2 Scheduling Algorithms

Based on the information gathering policy discussed above a scheduling algorithm can be run. MOS [14] uses processor load as the only criteria for process migration. The measure of load is the number of processes waiting for processor time. This number is sampled at intervals of 'q' seconds, where

$t = u\ q$; t = the load vector update time.

The local load is measured as

$$V_t = \left(\sum_{i=1}^{u} W_i \right) / (u - \omega)$$

where,

W_i = the number of processes waiting for processor time during the interval (q_{i-1}, q_i).

ω = the number of time quanta (q) out of u possible quanta consumed as the operating system overhead.

If a node finds itself more heavily loaded than others, it selects a process that has been running for some time (more than a fixed minimum) and migrates it to the node with the lowest load. The criteria of a process having had a minimum amount of time at the host node prevents needless migration, that is, only long running processes are migrated.

In this algorithm, the only criteria for load was the number of processes that were waiting in a specified interval.

It might be desirable, however to include a larger number of variables into the decision making algorithm. To achieve this, the Mc Culloch Pitts evaluation procedure [15] can be used. A Mc Culloch -Pitts neuran is a decision cell with Excitation (E) and Inhibitions (I) (Fig.3). The output of this cell is the sum of all Excitation, or zero if the sum of Inhibition is non-zero.

To select a candidate for migration, a process is evaluated through this procedure in context of the current environment. The process with the lowest output value but not zero is the best candidate. If desired, a certain minimum value can be fixed, above which no process migrates. The information collection algorithm is modified so that each load entry contains information describing various factors of the node status. Using this information a node is selected to receive the process. Thereafter either the process is migrated directly or bidding is performed to check whether the receiver state has changed in the duration.

3.2 KEEPING TRACK OF A MIGRATING PROCESS

A long running process may be expected to migrate a number of times in a distributed system. Functions like interprocess communication and calling system processes require the location of a specified process to be known, or at least be traceable at any given time. Keeping track of

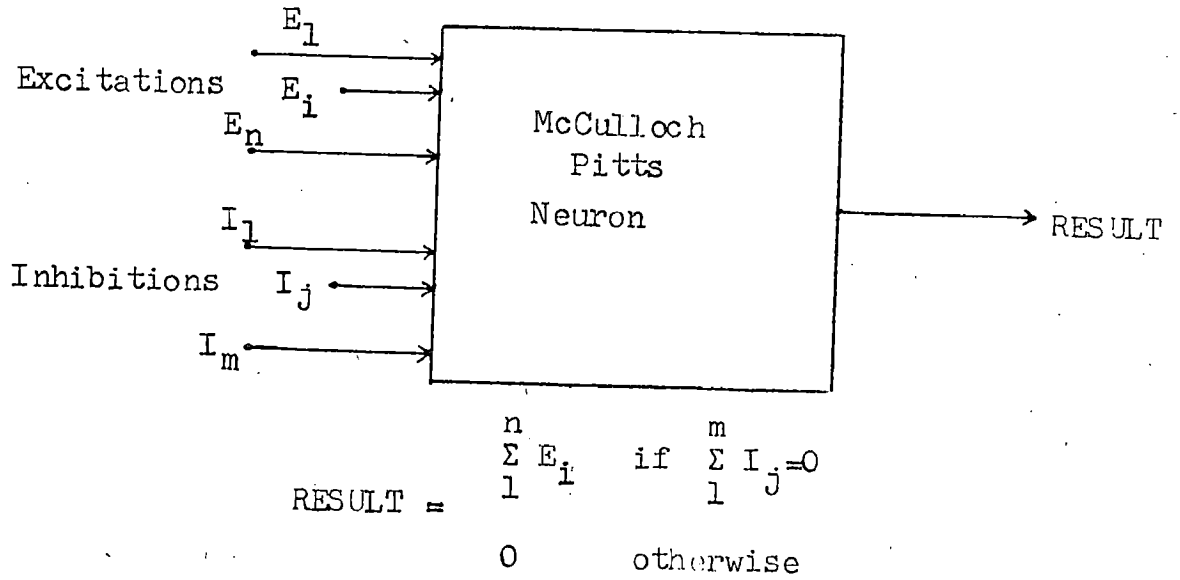
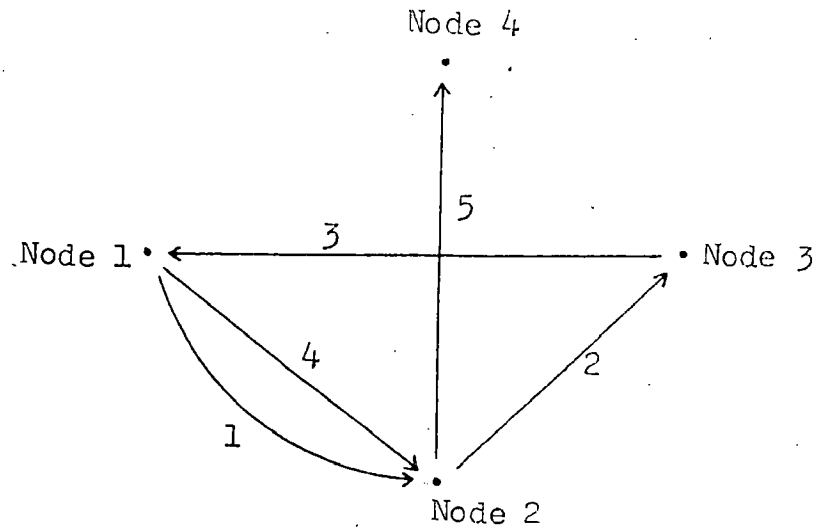


Fig. 3 The McCulloch Pitts Neuron

a system process should normally be simple as the total number of such processes is known before hand. A directory indicating the last known location of a system process is either maintained at every node, or one or more managers take care of this task for the whole system. Except in cases where the manager is totally centralised, updating a process location on each directory for every move is not economical. The number of messages involved would become too large. In most systems, the managers which migrate a process keep track of its last known destination. Any requests are then forwarded, and depending on the current process location may be reforwarded. Such a method, though economical is not very dependable, especially in the face of the failure of an intermediate node. The example given below elucidates on this point.

Consider a system with four nodes. A process resides on Node # 1 initially. With time the process migrates as shown in Fig. 4(a). The record of migration history at the nodes would be as in Fig. 4(b). Now suppose Node # 2 fails and in doing so migrates the process to Node # 4. If Node # 3 needs the process it sends a request to Node # 1 as per its records. Forwarding the request to Node # 2 is not possible as it is gone off-line. The only alternative is to conduct a search. Implementing a simple searching algorithm, the request could be forwarded to Node # (i+1) if Node # i is



Path 1-2-3-1-2-4 Taken By a Process i

NODE	P_1	...	P_i	...	P_n
1			2		
2			✖		
3			1		
4			4		

OFFLINE

'Last Known Destinations' After Failure of Node 2

Fig. 4 Process Path and the Last Known Destination.

off line. Clearly this will not work in our example, as the request will be returned to Node # 3, forcing it to deduce that the process is not available.

Another way out is to break the search into two parts. Initially an attempt is made to track the process through its migration history. When a missing link is encountered, a linear search is started afresh, ending at the node where the search began (Node # 1) instead of the node where the request originated (Node # 3). Such a method would be useful in highly reliable systems only, where the probability of failure is very small, and migration and request timings are such that on the average tracking is faster than searching. In less reliable systems a linear search . than tracking would be more efficient. The searching tracking combination can be improved upon by providing the searching algorithm a history of the tracking. That is during tracking every intermediate node adds its identification to the message. When searching is performed the nodes that have been involved in tracking are skipped entirely.

Most of the systems do not keep a fully distributed process directory [8,16], preferring to allocate the task to a few selected nodes. A failing node then just has to inform its manager of the migration and so the records can be kept upto date. If a manager-node fails, the resident

directory is transferred to some new node in the group which then acts as the manager. This node will inform other managers and its group of the change, and all operations can then proceed as before.

Problems similar to these arise when one of the two communicating user processes migrate. As user processes are dynamically created and deleted, keeping a record of all these processes would require a large amount of space, especially in a system based on a network of time sharing computer. Informing all relevant nodes about a migrated processes is again infeasible because of the large amount of overhead that would be incurred. A possible solution could be achieved by associating an entry with the logical link the processes use to identify each other while communicating. This entry would be resident in the Kernel and would contain the last known destination of the link owner. When a process migrates the entry would be transferred to the new kernel as a part of the processes context. Assuming that the processes communicate more frequently than the migrate, any message will be delayed by a single hop only. By addressing every message through the kernel instead of the link owner directly transparency is ensured [6]. Where necessary, a linear searching algorithm can be employed, though such occasion will not be frequent. Having communication through kernels has another advantage. The

relevant kernels know which processes are owners of links, and when migrating them keep a record for forwarding messages. After the first redirection, both the sending and receiving kernels are informed and the record deleted. Thus space is not wasted.

3.3 DISTRIBUTION OF PROCESS MANAGEMENT FUNCTIONS

In a centralised systems, process management tasks are generally shared by three entities [6]. The Traffic Controller keeps track of the process state and resource requirements. Which process is to be actually loaded and started at a given instant is decided by the process scheduler or Local Process Scheduler (as we shall be referring to it). Assigning the required resources to the process and starting it is the task of the Dispatcher.

A Distributed system requires a more complex process management scheme. Processes not only have to be managed locally within a node but also globally - requiring a system-wide approach. It is convenient here, to split the management scheme into two major segments. One segment takes care of local scheduling and will consist of the three entities mentioned earlier. Global management deals with deciding whether the process is runnable at the current node, packing and sending a process to another node or

receiving one and including it in the local process list. The global management scheme is the one that is relevant here.

There are two ways in which a node can receive a process. Some migrate to it through the bidding and scheduling methods - that is through negotiation. Failing nodes on the other hand might randomly distribute their processes, the main concern being preventing the loss of a process rather than their efficient distribution. Processes falling in the former category are guaranteed [17], their resources and processor time being assured. Randomly migrated processes, however cannot be guaranteed and might require to be migrated further before they find a niche.

A possible configuration for a process manager is the one shown in Fig. 5. The process manager exists in two parts. Local process scheduling function are carried out in each node as usual. The traffic controller acts as a go-between for the local and global management functions as it keep track of internal resource requirements and assesses process characteristics provided by external nodes. To migrate a process that is resident in the node, the traffic controller and scheduling algorithm cooperate. Thereafter the bidder is instructed to send the

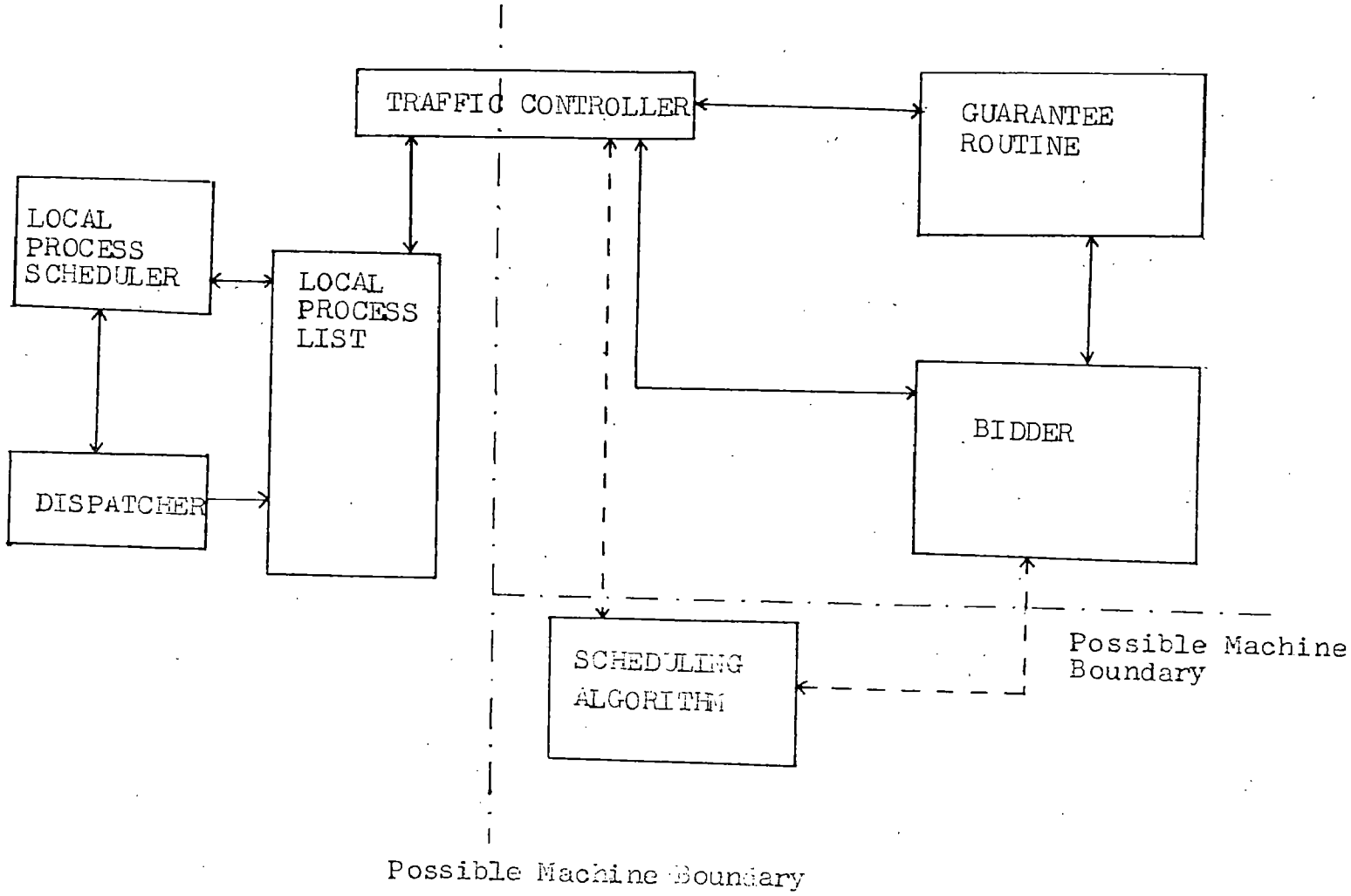


Fig. 5 Distribution of Process Management Functions.

process to a suitable location. Processes migrated to a node are checked by the bidder to see whether they have been already accepted. Processes that have been bid for are marked as guaranteed. They are then placed in the local process list. If an unbid process arrives, the guarantee routine in conjunction with the traffic controller examines it. Depending on the availability of the resources, it can be guaranteed and included in the local process list or marked for migration and returned to the bidder. Bid requests sent by other nodes are processed by the bidder with the traffic controller and scheduling algorithms.

The picture presented above is a very general one. Different operating systems would implement the configurations in different ways. In a totally decentralised system the complete structure would be replicated in each node together with an information collection algorithm to support dynamic scheduling. However a system that broadcasts bid requests instead of sending then to specific nodes would have no need for scheduling or information collection algorithms. Conversely, a system can depend wholly on a scheduling algorithm to specify a destination and thus eliminating the bidding routines. Another criteria for the configuration is the amount of decentralisation permitted by the system. Only the local process manager might be replicated, while a partially centralised process combines the bidder and

guarantee routine. This process could either be a self migrating one or could perform the operation remotely. The scheduling algorithm can be run as a separate entity, possibly on more than one, but not all nodes, and in constant communication with the bidding process. In fact, the scheduling algorithm would preferably be a part of the information collection routine.

An object-oriented system would further modify this. Instead of having the local process list as a separate data structure, it would form a part of an object. Most of the operations on this list are performed by the traffic controller, so it would be logical to include the functions of the traffic controller in this object. The local process scheduler and dispatcher would be objects themselves and operate by exchanging messages with the process list object.

3.4 PHYSICAL TRANSPORTATION OF A PROCESS

After making decisions as which processes to migrate and where, the final task of actually moving the process remains. This transportation requires the entire process context to be saved and sent to the destination where it will be reloaded and started again. A process's context consists of its entire object code, data segments and allocated system facilities.

The operating system recognises a process by a logical number or handle [18]. Associated with this identifier is a table that lists information about the process and which is used by the operating system [6]. This table is called the Process Descriptor List (PDL)[8] or, in an object oriented system, the Process Work Object (PWO)[12]. In case of processes grouped together, like the task forces of MEDUSA [8], another list called the Shared Descriptor List (SDL) may be present to describe the group characteristics. These descriptors are useful when a process has to be migrated. For scheduling purposes the information contained in such descriptors provides a picture of the requirements of a process.

The object code, data and system utilities used by a process can be called its working set [18]. Depending upon the implementation, the Process Working Set (PWS) might be organised as files, processes pages, segments, objects or a combination of these. For the purpose of migration, the PWS has to be saved in a transmittable form. Typically the PWS of a process is much larger than its PDL. Thus, whereas the latter can easily be fit into a message, transporting the former presents quite a few problems. Breaking up the PWS into many separate messages and transmitting them in packets is possible, but the overheads tend to increase due

to the large number of messages involved. Another solution is to dump the whole PWS into one or more files and use a File Transfer Protocol for moving the file. File specifications can be sent through an ordinary message packet. Some systems provide a sort of switchable DMA link, using which it could be possible to transfer the complete PWS at one go. A modification of this DMA is used by MEDUSA, the Cm* operating system. In MEDUSA the facility for memory sharing between processors is provided [8]. By appropriate switching of interconnection through K maps the destination processor can connect to the source computer's memory. The PWS can then be copied. It should be mentioned here that though two processors can share their memories, it is not possible for a computer to execute a remote object code as the connections exists for very short periods. In fact, it is this restriction that causes Cm* to be classified as a distributed system rather than a multiprocessor, where memory can be shared at the instruction level.

Fig. (6) shows how process migration takes place in DEMOS/MP [5]. The source processor (M# 1) is running a process A originally. Desirous of downloading itself, or for any other reason, it initiates proceedings by sending a bid request to machine M # 2. This node, finding enough free resources to accomodate A, responds with an acceptance.

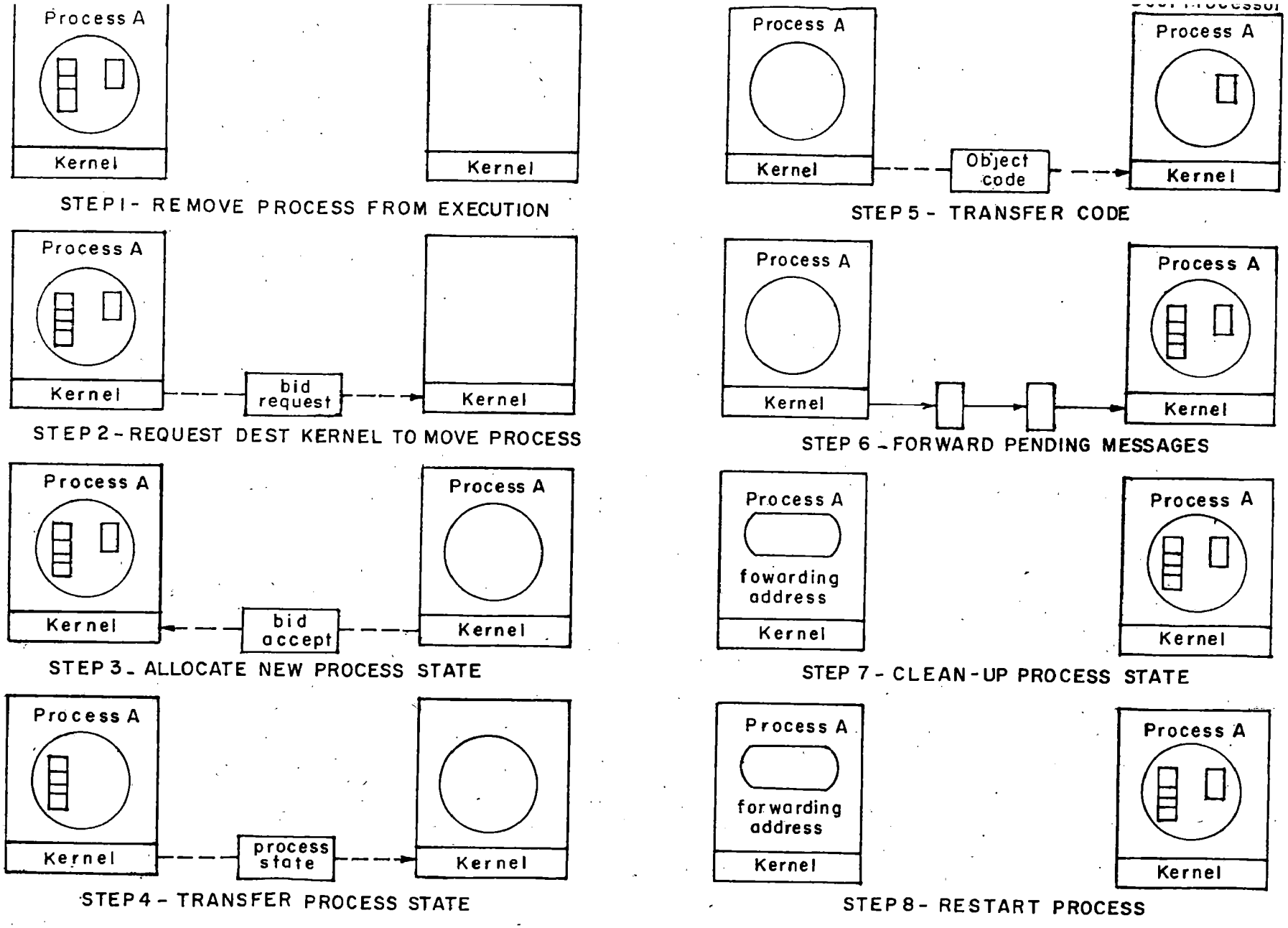


FIG.6 STEPS IN MOVING A PROCESS

Machine M # 2 then creates a process shell (blank process) locally. One by one, various descriptors are sent by M # 1, received by M # 2 and placed in the corresponding tables. After sending the PWS, any messages addressed to A but not processed yet are forwarded to M # 2. Now the process in M # 1 is just an empty shell and is deleted. A forwarding address is left with ' A at machine M # 1. Execution of the process starts on machine M # 2 from the last Program Counter value.

CHAPTER - 4

A PROCESS MIGRATION PACKAGE ON THE DECSYSTEM-2050

This chapter deals with the implementation of a software environment which supports process migration on the DECSYSTEM-2050. First of all, some of the facilities provided by TOPS-20 - the DECSYSTEM-2050 operating system- have been detailed. These are the facilities that have been widely used in our implementation, and to a large extent, have decided its nature and operational features. Later sections discuss the program organisation and its working.

4.1 OVERVIEW OF TOPS-20 FACILITIES

4.1.1 The Process Structure

A Process (or Fork) is an executable entity [18]. It has its own 512-page (maximum) address space, accumulators and program counter. In TOPS-20 each process is scheduled independently of the others. The highest process is the EXEC program which is created by the system when an user logs-in. Other processes created either on the user's request or by the system are inferior to the EXEC and form a tree structure (Fig.7), from which the term fork is derived. It is clear from the tree that a process can

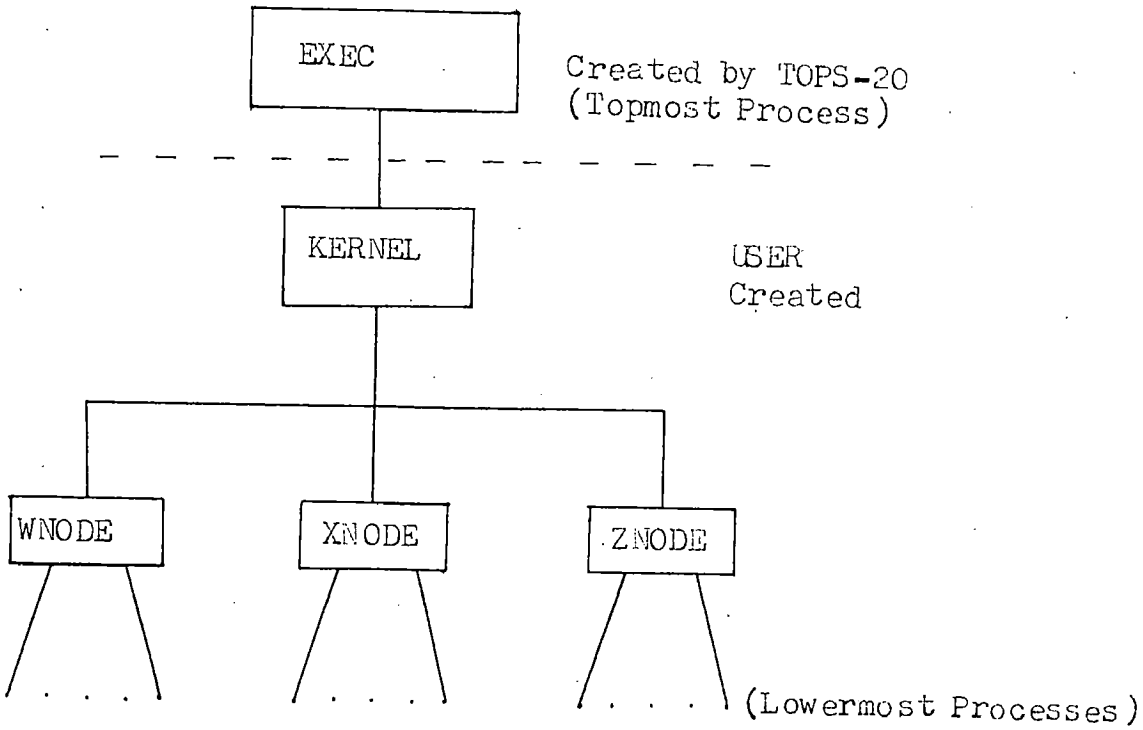


Fig. 7 Process Tree in TOPS-20

Depicts the Package Process Hierarchy

Wnode, Xnode, Znode Form the Distributed System.

The Lowermost Processes Migrate.

have only one superior but many inferiors . Processes sharing the same superior are called parallel processes, though, this does not mean that they are concurrent.

In order to exert control over its inferiors, a process be able to address then by some identifier. This identifier, in TOPS-20 is called a fork handle and is a quantity between 400000 and 400777 (both octal). Fork handles are relative. That is, they are meaningful only tb the process to which they were given.

TOPS-20 provides a JSYS (Trap to system) CFORK for creating a process. A process created in this manner is a virgin process, having no address space. Its accumulators and programm counters have not been set. The first step is to provide it an address space with an executable code. There are two way to do this. One is using the PMAP JSYS to assign a file page to the process. This method is not preferred as each page has to be handled separately. Also, the entry vector has to be properly read and set. Here, we have used the GET JSYS which can load a complete EXE file into the process address space.

A process can be started by one of the two JSYS-SFRKV or SFORK. In our implementation, SFRKV has been used to start a process the first time. SFRKV does not need any starting address to be specified as it is available from

the entry vector. For restarting a halted process from the last program counter value, SFORK has been used.

Various interrupt and Interprocess communication facilities are available to a process. These have been discussed in the following section.

4.1.2 Inter-Process Communication Facility

The Interprocess Communication Facility (IPCF) allows messages to the sent between co-operating processes [18,19]. Sender and receiver processes are identified by a unique Process Identifier (PID), which is a 36-bit quantity assigned by TOPS-20. A special system program INFO is the information centre for IPCF. This program performs functions by which names and process identifiers are associated.

To avail itself of the IPC facility, a process, first of all, has to get a PID. This can be done using the MUTIL JSYS. As the PID is system assigned, it changes every time a process is re-initialized. It is therefore difficult to initiate communication between two processes. For this purpose, TOPS-20 provides the services of INFO, which associates a PID with a user specified logical name. Communicating processes can be provided each others logical names before hand. Assigning a logical name to a PID and asking the PID of a logical name can be done by sending messages to INFO. INFO's PID can be asked for by specifying a special function code in the MUTIL JSYS.

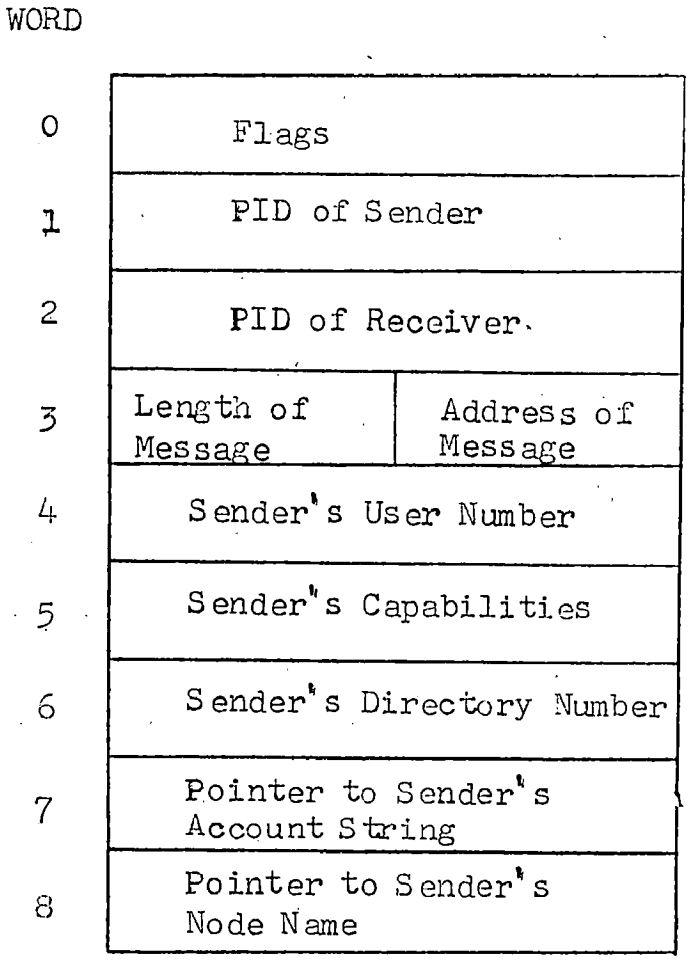


Fig. 8 Format of the Packet Descriptor

A message packet in TOPS-20 has two parts. One is an IPCF Packet Descriptor Block. Entries in this block completely identify the sender of a message. Fig. 8 shows the format of the Packet Descriptor. However only the first four words are essential, and throughout our implementation, only these have been used. The second part of the message packet is the message itself whose length and storage address in the sender's address space is specified in the packet descriptor.

Message sending and receiving are performed by the MSEND and MRECV JSYS respectively. MRECV copies the packet descriptor and message into areas specified by the receiver. A software channel assigned to a PID will cause a process to be interrupted when a message is sent to it. This facility permits the process to use its time for other functionsinstead of polling for a message.

4.1.3 The Interrupt Facility

TOPS-20 provides thirty six interrupt channels, each being a software entity associated with an event [18,19]. Some channels are permanently assigned to particular events such as file data errors, process halt, etc. Others can be assigned by the programmer for terminal interrupts, IPCF interrupt or program initiated interrupts.

Two tables provided by the programmer are referred to by the interrupt system. These are the channel table and level table, traditionally called CHNTAB and LEVTAB. The channel table has thirtysix entries (words) each corresponding to one channel. In the left half of a word, the priority (0-3) of the channel is specified and the address of the service routine in the right half. Interrupt priorities are in decreasing order from 1 through 3, 0 being a de-activated level. A high priority interrupt is permitted to pre-empt a lower one.

The level table has three entries, each specifying the PC storage location for the corresponding level. As two interrupts of the same level cannot be processed at the same time, only one storage location per level is needed.

There are various steps in setting up the interrupt system. The address of the tables CHNTAB and LEVTAB are brought to the TOPS-20's attention by the SIR JSYS. Then EIR is used to enable the interrupt systems. Interrupt channels being used by the process have to be separately activated using the JSYS AIC. This system call accepts a thirty six bit (one word) argument (one bit per channel), with the bits that are set indicating the channels to be activated.

For channels that have been permanently assigned events no further initialization is needed. User defined channels

have yet to be assigned events. For terminal interrupts the ATI system call is used. There are thirty six possible interrupts origination from the TTY. Twentysix refer to the control character CTRL/A through CRTL/Z, are the relevant ones here. One of these has been used for initiating the user Interface (discussed later).

IPC interrupt channels are assigned using MUTIL, after a PID for the process has been obtained. The third interrupt channel used by us is the process termination interrupt and is assigned to channel 19 by the system.

4.2 OPERATIONAL ORGANISATION OF THE PROGRAM

The software routines of this package fall into four independent segments. They are the Initialization Routine, the User Interface the IPC (Inter Process Communication) Handler and the Process Monitor. Each of these segments has a separate function and can operate entirely independent of the others, although not concurrently. A fifth segment contains those subroutines which are used by the other four. In a way each segment can be viewed as an object except for the shared databases. If for some reason, one segment has to initiate the services of another, it has to do so through the standard interface, and not by directly calling an internal subroutine.

On system startup, the initialization routine is automatically invoked. After setting up various databases, it leads the system (or node) into a wait state. The other three segments are interrupt driven. They become active only on receiving a specified interrupt, and return to the wait state after performing the desired functions.

Functioning as an operating system, the package supports various processes. Based on the DEMOS/MP[5] organisation, processes have been classified into two categories- user process and system processes- user processes are created on a specific request, and at a time, only one user process can exist on a node. These processes can be killed restarted or migrated at the user's discretion. System process are more priviledged. They are decided before system star up. A user is permitted to ask for the services of a system process, but cannot kill or migrate them. Only one copy of a system process can exist in the system at a time typically loaded by the node that comes up first. However, if desired,a process can be marked to be loaded by a specified node only, in which case duplication of processes may result.

System processes migrate transparently. That is, when requested by a user, it can be transported from a remote node to the current one without the user being aware of it, Similarly a failing system will distribute all its processes among the other nodes that are on line.

4.2.1 System Databases

The system databases comprise of the different tables and files that the package uses, either as a working space or for initialization. Organisation of the important databases has been discussed here.

SYSTEM PROCESS RECORD (SYSTAB): This table contains a record of all processes in the system, whether or not they are present on the current node. The structure of this table is shown in Fig. 9(a).

Out of eight words reserved per process, only four are being used in this version, the remaining are reserved for possible future expansion. The maximum number of system processes currently permitted is eight.

The process number is a code which gives information about the global process number, the creating node and the present (or last known) destination. This coding has been derived from that of DEMOS/MP. As a process moves from node to node the last known destination entry in the code changes. In the case of system processes, the creating node entry is useless, as the global process number is unique. It has been included however, for compatibility with the numbering systems employed for user created processes.

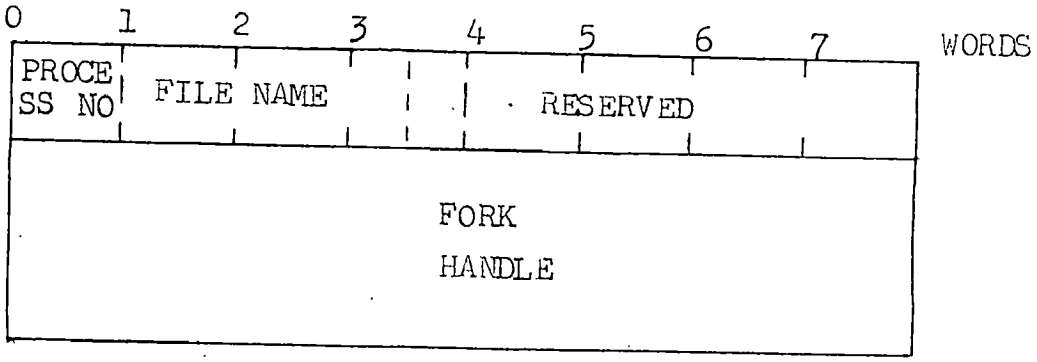


Fig.9(a) Format of SYSTAB

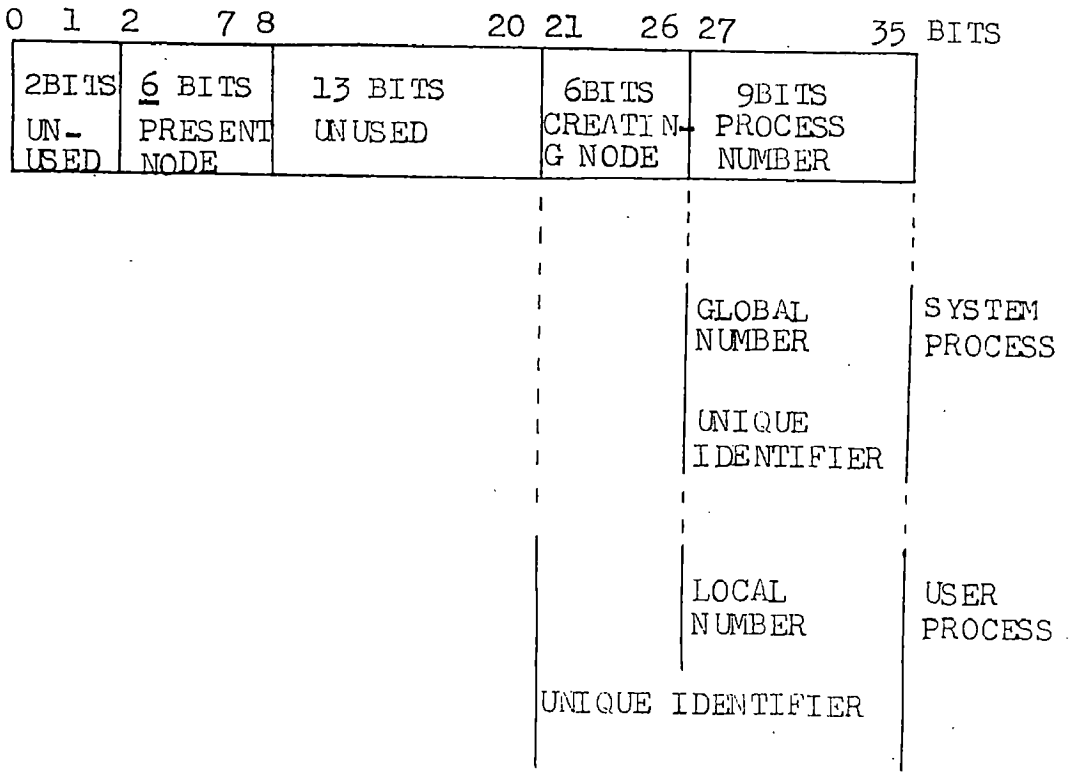


Fig. 9(b) The Global Naming Scheme.

A process name is actually the file name with the extension (.EXE) in which the executable code for that process is stored. This filename (without the extension) is the one the user will use for referring to that process .

Fork handle, as has been explained earlier, is an 18 bit sequence provided by TOP-20 and is used to refer to a process. It is used internally by the package.

USER PROCESS RECORD (CURFRK,CURPRN): As only one user process is permitted to exist at a time on one node, no elaborate arrangements for its record have been made. Two locations for the current process number and current fork number have been kept. A user process exists on a node either when a user creates it or when one migrates from elsewhere Similarly a node becomes free when the current process is killed by the user or migrated to another node. To uniquely identify a user process its local process num ber (given by the creating node) is concatenated with the creating node number (Fig.9).

NODE IDENTIFICATION RECORD: In an actual distributed system , the nodes would be in communication with each other through a network. Here this is effected by using the TOPS-20 Inter-process Communication facility. Two tables- one for the node's logical name (EPIDNM) and the other for the identification number (EPIDNO) are created. In EPIDNM, every possible

node - on line or not - has an entry. The logical name is of the form EXCn, where 'n' is an integer from 1 upwords. EXC0 is the name used to refer to TOPS-20. These logical names (except EXC0) are registered with TOPS-20, and the corresponding number allotted is stored in EPIDNO. As in EPIDNM, the node number is an index into EPIDNO. A A zero is entered against that node that has not come on line yet.

DIRECTORY OF IPC COMMANDS (MSGTAB): This is a table which contains addresses of all possible commands received through a message. The index of the table in the command code with the corresponding service routine being the entry. An advantage of using this approach is that no comparison or interpreting of the message code is required. Simple indirect indexed subroutine calls are sufficient.

DIRECTORY OF USER COMMANDS (USRTAB): Similar to the directory of IPC commands, this table is used by the user interface. Depending on the user's command, the index is assigned and an indirect jump performed.

SYSTEM INITIALISATION FILE (SYSTEM.TBL): All systems process entries are stored in this file. It is used by a node during the initialization phase. The first entry is a number denoting the total entries in the file. Subsequent entries are of the form

PROCESS CODE PROCESS (FILE) NAME

The process code is a number specifying the node which will load the particular process. If it is zero, the file is loaded by the first node coming on line. Presently no provision has been made to specify a process that would be loaded by every node.

Other less important databases are the stacks and interrupt initialization tables. Entries for recording the current node number, total number of system processes, number of nodes on line and temporary workpads do not require an explanation.

4.2.2 The Initialization Routine

System startup is initiated by this routine. After storing the system start time, it accepts the boot command and the node number. The various operations performed afterwards are listed below.

LOADING THE SYSTEM TABLE : The system file is read in the correct format and corresponding entries into the table are made. At this stage no files are loaded as the information about other nodes on line is not available.

PERFORM IPC INITIALIZATION TASKS: Before a node is ready to communicate with other nodes, it has to initialize the various process identifiers. First of all the node has to

get a PID for itself and one for <SYSTEM>INFO. Then it registers its logical name with the system so that the PID is available to other nodes. This is done by sending a message to <SYSTEM>INFO. The third phase is getting a PID for other nodes on line. To do this, one by one the logical names of other nodes are sent to <SYSTEM>INFO and the result stored. Finally the nodes that are on line are informed about this node's startup.

INTERRUPT INITIALIZATION: A node uses three interrupts. One for each of the three interrupt driven routines. All the three interrupt channels are assigned and activated except the user interface interrupt. This will be the final task in order to allow initialization to complete before any commands are accepted.

STARTING SYSTEM PROCESSES: The system process table is scanned and the processes loaded. Each process is started once and allowed to come to the first halt.

ENTER THE WAIT STATE: All initialization tasks are over at this stage. A message to this effect is printed and the user interface activated. There after the node enters a wait or 'sleep' state.

4.2.3 The User Interface

The user interface is that part of the software which accepts commands from the user. To initiate this segment

(or to call the attention of the systems), the user has to press a specified control character on the keyboard. Commands are given in the form of numbers which are listed in the command menu. This approach has been taken to make the implementation and use of the package simple.

Commands that the user can give are

CREATE PROCESS (CREATE): Creates a user process from an EXE file specified by the user. Any previously existing process is automatically killed.

KILL PROCESS (KILLPR): The current user process is killed and the corresponding entries deleted.

MIGRATE PROCESS (MGRATE): Migrates the current user process to a specified node. Gives error messages if no process is running, or no other node is on line. Another possible error is the specified node not being free. Actually this routine just sends a bid request to the specified node, and exit after marking the node status as busy. Further processing will be taken up by the IPC handler. Thus if the remote node accepts the process, it sends a reply to the bidding node. This reply is received by the IPC handler which will pack and migrate the process. In case the remote node is occupied , an error message is printed. Any of these responses terminate in automatically initiating an interrupt on the user interface channel. While bidding is in operation, the user will not be able to give a new command.

GET A SYSTEM PROCESS (GETSYS): On initiation this routine prints the system process list and corresponding global process number. After the user selects the required process, a search for the process is made internally. If it is present it is run, otherwise a request is sent to the last known destination if possible, else a linear search is made. As before the user interface exits, marked 'busy', and process migration occurs or is refused Re-entry to the user interface is automatic.

RESTART CURRENT PROCESS (RESCPR): Restart the current user process. A user process terminates either voluntarily or is forced to do so by interrupting from the key board.

LOGOUT FROM THIS NODE (LOGOUT): An exit from the user interface is made, and the node goes into an idle state. It however remain on line and can be awakened by pressing the specified control character from the keyboard.

SYSTEM STATUS (SYSTAT): Lists the number of nodes on line and the status (online, offline) of each node.

EXIT FROM SYSTEM (SYSEXT): The node goes offline, effectively halting. Before doing so, it informs all the other nodes of its intention and migrates any system processes. In an actual system this would be a privileged command accessible only at the operator level. To give this effect, here it demands a password.

4.2.4 The IPC Handler

TOPS-20 provides the facility of assigning an interrupt channel to a PID . Then, any message sent to that PID will cause the specified interrupt service routine to be initiated. In our implementation, a message can be sent either by the user Interfaoe or by the IPC Handler, but is received by the latter only. A standard message structure has been decided upon. The first word always specifies the sending and receiving node numbers, followed by the message code. Arguments if any are sent in the following words. The size of the message block is not fixed as different codes are accompanied by different argument lengths. This, however, causes no problems as the format of arguments for every code is rigidly decided, and interpreted accordingly. The message code acts as an index into the table of IPC service routines. The routines are

SELF COMING UP (SLFUP;CODE=1): A new node coming up sends this message to all other nodes that are already on line during its initialization phase. On receiving such a message, an entry in the PID table is made.

SELF GOING OFF-LINE (OFFLN; CODE =2): This message is sent by a node that is going down, to all other nodes that are on line. The action taken against this message is the deletion of the PID entry.

REQUEST FOR SYSTEM PROCESS (RQSYS: CODE=3): A request for a system process, the number of which is given in the first argument word. The receiving node searches its table for the system process. If it does not have that process, the message is forwarded. In case the required system process is not available, this request will ultimately reach the node from where it originated. This node recognising it will inform the user of non-availability of the processand terminate the message. To avoid redundant message forwarding a special convention for the header has been adopted. When a node forwards a message, it does not change the entry for the sending node number. Thus this entry always points to the node where the message originated and not to an intermediate node. Any positive action, like migrating the process is done directly and not by backtracking.

SENDING SYSTEM PROCESS (SYSPR: CODE=4): A process description block is accompanied by this message when a system process is being migrated. The receiving node will load and start the system process.

The four message codes that will be discussed now, form the bidding messages for process migration. As no scheduling algorithm has been implemented, the bidding is initiated manually by the user.

REQUEST FOR SYSTEM PROCESS (RQSYS; CODE=3): A request for a system process, the number of which is given in the first argument word. The receiving node searches its table for the system process. If it does not have that process, the message is forwarded. In case the required system process is not available, this request will ultimately reach the node from where it originated. This node recognising it will inform the user of non-availability of the processand terminate the message. To avoid redundant message forwarding a special convention for the header has been adopted. When a node forwards a message, it does not change the entry for the sending node number. Thus this entry always points to the node where the message originated and not to an intermediate node. Any positive action, like migrating the process is done directly and not by backtracking.

SENDING SYSTEM PROCESS (SYSPR; CODE=4): A process description block is accompanied by this message when a system process is being migrated. The receiving node will load and start the system process.

The four message codes that will be discussed now, form the bidding messages for process migration. As no scheduling algorithm has been implemented, the bidding is initiated manually by the user.

TAKE A (USER) PROCESS (TAKPR, CODE=5): When the user desires to migrate the current process, the node sends this message to the specified node. The receiver will either send an accptance or a rejection, depending upon whether a free slot is available.

ACCEPTING THE (USER) PROCESS (ACCPR; CODE=6): The response to a bid request when accepted by a node. The node marks the process record as occupied.

NOT ACCEPTING THE (USER) PROCESS (NAKPR; CODE=7): Refusal of the bid. A message to this effect will be flashed to the user who initiated the bid.

SENDING A (USER) PROCESS (SNDPR; CODE=10): When a bid request is accepted, the bidding node sends the process descriptor block prefaced by this code, and the process migrates.

OFFLOADING (SYSTEM)PROCESSES (OFFLD, CODE=11): A system that is going down, uses this code with any system process it might migrate. Response to this code on the receiver side is the same as SYSPR, except that the process is not started.

4.2.5 The Process Monitor

The role played by this routine is small but necessary due to synchronisation considerations. An interrupt is initiated on the assigned channel of this routine whenever

a process halts either voluntarily or forcibly.

First of all the values of the current process number and current process fork are tested. If they are zero, the process halted was a system process and no action is taken. Non-zero values of the mentioned location implies the termination of a user process. The status of the process is read to check whether the termination was voluntary or forced. A forced termination needs no followup action, as the user interface itself had caused the halt. Only in the case of a voluntary user process halt is a forced initiation of the user Interface necessary. This is so, because a process termination must be followed by the operating systems entry into the command phase. The user Interface is activated by causing a software interrupt on the channel assigned to it.

4.2.6 Common Routines

This last segment contain those routine that are used by the other segments.

PACK A PROCESS (PACKPR): Packs a process into a transportable form. This is the last stage in migrating a process from a node. The packed form is the one that is ultimately sent by the bidder.

PACKPR accepts the process number in accumulator X and the process handle in W. The process is then packed into a process descriptor and a file. Fig. 10 shows the

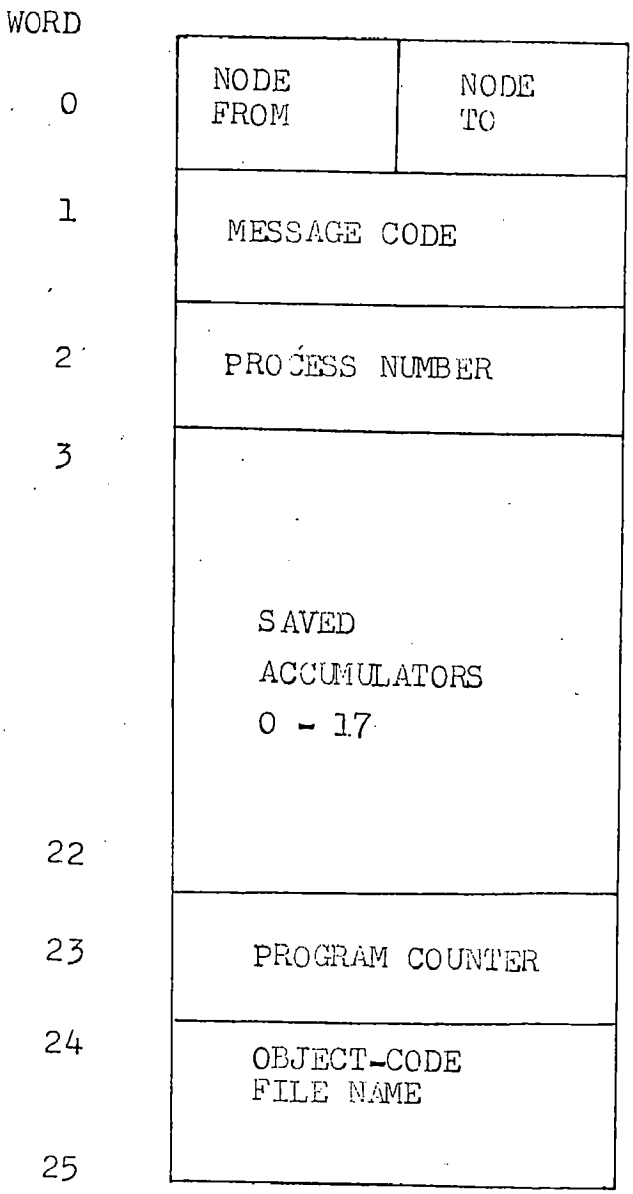


Fig.10 Format of Process Descriptor Packet
(Word Numbers are Octal)

format of the process descriptor. The file contains the saved executable code and variables of the process. A problem that arises here is that of keeping the file - name unique for every migration instance. For this purpise a special format for the filename has been decided upon. The file name is of the form MAPNM.MAP. 'N' and 'M' are variable quantities and are dynamically set by the routine. 'N' is the node number which is migrating the process. 'M' is an octal digit (between 1 and 7). Starting with 1, the number increases through 7 and returns to 1. As the receiving node deletes the file immediately after loading it, duplication of filename is avoided.

The process descriptor is sent as a message, while the file would be managed by a File Transport Protocol. Here, all nodes operate from the same directory, so no FTP is required.

LOAD A (MIGRATED)PROCESS (LDPROC): The reverse of packing, this routine unpacks the descriptor, creates a process, and loads it. The MAP file is then deleted. Returns the new fork handle in accumulator W and the program counter value in X.

GET THE NEXT PID (NXTPID): Scans the table EPIDNO sequentially and returns the value of the next node on line and its PID. It will, in no case, return the PID of the current node for a result. By setting accumulator 'Q' the search can be restarted from the beginning of the table after its end has been reached. A negative value means that the end of table has been reached with no returnable value.

SEND A MESSAGE (SNDMSG): Instead of the other routines performing a direct message send, this routine, is called after the message has been fully compiled. The reason for this is that a few processable errors can occur during a message send. Most important of these is the message queue being full. If such an event occurs, a constant polling is done until the message can be sent.

GET THE ERROR NUMBER (ERRNUM): Used for getting the error code in case of a processable JSYS error.

ERROR MESSAGE (ERLEVn): Non-processable JSYS errors are intercepted and the system error message, with the program counter value is flashed. Errors at different levels cause jumps to different location, however the result phase is the same.

4.3 USING THE PACKAGE

The program was designed so that it could run on different terminals (and hence different jobs), so that it would closely resemble a distributed systems as far as process migration is concerned. For this purpose IPCF capability, a special TOPS-20 privilege is required. Without IPCF capability it is not possible for processes created by different jobs to communicate. As this capability was not provided such a test cannot be carried out.

To run the package on one terminal, a special program (called KERNEL here) has been written, which takes an EXE file and loads it into a process. Copies of the EXE version of the package with different terminal interrupts were created. At a time only two copies of the package can be run as the system does not permit a job more than two PIDs.

On startup, the package (called node hereafter) will respond with a message and the date. It then prints an asterisk, and waits for the Boot command, which is given by pressing the key 'B'. No carriage return -linefeed sequence (CRLF) is needed. Next it asks for the node number. Again no CRLF is required. If any system processes are to be loaded by this node, their file names will be printed out. The node then informsthe user that it is ready and enters a wait state.

Pressing the specified control character on the terminal will generate an interrupt to the user interface, which asks for the command. As the command structure is menu based, the user need only press the command number, not followed by a CRLF. For the command menu itself either 0(zero) or the question mark may be pressed. For system processes, when the command (no.4) is given a menu of system processes is printed and the user can choose a desired process.

In the present version a maximum of three nodes are permitted. In case more are desired, the value of.MAXND in the file SYM.MAC should be changed. On compilation, this file will produce a universal file called SYMBOL.UNV, which contains all the important symbols used in the package.

Before booting a node, the following points must be clearly considered. The system process file SYSTEM.TBL must exist and be in the correct format. No arrangements have been made for processing data file errors. All processes must be loaded from EXE files only. An EXE file can be created by loading a program and then giving the SAVE (TOPS-20) monitor command. It is better to give a RESET monitor mode command before startup so that any assigned PIDs may be released. Two nodes must not be assigned the same terminal interrupt, as it leads to an unpredictable behaviour.

As a final comment, we are emphasising the difference between commands 6(Log off from this node) and 8(log off this node from the systems). The former sends the user interface back into the wait state. However, the node remains on line and can be awakened by the keyboard interrupt. Command 8 causes the system to go off line, by migrating its systems processes and executing a halt. The system then flashes a last message indicating that it has been shut down. A shut down once initiated cannot be revoked.

CHAPTER - 5

CONCLUSION

5.1 SUMMARY AND RESULTS

The package developed here is actually the upper level process manager of a Distributed Operating System. As we were concerned only with the process migration aspect of a distributed system, it was not necessary to widen the scope of the program any further. Among the facilities given to the user, are the routines for process creation, migration and termination. Transparent system process migration, which is essential in a distributed system, has been implemented. All interprocess communication occurs in the background, without the user being aware of the network, which again, is a plus point.

The program runs in the USER mode of TOPS-20 instead of the EXECUTIVE- replacing the EXEC- and, hence certain restrictions automatically apply to its performance. It would have been desirable to modify the existing system calls, or at least add to them in a manner that would provide a greater freedom to our implementation. Thus, all the resource management functions are left to the host computer, as there is no way by which the package can perform them. The only area where our environment has any control is deciding which process is not to be scheduled. This is the reason for our not implementing any scheduling algorithm.

It is obvious that a node behaves more like a personal computer than a time-sharing one. That is, one and only one process, in a node, can be active at a given instant. This has been done so that the confusion created by many processes reading from and writing on the same terminal is avoided.

Appendix-B lists three printouts showing how the package would respond if the nodes communicated on different terminals. This has been done by directing the output of a node to a file instead of the terminal. At every instant the time has been output so that the sequence of events is distinguishable. Notice that only two nodes are on line at a given instant because of TOPS-20 restrictions.

Despite these restriction, the system worked quite well and gave satisfactory results.

5.2 SUGGESTIONS FOR FUTURE WORK

Some problems associated with process migration have been left untouched. One of these is directing the output of a migrated process to the correct node. For this, some way has to be found to intercept or simulate a print command. A possible method is to force an interrupt on a channel that has been reserved by the superior process. This causes a halting of the inferior process and an

interrupt to the superior. The main problem associated with such a method is accessing the address space of the inferior process. If the superior and inferior share the same address space, this access is possible, but then the inferior cannot be mapped to another file or process and hence cannot be migrated. This problem has not been further persued.

Another possible addition is a scheduling algorithm. This can be done by simulating resource demands. Similar simulated resources can be assigned to every node. It would be interesting to study the behaviour of different scheduling algorithms in centralised, distributed and replicated organisations.

REFERENCES

1. Tanenbaum, A.S., and Van Renesse, R., 'Distributed Operating Systems', ACM Computing Surveys, 17(4), Dec.85, pp. 419-470.

2. Watson, R.W., and Fletcher, J.G., 'An Architecture for the Support of Network Operating System Services', Computer Networks, 4(1980), pp. 33-49.

3. Tanenbaum, A.S., COMPUTER NETWORKS, Prentice-Hall of India Pvt.Ltd., New Delhi, 1985.

4. Enslow, P.H.Jr., 'What is a Distributed Data Processing System', IEEE Computer, 11(1), Jan.78, pp. 13-21.

5. Powell, M.L., and Miller, B.P., 'Process Migration in DEMOS/MP', 9th Symposium on Operating System Principles, 1983, pp. 110-119.

6. Madnick, S.E., and Donovan, J.J., OPERATING SYSTEMS, McGraw Hill Book Company, 1902.

7. Lazowska, E.D., et al, 'The Architecture of the EDEN System', 8th Symposium on Operating System Principles, 1981, pp. 148-159.

8. Ousterhout, J.K., Scelza, D.A., and Sindhu, P.S., 'MEDUSA: An Experiment in Distributed Operating System Design', Communication of the ACM, 23(2), Feb. 80, pp. 92-104.

9. Wilkes, N.V., and Needham, R.M., 'The Cambridge Model Distributed System', Operating Systems Review, 14(1), Jan.80, pp. 21-29.

10. Brownbridge, D.R., et al, 'The Newcostle Connection- An Unixes of the World Unite', Software- Practice and Experience, 12(12), Dec.82, pp.1147-1162.

11. Walker, B., Kline, C., and Thiel, G., 'The LOCUS Distributed Operating System,' 9th Symposium on Operating System Principles, 1983, pp. 49-70.

12. Tanenbaum, A.S., and Mullender, S.J., 'An Overview of the AMEOBA Distributed Operating System', Operating Systems Review, 15(3), Jul 81, pp.51-64.

13. DECSYSTEM-20 Monitor Calls User Guide, Digital Equipment Corporation, Bedford, Masachussets, 1981.

14. Barak, A., and Shiloh, A., 'A Distributed Load Balancing Policy for a Multi-computer', Software-Practice and Experience, 15(9), Sep.85, pp.901-913.

15. Stankovic, J.A., and Sidhu, I.S., 'An Adaptive Bidding Algorithm for Processes, Clusters and Distributed Groups, 'Proceedings of the 4th International Conference on Distributed Computing Systems, 1984, pp. 49-59.

16. Miller, B.P., and Presotto, D., 'XOS: An Operating System for the X-TREE Architecture', Operating Systems Review, 15(2), Apr 81, pp. 21-32.

17. Ramamritham, K., and Stankovic, J.A., 'Dynamic Task Scheduling in Distributed Hard Real Time Systems', 4th International Conference on Distributed Computing Systems, 1984, pp. 96-107.

18. Gorin, R.E., INTRODUCTION TO DECSYSTEM-20 ASSEMBLY LANGUAGE PROCRAMMING, Digital Equipment Corporation, Bedford, Masachussets, 1981.

19. DECSYSTEM-20 Monitor Calls Reference Manual, Digital Equipment Corporation, Bedford, Masachussets, 1981.

20. DECSYSTEM-20 Macro Assembler Manual, Digital Equipment Corporation, Bedford, Masachussets, 1981.

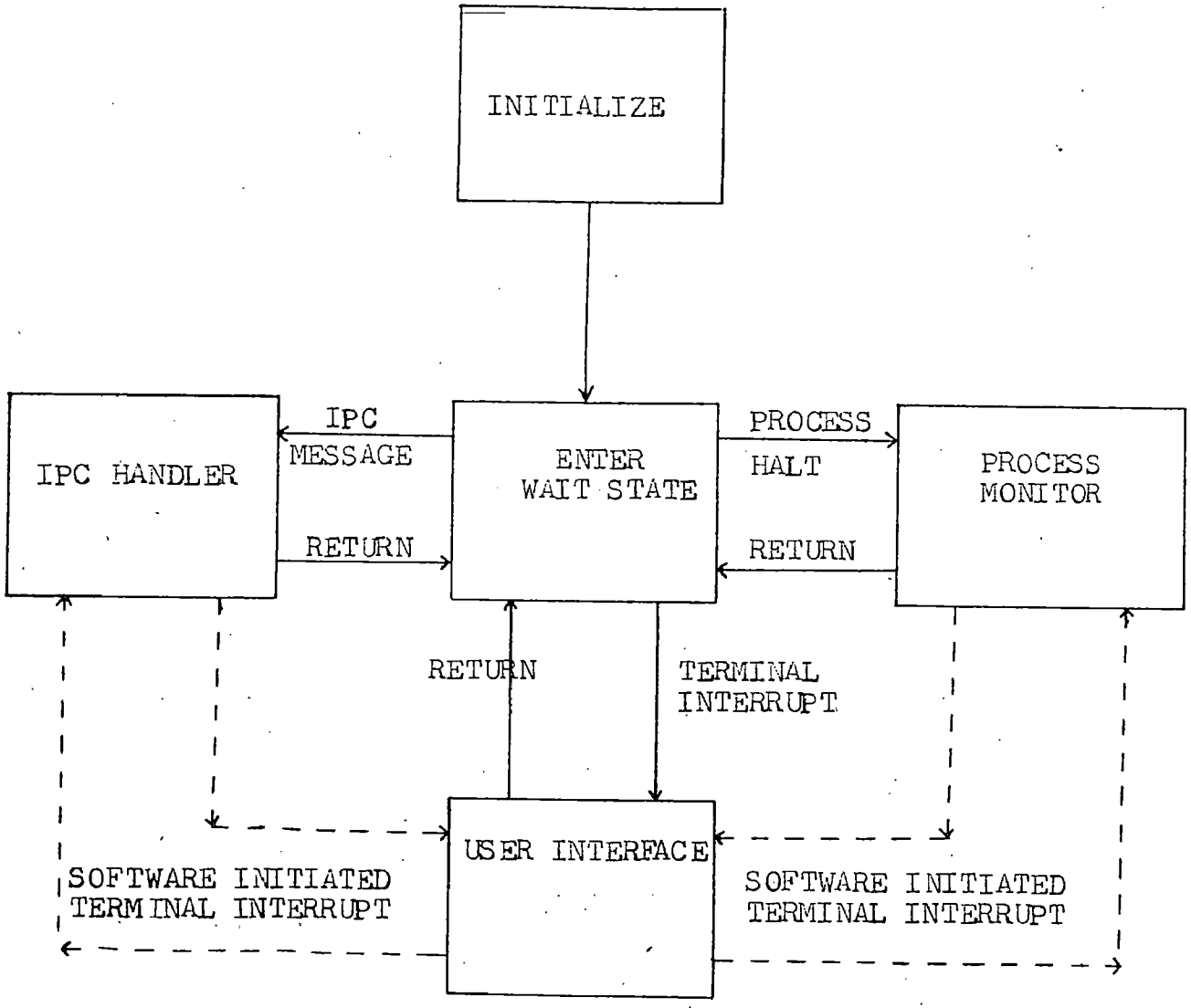


FIG.A-1 Flow of Control in the Package

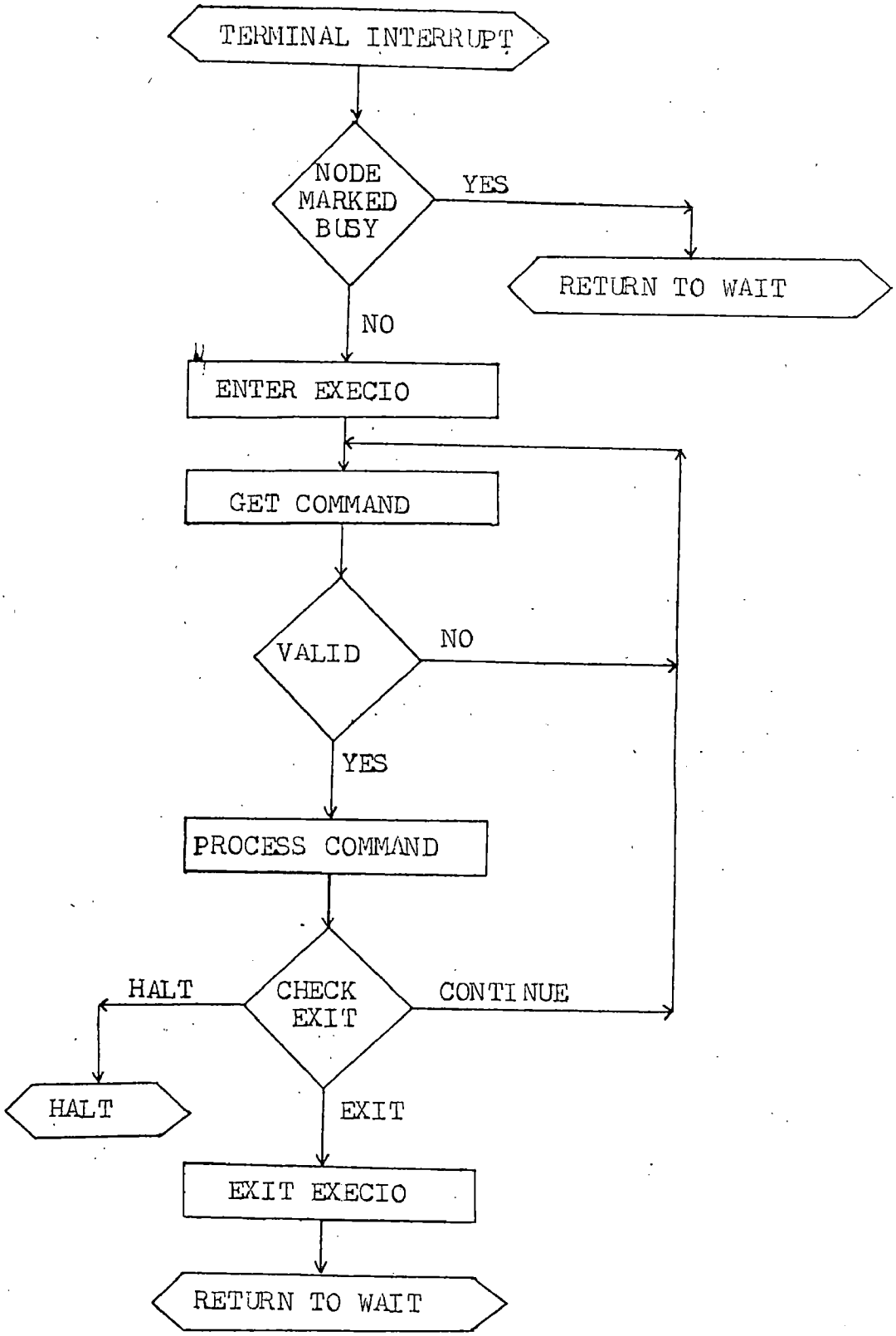


Fig.A-2 Servicing a Terminal Interrupt

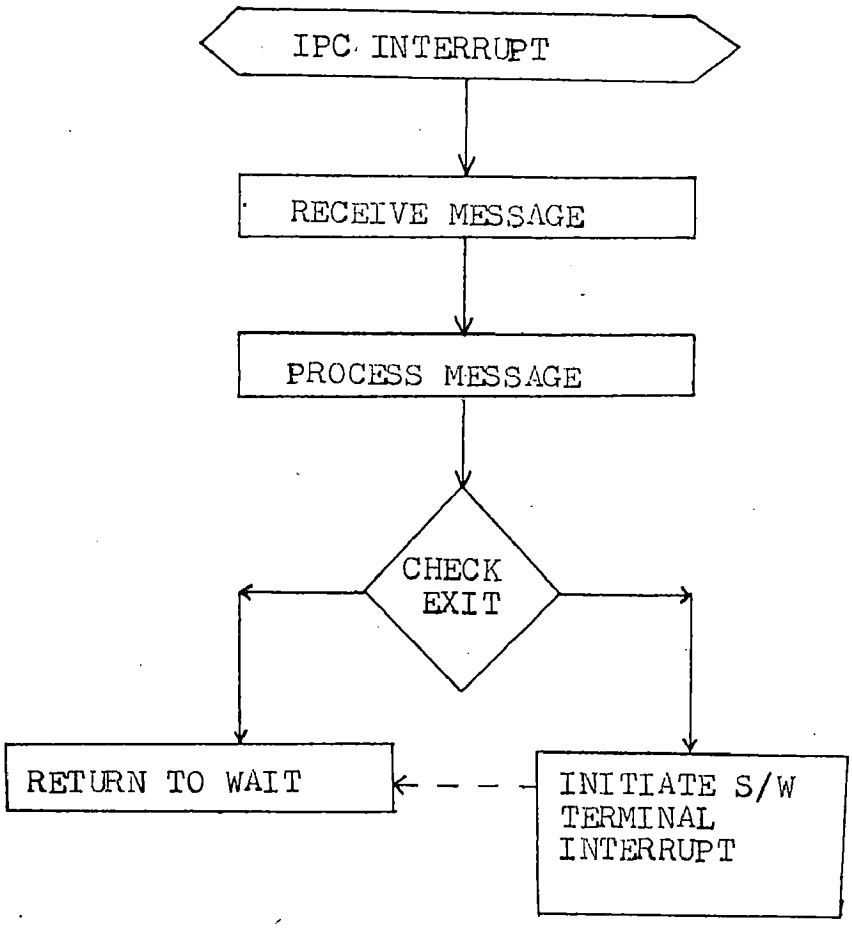


Fig. A-3 Servicing An IPC Interrupt.

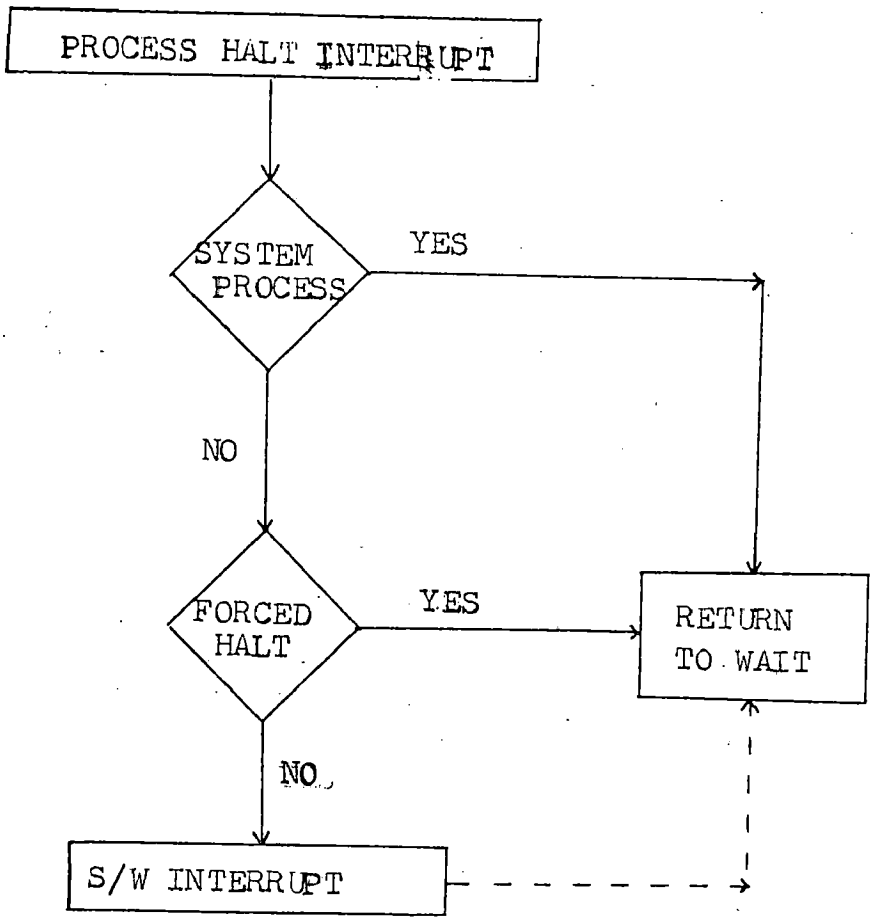


Fig. A-4 Servicing a Process Halt Interrupt

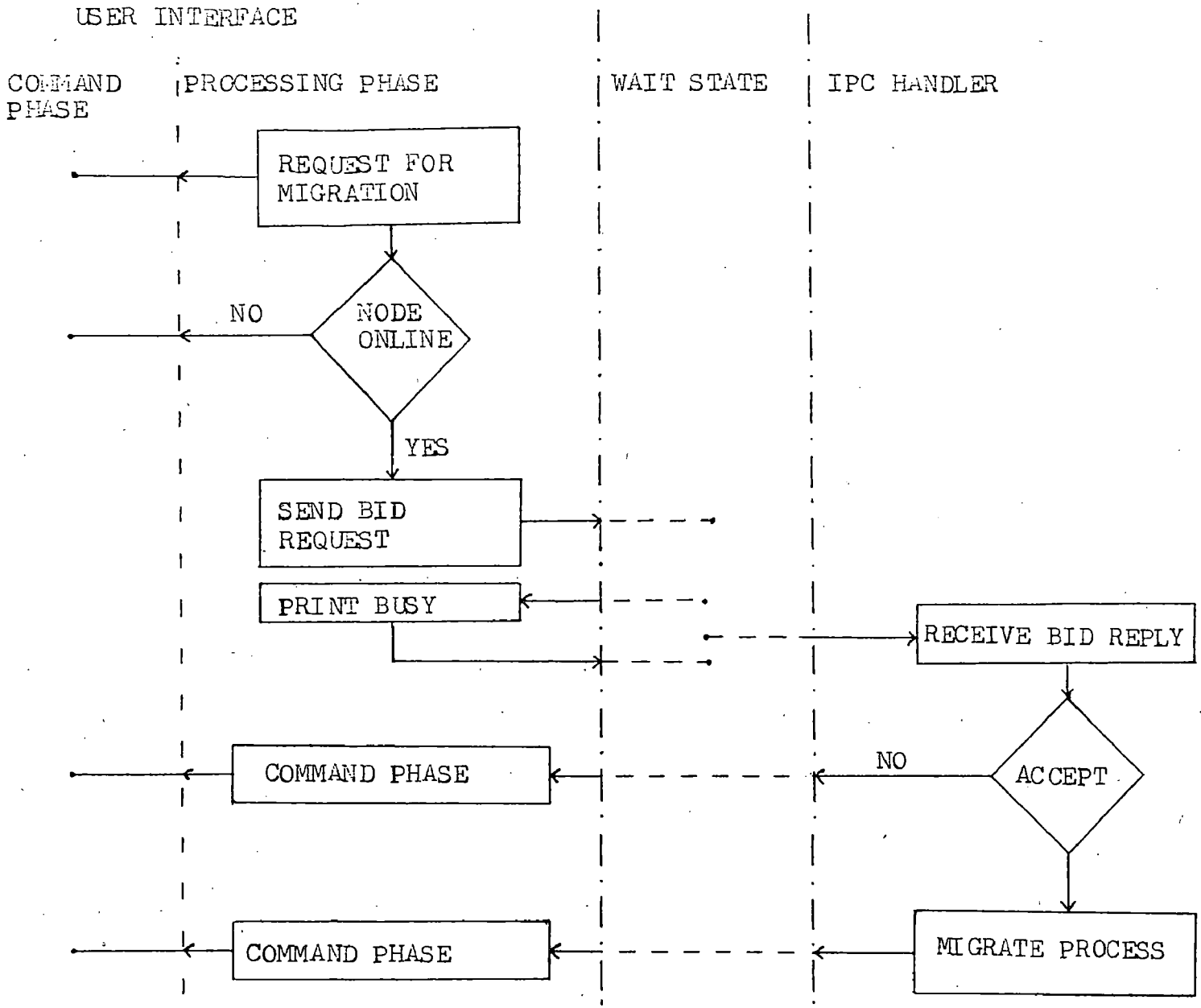


Fig.A-5 Flow of Control During a Bid for Process Migration.

```
;*****************************************************************
;    A PACKAGE FOR PROCESS MIGRATION
;         ON THE DECSYSTEM-2050
;*****************************************************************


          TITLE ZNODE
          SEARCH  MONSYM
          SEARCH  MACSYM
          SEARCH  SYMBOL
          .REQUIRE  SYS:MACREL


STKLEN==          20

SYSTAB: BLOCK   100
STACK1: BLOCK   STKLEN
STACK3: BLOCK   STKLEN
EPIDPO: BLOCK   .MAXND+1
SETIME: BLOCK   16
IPCB:   BLOCK   10
TABLE:  BLOCK   43
EPIDPN: ASCIZ/EXCJ/
          BLOCK   .MAXND
TAPELL: BLOCK   2
ACTAB3: BLOCK   20


MSGTAB: ?,,NONXCT
          ?,,SLEEP
          ?,,OFFLN
          ?,,NOSYS
          ?,,SYSPR
          ?,,TAKPR
          ?,,ACCPR
          ?,,HARPR
```

```
                0,,SNDPR
                0,,OFFLD

USRTAB: 0,,CMMENU
                0,,CREATE
                0,,KILLPR
                0,,MGRATE
                0,,GETSYS
                0,,RESCPR
                0,,LOGOUT
                0,,SYSTAT
                0,,SYSEXT
                0,,NODXCT

CHNTAB: 0
                3,,USRINT
                2,,MSGINT
                BLOCK CHNTAB+^D19-.
                1,,PRHINT
                BLOCK CHNTAB+^D36-.

DEVTAB: 0,,PCLEV1
                0,,PCLEV2
                0,,PCLEV3

CURPRK: 0
CURPRN: 0
FLGILC: 0
FLGSYS: 0
FILEXT: ASCIZ/.MAP/
FILNAM: ASCII/MAP00/
FILNUM: 0
LOGFIL: ASCIZ/SYSTEM.TBL/
```

```
NNODES: 2
NSYSFL: 0
PASSWD: ASCII/NOPAS/
PCLEV1: 1
PCLEV2: 0
PCLEV3: 0
PROCNO: 4
SECOND: 0
STBASE: 0
SELFRO: 0
STCODE: 0

MENU:ASCIZ/
                    COMMAND MENU
                    ------------

          COMMAND NO.              COMMAND DESCRIPTION

              0                PRINT COMMAND MENU
              1                CREATE A PROCESS
              2                KILL CURRENT PROCESS
              3                MIGRATE CURRENT PROCESS
              4                RUN SYSTEM PROCESS
              5                CONTINUE CURRENT PROCESS
              6                LOG OFF FROM THIS NODE
              7                SYSTEM STATUS

(PRIVILEGED COMMANDS -- PASSWORD NEEDED )

              8                LOG OFF THIS NODE FROM SYSTEM

<END OF COMMAND MENU>
/

MESSG1: ASCIZ/DECSYSTEM-2050 PSEUDO DISTRIBUTED OPERATING SYSTEM/
```

```
01120 ;*************************************************************
01130 ;       THE   INITIALIZATION ROUTINE
01140 ;*************************************************************
01150
01160 STARTX: RESET
01170         HRROI   A,STTIME
01180         MOVNI   B,1
01190         SETZ    C,
01200         ODTIM
01210         TIME
01220         MOVEM   A,SECOND
01230         MOVE    P,[IOWD STKLEN,STACK1]
01240         CRLF
01250         CRLF
01260         HRROI   A,MESSG1
01270         PSOUT
01280         CRLF
01290         HRROI   A,STTIME
01300         PSOUT
01310
01320
01330 ;-------------------------------------------------------------
01340 ;         WAIT FOR THE BOOT COMMAND
01350 ;-------------------------------------------------------------
01360
01370 STRTX2: CRLF
01380         CRLF
01390         MOVEI   A,"*"
01400         PBOUT
01410         PBIN
01420         SUBI    A,"B"
01430         JUMPE   A,  STRTX3
01440         MOVEI   A,7
01450         PBOUT
01460         JRST      STRTX2
01470
01480
```

```
01490 ;-----------------------------------------------------------
01500 ;          GET THE NODE NUMBER
01510 ;-----------------------------------------------------------
01520
01530 STRTX3: CRLF
01540         CRLF
01550         HRROI   A,[ASCIZ/NODE NO.:/]
01560         PSOUT
01570         PBIN
01580         CAIGE   A,"1"
01590         JRST      STRTX3
01600         CAILE   A,.MAXND+"0"
01610         JRST      STRTX3
01620         SUBI    A,"0"
01630         MOVEM   A,SELFNO
01640         CRLF
01650         CRLF
01660         CALL      INITLZ
01670
01680 ;-----------------------------------------------------------
01690 ;   READ SYSTEM.TBL  AND STORE IT
01700 ;           IN SYSTAB
01710 ;-----------------------------------------------------------
01720
01730         MOVSI   A,(GJ%SHT!GJ%OLD)
01740         HRROI   B,INIFIL
01750         GTJFN
01760         ERCAL     ERLEV0
01770         HRRZ    A,A
01780         MOVE    X,A
01790         MOVE    B,[070000,,OF%RD]
01800         OPENF
01810         ERCAL     ERLEV0
01820         MOVE    A,X
01830         MOVEI   C,^D8
01840         NIN
01850         ERCAL     ERLEV0
```

```
                MOVE    W,B
                MOVEM   W,NSYSFL
                JUMPE   W,STRTX4+5
                MOVE    R,B
                MOVEI   S,SYSTAB
                MOVE    A,X
                MOVEI   C,^D8
                SETZ    Y,
STRTX4:         MIN
                ERCAL     ERLEV0
                MOVEM   Y,(S)
                MOVE    W,S
                HRLI    W,.PNPTR
                DPB     B,W
                HRLI    W,.CNPTR
                DPB     B,S
                MOVE    A,X
                BIN
                CAIN    B," "
                JRST      .-3
                CAIN    B,11
                JRST      .-5
                MOVE    A,X
                ERKJFN
                ERCAL     ERLEV0
                MOVE    A,X
                HRRO    B,S
                ADJ     B,
                MOVEI   C,77
                MOVEI   D,12
                SIN
                ERCAL     ERLEV0
                ADDI    S,10
                MOVEI   C,^D8
                ADJ     Y,
                SOJG    R,STRTX4
                MOVE    A,X
```

```
                CLOSF
                ERCAL   ERLEV0

;---------------------------------------------------------------
;       ASK FOR A PID FOR SELF
;---------------------------------------------------------------

                MOVEI   X,.MUCRE
                MOVEM   X,IPCB
                MOVEI   X,.FHSLF
                MOVEM   X,IPCB+1
                SETZM     IPCB+2
                MOVEI   A,3
                MOVET   B,IPCB
                MUTIL
                ERCAL     ERLEV0
                MOVE    X,SELFNO
                MOVE    Y,IPCB+2
                MOVEM   Y,EPIDNO(X)

;---------------------------------------------------------------
;       ASK FOR <SYSTEM>INFO'S PID
;---------------------------------------------------------------

                MOVX    R,.MURSP
                MOVEM   R,IPCB
                MOVX    S,.SPINF
                MOVEM   S,IPCB+1
                SETZM     IPCB+2
                MOVEI   A,3
                MOVEI   B,IPCB
                MUTIL
                ERCAL     ERLEV0
                MOVE    R,IPCB+2
                MOVEM   R,EPIDNO
```

```
02600 ;-----------------------------------------------------------
02610 ;       ASSOCIATE PID WITH A LOGICAL NAME
02620 ;-----------------------------------------------------------
02630
02640         SETZM     IPCB
02650         MOVEM   Y,IPCB+1
02660         MOVEM   R,IPCB+2
02670         MOVE    R,[3,,TABLE]
02680         MOVEM   R,IPCB+3
02690         MOVE    A,[1,,.IPCII]
02700         MOVEM   A,TABLE
02710         SETZM     TABLE+1
02720         MOVE    R,EPIDNM(X)
02730         MOVEM   R,TABLE+2
02740         MOVEI   A,4
02750         MOVEI   B,IPCB
02760         CALL      SNDMSG
02770         SETZ    X,
02780         SETZ    Z,
02790
02800 ;-----------------------------------------------------------
02810 ;       CHECK FOR OTHER NODES ON-LINE
02820 ;-----------------------------------------------------------
02830
02840 STRTX5: AOJ     X,
02850         CAMN    X,SELFNO
02860         JRST      STRTX5
02870         CAILE   X,-MAXND
02880         JRST      STRTX6
02890         SETZM     IPCB
02900         MOVE    R,SELFNO
02910         MOVE    W,EPIDNO(R)
02920         MOVEM   W,IPCB+1
02930         MOVE    R,EPIDNO
02940         MOVEM   R,IPCB+2
02950         MOVE    S,[3,,TABLE]
02960         MOVEM   S,IPCB+3
```

```
;----------------------------------------------------------------
;       ASSOCIATE PID WITH A LOGICAL NAME
;----------------------------------------------------------------

        SETZM     IPCB
        MOVEM   Y,IPCB+1
        MOVEM   R,IPCB+2
        MOVE    R,[3,,TABLE]
        MOVEM   R,IPCB+3
        MOVE    A,[1,,.IPCIL]
        MOVEM   A,TABLE
        SETZM     TABLE+1
        MOVE    R,EPIDNR(X)
        MOVEM   R,TABLE+2
        MOVEI   A,4
        MOVEI   S,IPCB
        CALL      SNDMSG
        SETZ    X,
        SETZ    Z,

;----------------------------------------------------------------
;       CHECK FOR OTHER NODES ON-LINE
;----------------------------------------------------------------

STRTX5: ADJ       X,
        CAMN    X,SELFNO
        JRST      STRTX5
        CAILE   X,.MAXND
        JRST      STRTX6
        SETZM     IPCB
        MOVE    R,SELFNO
        MOVE    W,EPIDNO(R)
        MOVEM   W,IPCB+1
        MOVE    R,EPIDNO
        MOVEM   R,IPCB+2
        MOVE    S,[3,,TABLE]
        MOVEM   S,IPCB+3
```

```
                MOVE    Y,L2,,.IPCIW]
                MOVEM   Y,TABLE
                SETZM     TABLE+1
                MOVE    R,EPIDNM(X)
                MOVEM   R,TABLE+2
                MOVEI   A,4
                MOVEI   B,IPCB
                CALL      SNDMSG
                CALL      ENABLE
SSS:            MOVX    A,JP%TTL
                MOVEM   A,IPCB
                SETZM     IPCB+1
                SETZM     TABLE+1
                MOVEM   B,IPCB+2
                MOVE    R,[20,,TABLE]
                MOVEM   R,IPCB+3
                MOVEI   A,4
                MOVEI   B,IPCB
                MRECV
                ERCAL     ERLEV6
                HLRZ    C,TABLE
                CAIN    C,1
                JRST      SSS
                MOVE    C,TABLE+1
                MOVEM   C,EPIDNO(X)
                JUMPE   C,.+3
                ADD     Z,
                MOVEM   Z,NNODES
                JRST      STRTX5

;-----------------------------------------------------------------
;       SET UP THE INTERRUPT STRUCTURE
;-----------------------------------------------------------------

STRTX6: MOVEI   A,.FHSLF
        MOVE    B,[LEVTAB,,CHNTAB]
        SIR
```

```
03340           ERCAL     ERLEV0
03350           MOVEI     A,.FHSLF
03360           EIR
03370           ERCAL     ERLEV0
03380           MOVEI     A,.FHSLF
03390           MOVX      B,1B1+1B2
03400           AIC
03410           ERCAL     ERLEV0
03420
03430 ;-----------------------------------------------------------
03440 ;         ASSOCIATE PID WITH INTERRUPT CHANNEL
03450 ;-----------------------------------------------------------
03460
03470           MOVX      A,.MUPIC
03480           MOVEM     A,IPCB
03490           MOVEM     W,IPCB+1
03500           MOVEI     A,2
03510           MOVEI     B,IPCB
03520           MOVEM     A,IPCB+2
03530           MOVEI     A,3
03540           MUTIL
03550           ERCAL     ERLEV0
03560
03570 ;-----------------------------------------------------------
03580 ;         CHECK FOR SYSTEM PROCESS TO BE LOADED
03590 ;         BY THIS NODE AND LOAD THEM
03600 ;-----------------------------------------------------------
03610
03620           HRROI     A,[ASCIZ/FILES LOADED/]
03630           PSOUT
03640           CRLF
03650           CRLF
03660           MOVE      W,NSYSFL
03670           JUMPE     W,STRT10
03680           MOVEI     X,SYSTAB+1
03690           MOVE      Y,[.CNPTR,,SYSTAB]
03700 STRTX7:   LDB       Z,Y
```

```
                CAMN    Z,SELFNO
                JRST    STRTX8
                JUMPL   Z,STRTX9
                ADD     Z,NNODES
                JUMPE   Z,STRTX9
                ADDI    X,10
                ADDI    Y,10
                SOJG    W,STRTX7
                JRST    STRT10
STRTX9:         HRLI    R,.PNPTR
                HRR     R,Y
                MOVE    S,SELFNO
                DPB     S,R
                DPB     S,Y
STRTX8:         CONSZ   A,(GJ%SHT!GJ%OLD)
                HRRO    R,X
                GTJFN
                ERCAL   ERLEV%
                HRRZ    R,A
                SETOM   FL%SYS
                MOVX    A,CR%CAP
                CFORK
                ERCAL   ERLEV%
                HRRZ    S,A
                HLL     A,R
                MOVS    A,A
                GAN
                ERCAL   ERLEV%
                MOVEI   Z,2
                ADD     Z,X
                HRRM    S,(Z)
                MOVE    A,S
                SETZ    B,
                SFRKV
                ERCAL   ERLEV%
                MOVE    A,S
                KFORK
```

```
                ERCAL     ERLEV0
                HRRO      A,X
                PSOUT
                CRLF
                JRST      STRTX7+6

;-----------------------------------------------------------
;       INFORM OTHER NODES OF STARTUP
;-----------------------------------------------------------

STRT1:  SETZ      X,
        MOVE      R,SELFNO
        MOVS      S,EPIDNO(R)
        HRLM      R,TABLE
        MOVEI     A,.SLFUP
        MOVEM     A,TABLE+1
        SETZM       IPCB
        MOVEM     S,IPCB+1
        MOVE      A,[2,,TABLE]
        MOVEM     A,IPCB+3
        MOVE      A,X
        SETZ      Q,
        CALL        NXTPID
        JUMPL     A,STRT11
        MOVE      X,A
        HRRM      A,TABLE
        MOVEM     B,IPCB+2
        MOVEI     A,4
        MOVEI     B,IPCB
        CALL        SNDMSG
        JRST        STRT10+3
STRT11: HRROI     A,[ASCIZ/       PSEUDOS IN OPERATION/]
        PSOUT
        CRLF
```

```
04450 ;-----------------------------------------------------------
04460 ;       CLEAR THE INPUT BUFFER AND ASSIGN A
04470 ;                 TERMINAL INTERRUPT
04480 ;-----------------------------------------------------------
04490
04500         MOVEI   A,.FHSLF
04510         MOVX    B,1B^D19+1B1+1B2
04520         AIC
04530         ERCAL     ERLEV0
04540         MOVEI   A,.PRIIN
04550         CFIBF
04560         ERCAL     ERLEV0
04570         MOVE    A,[.TICCZ,,1]
04580         AII
04590         ERCAL     ERLEV0
04600         CRLF
04610         HRROI   A,[ASCIZ/PRESS CTRL\Z FOR ATTENTION/]
04620         PSOUT
04630         CRLF
04640         CRLF
04650
04660 ;-----------------------------------------------------------
04670 ;       ENTER  THE   WAIT   STATE
04680 ;-----------------------------------------------------------
04690
04700 WAITXX: NOP
04710         NOP
04720         WAIT
04730         JRST      WAITXX
04740
04750
04760 INITLZ: MOVE    X,SELFNO
04770         HRLI    Y,.PNPTR
04780         HRRI    Y,STBASE
04790         DPB     X,Y
04800         HRLI    Y,.CNPTR
04810         DPB     X,Y
```

```
04820           MOVE    Y,[POINT 7,FILNAM,27]
04830           LDB     R,Y
04840           ADD     R,X
04850           DPB     R,Y
04860           MOVEI   A,"C"
04870           HRRI    Y,EPIDNM
04880           SETZ    X,
04890           MOVE    S,EPIDNM
04900 INIT2:    ADJ     A,
04910           ADJ     X,
04920           ADJ     Y,
04930           MOVEM   S,EPIDNM(X)
04940           DPB     A,Y
04950           CAIE    X,.MAXND
04960           JRST      INIT2
04970           RET
04980
04990
05000
05010
05020 ;*****************************************************************
05030 ;         INTERRUPT  SERVICE  ROUTINE  FOR
05040 ;           TERMINAL   INTERRUPTS
05050 ;*****************************************************************
05060
05070
05080 USPINT:   MOVEI   A,.TICCZ
05090           DTI
05100           ERCAL     ERLEV1
05110           MOVEI   A,.PRIIN
05120           CFIBF
05130           ERCAL     ERLEV1
05140           MOVE    X,STCODE
05150           CAIN    X,1
05160           JRST      USR2
05170           CAIN    X,2
05180           JRST      USR3
```

```
05190           CAIN    X,3
05200           JRST      USROUT
05210           CAIN    X,4
05220           JRST      USROUT
05230           JRST      USR3
05240 USR2:     MOVE    Y,CURPRN
05250           JUMPE   Y,USR3
05260           SETZM     STCODE
05270           SETOM     FL%IIC
05280           MOVE    A,CURFRK
05290           RFORK
05300           ERCAL     ERLEV1
05310           DEBRK
05320 USR3:     CALL      EXECIO
05330           MOVEI   A,.PRIIL
05340           CF16F
05350           ERCAL     ERLEV1
05360           MOVE    A,[.TICCZ,,1]
05370           ATI
05380           ERCAL     ERLEV1
05390           DEBRK
05400 USROUT:   HRROI   A,[ASCIZ/[BUSY]/]
05410           PSOUT
05420           CRLF
05430           JRST      USR3+1
05440
05450
05460 ;*****************************************************************
05470 ;          USER  I/O    INTERFACE
05480 ;   RESPONDS TO TERMINAL INTERRUPT CTRL/Z
05490 ;*****************************************************************
05500
05510
05520 EXECIO:   HRROI   A,[ASCIZ/        <COMMAND LEVEL>/]
05530           PSOUT
05540           CRLF
05550           CRLF
```

```
05560 EXX:    MOVEI   A,"<"
05570         PBOUT
05580         MOVEI   A,.PRIOU
05590         MOVNI   B,1
05600         HRLI    C,777000
05610         ODTIM
05620         MOVEI   A,">"
05630         PBOUT
05640         CRLF
05650         HRROI   A,[ASCIZ/COMMAND: /]
05660         PSOUT
05670         PBIN
05680         CAIN    A,"2"
05690         MOVEI   A,"0"
05700         CAILE   A,"9"
05710         JRST      EXCERR
05720         SUBI    A,"0"
05730         JUMPL   A,EXCERR
05740         MOVE    X,A
05750         CRLF
05760         CALL      DISINT
05770         CALL      @USRTAB(X)
05780         MOVE    X,A
05790         CRLF
05800         CALL      ENBINT
05810         JUMPL   X,EXEC2
05820         CRLF
05830         JRST      EXX
05840 EXCERR: NOP
05850         CRLF
05860         HRROI   A,[ASCIZ/ILLEGAL COMMAND/]
05870         PSOUT
05880 EXC3:   CRLF
05890         JRST      EXX
05900 EXEC2:  RET
05910
05920
```

```
;----------------------------------------------------------------
;          CREATE  A  PROCESS
;----------------------------------------------------------------

CREATE: HRROI   A,[ASCIZ/CREATE A PROCESS/]
        PSOUT
        CRLF
        SKIPGE    CURPRN
        JRST      CRTERR
        HRROI   A,[ASCIZ/FILE: /]
        PSOUT
        MOVSI   A,(GJ%SHT!GJ%OLD!GJ%FNS!GJ%CFM)
        MOVE    B,[.PRIIN,,.PRIOU]
        GTJFN
        ERJMP     CRTER2
        HRRZ    X,A
        MOVX    A,CR%CAP
        CFORK
        ERCAL     ERLEV3
        HRRZ    Y,A
        HRL     A,X
        MOVS    A,A
        GET
        ERJMP     CRTER3
        HRROI   A,[ASCIZ/STARTING PROCESS/]
        PSOUT
        CRLF
        MOVE    R,PROCNO
        ADJ     R,
        CAIN    R,777
        SETZ    R,
        MOVEM   R,PROCNO
        ADD     R,STBASE
        MOVEM   R,CURPRN
        MOVEM   Y,CURFRK
        MOVE    A,Y
        SETZ    B,
```

```
                SFRKV
                ERCAL   ERLEV3
                MOVEI   Z,1
                MOVEM   Z,STCODE
                MOVN    A,Z
                RET

CRTERR: HRROI   A,[ASCIZ/NO SPACE FOR A NEW PROCESS/]
                PSOUT
                CRLF
                SETZ    A,
                RET

CRTER2: MOVEI   A,.PRIIN
                RWJFN
                ERCAL   ERLEV3
                PRIN
                CAIE    A,32
                JRST    CRTER4
                CALL    ABORT
                SETZ    A,
                RET

CRTER3: MOVE    A,Y
                KFORK
                ERCAL   ERLEV3
CRTER4: CRLF
                HRPOI   A,[ASCIZ/ERROR: /]
                PSOUT
                HRLOI   B,.FBSLF
                MOVEI   A,.PRIOU
                ERSTR
                ERCAL   ERLEV3
                JFCL
                JFCL
                SETZ    A,
                RET
```

```
0670 ABORT:  HRROI   A,[ASCIZ/       XXX/]
0680         PSOUT
0690         CRLF
0700         RET
0710
0720 ;-----------------------------------------------------
0730 ;       KILL  CURRENT  PROCESS
0740 ;-----------------------------------------------------
0750
0760 KILLPR: HRROI   A,[ASCIZ/KILL CURRENT PROCESS/]
0770         PSOUT
0780         CRLF
0790         MOVE    A,CURFRK
0800         JUMPE   A,KLLERR
0810         KFORK
0820         ERCAL     ERLEV3
0830         SETZM     CURPRN
0840         SETZM     CURFRK
0850         HRROI   A,[ASCIZ/PROCESS KILLED/]
0860         PSOUT
0870         CRLF
0880         SETZM     STCODE
0890         SETZ    A,
0900         RET
0910
0920 KILLERR: HRROI  A,[ASCIZ/NO PROCESS RUNNING/]
0930         PSOUT
0940         CRLF
0950         SETZB   A,STCODE
0960         RET
0970
0980 ;-----------------------------------------------------
0990 ;       MIGRATE  CURRENT  PROCESS
1000 ;-----------------------------------------------------
1010
1020 MGRATE: HRROI   A,[ASCIZ/MIGRATE CURRENT PROCESS/]
1030         PSOUT
```

```
            CRLF
            SKIPG   CURPRN
            JRST    MGRER2
            SKIPG   NNODES
            JRST    MGRER3
            HRROI   A,[ASCIZ/TO NODE NO. :/]
            PSOUT
            PSIN
            SUBI    A,"0"
            CAMN    A,SELFNO
            JRST    MGRER4
            JUMPLE  A,MGRER5
            CAILE   A,.MAXND
            JRST    MGRER6
            MOVE    R,EPIDNO(A)
            JUMPE   R,MGREP4
            GETZN   IPCB
            MOVE    X,SELFNO
            MOVE    S,EPIDNO(X)
            MOVEM   S,IPCB+1
            MOVEM   R,IPCB+2
            MOVE    W,[2,,TABLE]
            MOVEM   W,IPCB+3
            HRL     A,X
            MOVEM   A,TABLE
            MOVSI   W,.TAKPR
            MOVEM   W,TABLE+1
            MOVEI   A,4
            MOVEI   R,IPCB
            CALL    SNDMSG
            CRLF
            MOVEI   X,2
            MOVEM   X,STCODE
            MOVMI   A,1
            RET
```

```
7410 MGRER2: HERROR   A,[ASCIZ/NO PROCESS RUNNING/]
7420         PSOUT
7430         SETZ     A,
7440         RET
7450
7460 MGRER3: HERROR   A,[ASCIZ/NO NODES ON-LINE/]
7470         PSOUT
7480         CRLF
7490         SETZ     A,
7500         RET
7510
7520 MGRER4: HERROR   A,[ASCIZ/CANNOT MIGRATE TO SELF/]
7530         PSOUT
7540         CRLF
7550         SETZ     A,
7560         RET
7570
7580 MGRER5: HERROR   A,[ASCIZ/ILLEGAL NODE NO./]
7590         PSOUT
7600         CRLF
7610         SETZ     A,
7620         RET
7630
7640 MGRER6: HERROR   A,[ASCIZ/NODE OFF-LINE/]
7650         PSOUT
7660         CRLF
7670         SETZ     A,
7680         RET
7690
7700 ;-----------------------------------------------------------
7710 ;       GET  A  SYSTEM   PROCESS
7720 ;-----------------------------------------------------------
7730
7740 GETSYS: HERROR   A,[ASCIZ/GET SYSTEM PROCESS/]
7750         PSOUT
7760         CRLF
7770         CALL     SYSLST
```

```
                HSTRG   A,[ASCIZ/PROCESS NO. :/]
                PSOUT
                PJIFN
                SUBI    A,"0"
                JUMPL   A,GTSER2
                CAILE   A,9
                JRST    GTSER2
                SETZM   CURPRH
                SETZM   CURFRK
                IMULI   A,10
                MOVE    R,SYSTAB(A)
                JUMPE   R,GTSER2
                MOVS    X,[.PNPTR,,SYSTAB]
                ADD     X,A
                HRR     Y,X
                CAME    Y,SELFNO
                JRST    GTS2
                ADDI    A,3
                MOVE    W,SYSTAB(A)
                MOVE    A,W
                PFSTS
                ERCAL   ERLEV3
                SETOM   FL%SYS
                MOVE    A,W
                PJRZ    B,B
                SETORK
                ERCAL   ERLEV3
                MOVE    A,W
                KITORK
                ERCAL   ERLEV3
                SSTZM   STCODE
                SETZ    A,
                RET
;
GTS2:           MOVE    W,A
                MOVS    S,EPIDNO(Y)
                JUMPG   S,GTS3
```

```
                MOVE    A,Y
                CALL      NXTPID
                JUMPL   A,GTSER3
                MOVE    Y,A
                MOVE    S,B
GTS3:           SETZM     IPCB
                MOVE    X,SELFNO
                MOVE    Z,EPIDNO(X)
                MOVEM   Z,IPCB+1
                MOVEM   S,IPCB+2
                MOVE    Z,[3,,TABLE]
                MOVEM   Z,IPCB+3
                HRL     Y,X
                MOVEM   Y,TABLE
                MOVSI   Z,-RQSYS
                MOVEM   Z,TABLE+1
                MOVEM   R,TABLE+2
                MOVEI   A,4
                MOVSI   B,IPCB
                CALL      SNDMSG
                MOVEI   A,3
                MOVEM   A,STCODE
                SUTO    A,
                PJF
GTSER2:         CRLF
                PSNOI   A,[ASCIZ/NO.-EXISTANT PROCESS/]
                PSOUT
                CRLF
                SETZ    A,
                RET
GTSER3:         CRLF
                PSNOI   A,[ASCIZ/PROCESS NOT AVAILABLE NOW/]
                PSOUT
                CRLF
                SETZ    A,
```

```
08520          RET
08530
08540 ;---------------------------------------------------------------
08550 ;        RESTART CURRENT PROCESS
08560 ;---------------------------------------------------------------
08570
08580 RESCPR: HRROI   A,[ASCIZ/RESTART CURRENT PROCESS/]
08590          PSOUT
08600          CRLF
08610          MOVE    A,CURPRN
08620          JUMPE   A,RCPER2
08630          MOVE    A,CURFRK
08640          RFSTS
08650          ERCAL     ERLEV3
08660          HRRZ    B,B
08670          AOJ     B,
08680          MOVE    A,CURFRK
08690          SFORK
08700          ERCAL     ERLEV3
08710          MOVEI   A,1
08720          MOVEM   A,STCODE
08730          SETO    A,
08740          RET
08750
08760 RCPER2: HRROI   A,[ASCIZ/NO PROCESS RUNNING/]
08770          PSOUT
08780          CRLF
08790          SETZ    A,
08800          RET
08810
08820 ;---------------------------------------------------------------
08830 ;   LOGOUT THIS NODE FROM SYSTEM
08840 ;---------------------------------------------------------------
08850
08860 SYSEXT: CALL      PASWRD
08870          JUMPGE  A,.+3
08880          SETZ    A,
```

```
                RET
                SETZ    Z,
                SETZ    S,
                SLEEP
SXT2:           ADD     S,
                CAML    S,USYSFL
                JRST      SXT3
                MOVE    Y,S
                IMULI   Y,10
                ADDI    Y,3
                HRRZ    A,SYSTAB(Y)
                JUMPE   A,SXT2
                MOVE    B,A
                MOVE    A,Z
                SETO    Q,
                CALL      PXTPID
                JUMPL   A,SXT3
                MOVE    Z,A
                HRL     A,SELFNO
                MOVEM   A,TABLE
                MOVE    B,EPIDNO(Z)
                MOVEM   B,IPCB+2
                MOVE    A,SELFNO
                MOVE    B,EPIDNO(A)
                MOVEM   B,IPCB+1
                MOVE    B,[30,,TABLE]
                MOVEM   B,IPCB+3
                SETZM     IPCB
                SUBI    Y,3
                MOVE    X,SYSTAB(Y)
                CALL      PACKPR
                MOVEI   A,.OFFLD
                MOVEM   A,TABLE+1
                MOVEI   A,4
                MOVEI   B,IPCB
                CALL      SNDRSG
                JRST      SXT2
```

```
                SXT3:   MOVE    X,SELFNO
                        MOVE    W,EPIDNO(X)
                        MOVEM   W,IPCB+1
                        SETZM     IPCB
                        ERLM    X,TABLE
                        MOVE    X,[2,,TABLE]
                        MOVEM   X,IPCB+3
                        MOVEI   X,.OFFLN
                        MOVEM   X,TABLE+1
                        SETZ    A,
                        SETZ    B,
                SXT4:   CALL      EXTPID
                        JUMPL   A,SXT5
                        EXPR    A,TABLE
                        MOVEM   B,IPCB+2
                        MOVE    Z,A
                        MOVEI   A,4
                        MOVEI   B,IPCB
                        CALL      SNDMSG
                        MOVE    A,Z
                        JRST      SXT4
                SXT5:   MOVX    A,.MODES
                        MOVEM   A,IPCB
                        MOVE    B,SELFNO
                        MOVE    A,EPIDNO(B)
                        MOVEM   A,IPCB+1
                        MOVEI   A,2
                        MOVEI   B,IPCB
                        MUTIL
                        ERCAL     ERLEV3
                        HLPOI   A,[ASCIZ/       SYSTEM SHUT DOWN/]
                        PSOUT
                        CLOF
                        CLOF
                        HALTF
                        JRST      STARTX
```

```
09630 ;-----------------------------------------------------------
09640 ;      LOGOUT FROM THIS NODE
09650 ;-----------------------------------------------------------
09660
09670 LOGOUT: HRROI   A,[ASCIZ/LOGGING OUT FROM NODE/]
09680         PSOUT
09690         CRLF
09700         SETZM     STCODE
09710         SETZM     CURPRN
09720         SETZM     CURFRK
09730         SETO    A,
09740         RET
09750
09760 ;-----------------------------------------------------------
09770 ;       UNIMPLEMENTED COMMAND
09780 ;-----------------------------------------------------------
09790 NONXCT: HRROI   A,[ASCIZ/UNIMPLEMENTED COMMAND/]
09800         PSOUT
09810         CRLF
09820         SETZ    A,
09830         RET
09840
09850 ;-----------------------------------------------------------
09860 ;       PRINT COMMAND MENU
09870 ;-----------------------------------------------------------
09880
09890 CMMENU: HRROI   A,MENU
09900         SETZ    B,
09910         PSOUT
09920         SETZ    A,
09930         RET
09940
09950 ;-----------------------------------------------------------
09960 ;  PRINT LIST OF SYSTEM PROCESSES
09970 ;-----------------------------------------------------------
09980
09990 SYSLST: HRROI   A,[ASCIZ/SYSTEM PROCESSES/]
```

```
                PSOUT
                CRLF
                SETZ    X,
                SETZM     TMPFIL+1
                MOVEI   Y,SYSTAB+1
        SYSL2:  MOVEI   A,"V"
                ADD     A,X
                PSOUT
                MOVEI   A,11
                PSOUT
                MOVE    A,(Y)
                MOVEM   A,TMPFIL
                HRROI   A,TMPFIL
                PSOUT
                CRLF
                AOJ     X,
                ADDI    Y,10
                CAMGE   X,NSYSFL
                JRST      SYSL2
                CRLF
                CRLF
                RET

        ;-----------------------------------------------------------------
        ;       ASK FOR PASSWORD
        ;-----------------------------------------------------------------

        PASWRD: HRROI   A,[ASCIZ/PASSWORD:/]
                PSOUT
                CALL      NOECHO
                MOVE    X,[POINT  7,TABLE]
                SETZM     TABLE
                PBIN
                CAIN    A,12
                JRST      PSWD
                IDPB    A,X
                JRST      PASWRD+5
```

```
1.37^ PSWD:   CRLF
1.38^         CALL    ECHOON
1.39^         MOVE    X,TABLE
1.40^         CAME    X,PASSWD
1.41^         JRST    PASERR
1.42^         CRLF
1.43^         SETZ    X,
1.44^         SETZ    A,
1.45^         RET
1.46^
1.47^ PASERR: HRROI   A,[ASCIZ/ILLEGAL PASSWORD/]
1.48^         PSOUT
1.49^         CRLF
1.50^         SETO    A,
1.51^         HLT
1.52^
1.53^ ;-----------------------------------------------------------
1.54^ ;       PRINT SYSTEM STATUS
1.55^ ;-----------------------------------------------------------
1.56^
1.57^ SYSTAT: CRLF
1.58^         CRLF
1.59^         HRROI   A,[ASCIZ/SYSTEM STATUS/]
1.60^         PSOUT
1.61^         CRLF
1.62^         CRLF
1.63^         HRROI   A,[ASCIZ/NUMBER OF NODES IN SYSTEM :/]
1.64^         PSOUT
1.65^         MOVEI   A,.PRIOU
1.66^         MOVEI   B,.MAXND
1.67^         MOVEI   C,^D8
1.68^         NOUT
1.69^         ERCAL   ERLEV3
1.70^         CRLF
1.71^         SETZ    X,
1.72^ SYST2:  ADJ     X,
1.73^         CAILE   X,.MAXND
```

```
1074          JRST    SYST5
1075          MOVE    Y,EPIDNO(X)
1076          HRROI   A,[ASCIZ/       NODE       /]
1077          PSOUT
1078          MOVEI   A,"0"
1079          ADD     A,X
1080          PBOUT
1081          JUMPE   Y,SYST3
1082          CAME    X,SELFNO
1083          JRST    SYST4
1084          HRROI   A,[ASCIZ/       SELF/]
1085          PSOUT
1086          CRLF
1087          JRST    SYST2
1088 SYST3:   HRROI   A,[ASCIZ/       OFF-LINE/]
1089          PSOUT
1090          CRLF
1091          JRST    SYST2
1092 SYST4:   HRROI   A,[ASCIZ/       ON-LINE/]
1093          PSOUT
1094          CRLF
1095          JRST    SYST2
1096 SYST5:   SETZM    STCODE
1097          SETZ    A,
1098          RET
1099
1100
1101 ;*************************************************************
1102 ;       INTERRUPT  SERVICE  ROUTINE
1103 ;            FOR  PROCESS  HALT
1104 ;*************************************************************
1105
1106
1107 PRHINT: MOVE    A,FL%SYS
1108          SETZM    FL%SYS
1109          JUMPN   A,PRH2
1110          MOVE    A,FL%IIC
```

```
1110^           SETZM     FL%IIC
1112^           JUMPN     A,PRM2
1113^           HRROI     A,[ASCIZ/CURRENT PROCESS TERMINATED/]
1114^           PSOUT
1115^           CRLF
1116^           SETZM     STCODE
1117^           MOVEI     A,.FHSLF
1118^           MOVX      B,1B1
1119^           IIC
1120^           ERCAL     ERLEV1
1121^ PRM2:     DEBRK
1122^
1123^
1124^
1125^ ;*****************************************************************
1126^ ;         INTERRUPT SERVICE ROUTINE
1127^ ;            FOR IPC MESSAGES
1128^ ;*****************************************************************
1129^
1130^
1131^
1132^ MSGINT: JSR         SAVAC3
1133^           CRLF
1134^           HRROI     A,[ASCIZ/MESSAGE RECEIVED AT NODE : /]
1135^           PSOUT
1136^           MOVEI     A,.PRIOU
1137^           MOVEI     C,^D8
1138^           MOVE      B,SELFNO
1139^           NOUT
1140^           NUP
1141^           CRLF
1142^           MOVX      R,IP%TTL
1143^           MOVEM     R,IPCB
1144^           SETZM     IPCB+1
1145^           MOVE      S,SELFNO
1146^           MOVE      R,EPIDNO(S)
1147^           MOVEM     R,IPCB+2
```

```
1148?           MOVE    R,[30,,TABLE]
1149?           MOVEM   R,IPCB+3
115??           MOVEI   A,4
1151?           MOVEI   B,IPCB
1152?           PRECV
1153?           ERCAL     ERLEV1
1154?           HKROI   A,[ASCIZ/MESSAGE CODE : /]
1155?           PSOUT
1156?           MOVEI   A,.PRIOU
1157?           MOVEI   C,^D8
1158?           MOVE    B,TABLE+1
1159?           NOUT
1160?           NOP
1161?           CRLF
1162?           MOVE    A,TABLE+1
1163?           CALL      @MSGTAB(A)
1164?           JUMPGE  A,MSG2
1165?           MOVEI   A,.PRIIN
1166?           CTIEP
1167?           ERCAL     ERLEV1
1168?           MOVLI   A,.FRSLF
1169?           MOVX    B,1B1
1170?           IIC
1171?           ERCAL     ERLEV1
1172?           SETZM     STCODE
1173? MSG2:     RET
1174? SAVAC3:   0
1175?           MOVEM   P,ACTAB3+17
1176?           MOVEI   P,ACTAB3
1177?           BLT
1178?           MOVE    P,[IOWD STKLEN,STACK3]
1179?           CALL      @SAVAC3
1180?           MOVSI   P,ACTAB3
1181?           BLT     P,17
1182?           DEBRK
1183?
1184?
```

```
11850 ;-----------------------------------------------------------
11860 ;        SENDER COMING ON-LINE
11870 ;-----------------------------------------------------------
11880
11890
11900
11910 SLFUP:  HLR     R,TABLE
11920         MOVE    S,IPCB+1
11930         MOVEM   S,EPIDNO(R)
11940         MOVEI   S,1
11950         ADDM    S,NNODES
11960         MOVEI   W,SYSTAB
11970         HRL     W,.CNPTR
11980         MOVEI   X,NSYSFL
11990 SLF2:   LDB     Z,W
12000         CAME    Z,R
12010         JRST      SLF3
12020         MOVE    Y,W
12030         HRL     Y,.PNPTR
12040         LDB     Z,Y
12050         CAME    Z,SELFNO
12060         JRST      SLF3
12070         DPB     R,Y
12080         HRRZ    Z,Z
12090         ADDI    Z,2
12100         ERR     A,(Z)
12110         KFORK
12120         ERCAL     ERLEV2
12130         SETO    Z,
12140         ADDM    Z,NSYSFL
12150 SLF3:   ADDI    W,10
12160         SOJGE   X,SLF2
12170         SETZ    A,
12180         RET
12190
12200
12210
```

```
12220 ;-----------------------------------------------------------
12230 ;       SENDER GOING OFF-LINE
12240 ;-----------------------------------------------------------
12250
12260 OFFLN:  HLR     R,TABLE
12270         SETZM     EPIDNO(R)
12280         SETO    Z,
12290         ADDM    Z,NNODES
12300         SETZ    A,
12310         RET
12320
12330 ;-----------------------------------------------------------
12340 ;       SENDER WANTS A SYSTEM PROCESS
12350 ;-----------------------------------------------------------
12360
12370 RQSYS:  HLRZ    R,TABLE
12380         CAMN    R,SELFNO
12390         JRST      RQS2
12400         MOVE    S,[.PRCNO,,TABLE+2]
12410         HRRI    S,TABLE+2
12420         LDB     Z,S
12430         IMULI   Z,10.
12440         ADDI    Z,3
12450         MOVE    Y,SYSTAB(Z)
12460         HLLZM   Y,SYSTAB(Z)
12470         HRRZ    Y,Y
12480         JUMPE     RQS3
12490         MOVE    W,[.PNPTR,,SYSTAB]
12500         SUBI    Z,3
12510         MOVE    X,SYSTAB(Z)
12520         ADD     W,Z
12530         DPB     R,W
12540         MOVE    W,Y
12550         CALL      PACKPR
12560         MOVSS     TABLE
12570         MOVEI   A,.SYSPR
12580         MOVEM   A,TABLE+1
```

```
12590           SETZM     IPCB
12600           MOVE      R,SELFNO
12610           MOVE      S,EPIDNO(R)
12620           MOVEM     S,IPCB+1
12630           HRR       R,TABLE
12640           MOVE      S,EPIDNO(R)
12650           MOVEM     S,IPCB+2
12660           MOVE      R,[30,,TABLE]
12670           MOVEM     R,IPCB+3
12680           MOVEI     A,4
12690           MOVEI     B,IPCB
12700           CALL      SNDMSG
12710           SETZ      A,
12720           RET
12730
12740 RQS2:     HRROI     A,[ASCIZ/PROCESS NOT AVAILABLE/]
12750           PSOUT
12760           CRLF
12770           SETZM     STCODE
12780           SETO      A,
12790           RET
12800
12810 RQS3:     MOVE      W,[.PNPTR,,SYSTAB]
12820           ADD       W,Z
12830           SOJ       W,
12840           LDB       Y,Z
12850           MOVE      X,EPIDNO(Y)
12860           JUMPN     X,RQS4
12870           MOVE      A,Y
12880           CALL      NXTPID
12890           MOVE      Y,A
12900           MOVE      X,B
12910 RQS4:     SETZM     IPCB
12920           MOVE      W,SELFNO
12930           MOVE      R,EPIDNO(W)
12940           MOVEM     R,IPCB+1
12950           MOVEM     X,IPCB+2
```

```
1296^           MOVE    X,[3,,TABLE]
1297^           MOVEM   X,IPCB+3
1298^           HRRM    Y,TABLE
1299^           MOVEI   A,4
1300^           MOVEI   B,IPCB
1301^           CALL      SNDMSG
1302^           RET
1303^
1304^ ;-----------------------------------------------------------
1305^ ;          SYSTEM PROCESS MIGRATED
1306^ ;-----------------------------------------------------------
1307^
1308^ SYSPR:   MOVE    Y,[.PRCNO,,TABLE+2]
1309^           LDB     S,Y
1310^           IMULI   S,10
1311^           MOVE    R,SELFNO
1312^           MOVE    W,[.PNPTR,,SYSTAB]
1313^           ADD     W,S
1314^           DPP     R,W
1315^           MOVE    V,S
1316^           CALL      LDPROC
1317^           MOVE    S,V
1318^           ADDI    S,3
1319^           HRRM    N,SYSTAB(S)
1320^           SETOM     FL%SYS
1321^           MOVE    A,N
1322^           MOVE    B,X
1323^           SFORK
1324^           ERCAL     ERLEV2
1325^           MOVE    A,N
1326^           WFORK
1327^
1328^           ERCAL     ERLEV2
1329^           SETZM     STCODE
1330^           SETO    A,
1331^           RET
1332^
```

```
;-----------------------------------------------------------
;       BID REQUEST FROM SENDER
;-----------------------------------------------------------

TAKPR:  MOVSS     TABLE
        SETZM       IPCB
        MOVE    R,IPCB+1
        MOVE    S,IPCB+2
        MOVEM   S,IPCB+1
        MOVEM   R,IPCB+2
        MOVE    R,[2,,TABLE]
        MOVEM   R,IPCB+3
        SKIPE     CURPRN
        JRST      TKP2
        SETOM     CURPRN
        MOVEI   R,.ACCPR
        MOVEM   R,TABLE+1
        MOVEI   A,4
        MOVEI   R,IPCB
        CALL      SNDMSG
        MOVEI   A,4
        MOVEM   A,SICODE
        SETZ    A,
        RET
TKP2:   MOVEI   R,.NAKPR
        MOVEM   R,TABLE+1
        MOVEI   A,4
        MOVEI   B,IPCB
        CALL      SNDMSG
        SETZ    A,
        RET

;-----------------------------------------------------------
;       BID ACCEPT FROM SENDER
;-----------------------------------------------------------

ACCPR:  MOVSS     TABLE
```

```
13700           MOVEI   R,.SNDPR
13710           MOVEM   R,TABLE+1
13720           MOVE    R,IPCB+1
13730           MOVE    S,IPCB+2
13740           MOVEM   S,IPCB+1
13750           MOVEM   R,IPCB+2
13760           MOVE    S,[30,,TABLE]
13770           MOVEM   S,IPCB+3
13780           MOVE    W,CURFRK
13790           MOVE    X,CURPRN
13800           CALL      PACKPR
13810           MOVEI   A,4
13820           MOVEI   B,IPCB
13830           CALL      SNDMSG
13840           HRROI   A,[ASCIZ/PROCESS MIGRATED/]
13850           PSOUT
13860           CRLF
13870           SETZM     CURPRN
13880           SETZM     CURFRK
13890           SETZM     STCODE
13900           SETO    A,
13910           RET
13920
13930
13940 ;-----------------------------------------------------------------
13950 ;         BID REFUSED BY SENDER
13960 ;-----------------------------------------------------------------
13970
13980
13990 NAKPP:    HRROI   A,[ASCIZ/SPECIFIED NODE BUSY/]
14000           PSOUT
14010           CRLF
14020           SETZM     STCODE
14030           SETO    A,
14040           RET
14050
14060
```

```
;-----------------------------------------------------------
;        A USER PROCESS MIGRATED
;-----------------------------------------------------------

SADPR:  MOVE    R,TABLE+2
        MOVEM   R,CURPRN
        MOVE    R,[.PNPTR,,SYSTAB]
        HRRI    R,CURPRN
        MOVE    S,SELFNO
        DPB     S,R
        CALL      LDPROC
        MOVEM   W,CURFRK
        MOVE    A,W
        MOVE    B,X
        SFORK
        ERCAL     ERLEV2
        MOVEI   A,1
        MOVEM   A,STCODE
        SETZ    A,
        RET

;-----------------------------------------------------------
;     SENDER OFFLOADING SYSTEM PROCESS
;-----------------------------------------------------------

OFFLD:  MOVE    Y,[.PRCNO,,TABLE+2]
        LDB     S,Y
        IMULI   S,10
        MOVE    R,SELFNO
        MOVE    W,[.PNPTR,,SYSTAB]
        ADD     W,S
        DPB     R,W
        MOVE    V,S
        CALL      LDPROC
        MOVE    S,V
        ADDI    S,3
        HRRM    W,SYSTAB(S)
```

```
14440           SETZ    A,
14450           RET
14460
14470
14480 ;*****************************************************************
14490 ;           COMMON  ROUTINES
14500 ;*****************************************************************
14510
14520
14530
14540 ;---------------------------------------------------------------
14550 ;         PACK A PROCESS FOR MIGRATION
14560 ;---------------------------------------------------------------
14570
14580 PACKPR: MOVEM   X,TABLE+2
14590         MOVE    A,W
14600         MOVEI   B,TABLE+3
14610         RFACS
14620         ERCAL     ERLEV4
14630         MOVE    A,W
14640         RFSTS
14650         ERCAL     ERLEV4
14660         HRRZM   B,TABLE+23
14670         MOVE    R,FILNUM
14680         ADJ     R,
14690         CAIN    R,10
14700         SETZ    R,
14710         MOVEM   R,FILNUM
14720         IMULI   R,2
14730         ADD     R,FILNAM
14740         MOVEM   R,TABLE+24
14750         MOVE    R,FILEXT
14760         MOVEM   R,TABLE+25
14770         MOVSI   A,(GJ%SHT!GJ%NEW)
14780         HRROI   B,TABLE+24
14790         GTJFN
14800         ERCAL     ERLEV4
```

```
14810           HRRZ    A,A
14820           HRL     A,W
14830           MOVE    B,[777760,,20]
14840           SAVE
14850           ERCAL     ERLEV4
14860           MOVE    A,W
14870           KFORK
14880           ERCAL     ERLEV4
14890           RET
14900
14910 ;-----------------------------------------------------------------
14920 ;         UNPACK A MIGRATED PROCESS
14930 ;-----------------------------------------------------------------
14940
14950 LDPROC:   MOVX    A,CR%CAP
14960           CFORK
14970           ERCAL     ERLEV4
14980           HRRZ    W,A
14990           HRRZ    A,A
15000           MOVEI   B,TABLE+3
15010           SFACS
15020           ERCAL     ERLEV4
15030           MOVSI   A,(GJ%SHT!GJ%OLD)
15040           HRROI   B,TABLE+24
15050           GTJFN
15060           ERCAL     ERLEV4
15070           HRRZ    A,A
15080           HRL     A,W
15090           GET
15100           ERCAL     ERLEV4
15110           MOVE    X,TABLE+23
15120           MOVSI   A,(GJ%SHT)
15130           HRROI   B,TABLE+24
15140           GTJFN
15150           ERJMP     ERLEV4
15160           HRRZ    A,A
15170           DELF
```

```
15180           ERJMP     ERLEV4
15190           RET
15200
15210 ;-----------------------------------------------------------
15220 ;         ENABLE / DISABLE A PID
15230 ;-----------------------------------------------------------
15240
15250 DSABLE: MOVX     A,.MUDIS
15260           SKIPA
15270 ENABLE: MOVX     A,.MUENB
15280           MOVEM     A,IPCB
15290           MOVE      A,SELFNO
15300           MOVE      B,EPIDNO(A)
15310           MOVEM     B,IPCB+1
15320           MOVEI     A,2
15330           MOVEI     B,IPCB
15340           MUTIL
15350           ERCAL     ERLEV4
15360           RET
15370
15380 ;-----------------------------------------------------------
15390 ;         GET THE NEXT PID
15400 ;-----------------------------------------------------------
15410
15420 NXTPID: AOJ       A,
15430           CAMN      A,SELFNO
15440           JRST      NXTPID
15450           CAILE     A,.MAXND
15460           JRST      NXP2
15470           MOVE      B,EPIDNO(A)
15480           JUMPE     B,NXTPID
15490           RET
15500 NXP2:     JUMPE     0,NXP3
15510           SETZ      A,
15520           SETZ      0,
15530           JRST      NXTPID
15540 NXP3:     SETO      A,
```

```
15550           RET
15560
15570 ;-----------------------------------------------------------
15580 ;  SEND A MESSAGE AND PROCESS ANY ERRORS
15590 ;-----------------------------------------------------------
15600
15610 SNDMSG: PUSH    P,A
15620         PUSH    P,B
15630         MSEND
15640         ERJMP     SNDERR
15650         POP     P,B
15660         POP     P,A
15670         RET
15680 SNDERR: CALL      ERRNUM
15690         CAIN    A,601022
15700         JRST      SMSG2
15710         CAIN    A,601023
15720         JRST      SMSG2
15730         ERCAL     ERLEV4
15740 SMSG2:  POP     P,B
15750         POP     P,A
15760         JRST      SNDMSG
15770
15780 ;-----------------------------------------------------------
15790 ;       TERMINAL ECHO OFF
15800 ;-----------------------------------------------------------
15810
15820 NOECHO: MOVEI   A,.PRIIN
15830         RFMOD
15840         ERCAL     ERLEV4
15850         PUSH    P,R
15860         MOVX    R,TT%ECO
15870         ANDCM   B,R
15880         SFMOD
15890         ERCAL     ERLEV4
15900         POP     P,R
15910         RET
```

```
15920
15930 ;-----------------------------------------------------------
15940 ;       TERMINAL ECHO ON
15950 ;-----------------------------------------------------------
15960  .
15970 ECHOON: MOVEI   A,.PRIIN
15980         RFMOD
15990         ERCAL     ERLEV4
16000         PUSH    P,R
16010         MOVX    R,TT%ECO
16020         IOR     B,R
16030         SFMOD
16040         ERCAL     ERLEV4
16050         POP     P,R
16060         RET
16070
16080 ;-----------------------------------------------------------
16090 ;  ENABLE THE INTERRUPT STRUCTURE
16100 ;-----------------------------------------------------------
16110
16120 ENBINT: MOVEI   A,.FHSLF
16130         EIR
16140         ERCAL     ERLEV4
16150         RET
16160
16170
16180 ;-----------------------------------------------------------
16190 ;       DISABLE THE INTERRUPT STRUCTURE
16200 ;-----------------------------------------------------------
16210
16220 DISINT: MOVEI   A,.FHSLF
16230         DIR
16240         ERCAL     ERLEV4
16250         RET
16260
16270
16280
```

```
16290 ;-----------------------------------------------------------
16300 ;       GET THE CODE OF THE LAST ERROR
16310 ;-----------------------------------------------------------
16320 ERRNUM: MOVEI   A,.FHSLF
16330         GETER
16340         ERCAL     ERLEV4
16350         HRRI    A,B
16360         RET
16370
16380 ;-----------------------------------------------------------
16390 ;       PRINT SYSTEM ERROR MESSAGES
16400 ;-----------------------------------------------------------
16410
16420 ERLEV0: NOP
16430 ERLEV1: NOP
16440 ERLEV2: NOP
16450 ERLEV3: NOP
16460 ERLEV4: NOP
16470         HRROI   A,[ASCIZ/ERROR IN JSYS AT PC /]
16480         PSOUT
16490         MOVEI   A,.PRIOU
16500         MOVEI   C,^D8
16510         POP     P,B
16520         HRRZ    B,B
16530         SUBI    B,2
16540         NOUT
16550         ERCAL     .+1
16560         CRLF
16570         MOVEI   A,.PRIIN
16580         HRLOI   B,.FHSLF
16590         ERSTR
16600         JFCL
16610         JFCL
16620         CRLF
16630         HALTF
16640         JRST      STARTX
16650 END STARTX
```

```
00010 ;*************************************************
00020 ;       UNIVERSAL FILE CONTAINING THE SYMBOLS
00030 ;         USED EXTENSIVELY IN THE PACKAGE
00040 ;       COMPILES TO GIVE THE FILE SYMBOL.UNV
00050 ;*************************************************
00060
00070
00080         UNIVERSAL    SYMBOL
00090         SEARCH  MONSYM
00100
00110
00120 ;-------------------------------------------------
00130 ;       ASSIGN THE FOLLOWING SYMBOLS TO THE
00140 ;                ACCUMULATORS
00150 ;-------------------------------------------------
00160
00170
00180
00190         A=1
00200         B=2
00210         C=3
00220         D=4
00230         Q=5
00240         V=6
00250         L=7
00260         M=10
00270         R=11
00280         S=12
00290         W=13
00300         X=14
00310         Y=15
00320         Z=16
00330         P=17
00340
00350
00360
00370
```

```
00380 ;-------------------------------------------------
00390 ;       DEFINITION OF THE CODES USED IN
00400 ;         INTERPROCESS COMMUNICATION
00410 ;-------------------------------------------------
00420
00430
00440
00450          .SLFUP==        1
00460          .OFFLN==        2
00470          .RQSYS==        3
00480          .SYSPR==        4
00490          .TAKPR==        5
00500          .ACCPR==        6
00510          .NAKPR==        7
00520          .SNDPR==        10
00530          .OFFLD==        11
00540
00550
00560
00570 ;        DEFINITION OF THE MAXIMUM NUMBER
00580 ;             OF NODES IN THE SYSTEM
00590
00600          .MAXND==        3
00610
00620
00630 ;-------------------------------------------------
00640 ;         DEFINITION OF THE POINTERS TO
00650 ;       VARIOUS FIELDS IN A PROCESS NAME
00660 ;-------------------------------------------------
00670
00680
00690          .PNPTR==340600     ;PRESENT NODE PPOINTER
00700          .CNPTR==110600     ;CREATING NODE POINTER
00710          .PRCNO==1100       ;LOCAL PROCESS NUMBER
00720
00730
00740
```

```
00750 ;-----------------------------------------------
00760 ;       DEFINITION OF OPERANDS
00770 ;-----------------------------------------------
00780
00790 OPDEF   CALL    [PUSHJ P,]
00800 OPDEF   RET     [POPJ P,]
00810
00820
00830
00840 ;-----------------------------------------------
00850 ;       DEFINITION OF COMMONLY USED MACROS
00860 ;-----------------------------------------------
00870
00880 DEFINE CRLF <
00890         HRROI A,[BYTE(7) 15,12]
00900         PSOUT
00910         >
00920
00930 DEFINE DEFERR <
00940         CRLF
00950         HRROI A,[ASCIZ/ERROR:/]
00960         PSOUT
00970         MOVEI   A,.PRIIN
00980         HRLOI   B,.FHSLF
00990         ERSTR
01000         CRLF
01010         JFCL
01020         JFCL
01030         >
01040
01050
01060         END
```

APPENDIX - B

Three printouts, showing the operation of ZNODE, XNODE and WNODE are attached.

First ZNODE and XNODE are run as nodes 1 and 2 respectively. XNODE then goes down and WNODE comes up as node 3.

At various intervals the time is output to clarify the sequence of operations.

To get these printouts separately, the nodes were forced to take their input directly from files and give their outputs to other files. Only terminal interrupts were given manually.

```
DECSYSTEM-2050 PSEUDO DISTRIBUTED OPERATING SYSTEM
24-APR-87 12:13:34

*B

NODE NO.:1

FILES LOADED

PROCESS RESPONDING
TIMER.EXE

        PSEUDOS IN OPERATION

PRESS CTRL\Z FOR ATTENTION

        <COMMAND LEVEL>

<12:13:39>
COMMAND: 0
```

COMMAND MENU

COMMAND NO.	COMMAND DESCRIPTION
0	PRINT COMMAND MENU
1	CREATE A PROCESS
2	KILL CURRENT PROCESS
3	MIGRATE CURRENT PROCESS
4	RUN SYSTEM PROCESS
5	CONTINUE CURRENT PROCESS
6	LOG OFF FROM THIS NODE
7	SYSTEM STATUS

INTERACTION OF ZNODE - NODE 1

(PRIVILEGED COMMANDS -- PASSWORD NEEDED)

8 LOG OFF THIS NODE FROM SYSTEM

<END OF COMMAND MENU>

<12:13:39>
COMMAND: 7

SYSTEM STATUS

NUMBER OF NODES IN SYSTEM :3

NODE	1	SELF
NODE	2	OFF-LINE
NODE	3	OFF-LINE

<12:13:39>
COMMAND: 4
GET SYSTEM PROCESS
SYSTEM PROCESSES
0 TIMER

PROCESS NO. :0
PROCESS TIMER
REQUEST NO.: 1
MODE WORD : 0
[24-APR-87 12:13:39]
EXIT

<12:13:39>

```
INTERACTION OF ZNODE - NODE 1
        COMMAND: 6
        LOGGING OUT FROM NODE


                <COMMAND LEVEL>

        <12:14:27>
        COMMAND: 7


        SYSTEM STATUS

        NUMBER OF NODES IN SYSTEM :3
                NODE        1       SELF
                NODE        2       ON-LINE
                NODE        3       OFF-LINE


        <12:14:27>
        COMMAND: 7
        GET SYSTEM PROCESS
        SYSTEM PROCESSES
        0       TIMER


PROCESS NO. :0
PROCESS TIMER
REQUEST NO.: 3
MODE WORD : 0
[24-APR-87 12:14:38]
  EXIT


<12:14:38>
COMMAND: 1
CREATE A PROCESS
```

INTERACTION OF ZNODE - NODE 1

FILE: TEST.EXE

STARTING PROCESS

PROCESS STATUS : 500
PROCESS STATUS : 477
PROCESS STATUS : 476
PROCESS STATUS : 475
PROCESS STATUS : 474
PROCESS STATUS : 473
PROCESS STATUS : 472
PROCESS STATUS : 471
PROCESS STATUS : 470
PROCESS STATUS : 467
PROCESS STATUS : 466
PROCESS STATUS : 465
PROCESS STATUS : 464
PROCESS STATUS : 463
PROCESS STATUS : 462
PROCESS STATUS : 461
PROCESS STATUS : 460
PROCESS STATUS : 457
PROCESS STATUS : 456
PROCESS STATUS : 455
PROCESS STATUS : 454
PROCESS STATUS : 453
PROCESS STATUS : 452
PROCESS STATUS : 451
PROCESS STATUS : 450
PROCESS STATUS : 447
PROCESS STATUS : 446
PROCESS STATUS : 445
PROCESS STATUS : 444
PROCESS STATUS : 443
PROCESS STATUS : 442
PROCESS STATUS : 441
PROCESS STATUS : 440
PROCESS STATUS : 437

```
  INTERACTION OF ZNODE - NODE 1
PROCESS STATUS : 436
PROCESS STATUS : 435
PROCESS STATUS : 434
PROCESS STATUS : 433
PROCESS STATUS : 432
PROCESS STATUS : 431
PROCESS STATUS : 430
PROCESS STATUS : 427
PROCESS STATUS : 426
CURRENT PROCESS TERMINATED
        <COMMAND LEVEL>

<12:14:43>
COMMAND: 3
MIGRATE CURRENT PROCESS
TO NODE NO. :2

PROCESS MIGRATED
        <COMMAND LEVEL>

<12:15:05>
COMMAND: 6
LOGGING OUT FROM NODE


        <COMMAND LEVEL>

<12:17:05>
COMMAND: 7


SYSTEM STATUS

NUMBER OF NODES IN SYSTEM :3
        NODE      1     SELF
        NODE      2     OFF-LINE
        NODE      3     ON-LINE
```

```
        <COMMAND LEVEL>

<12:17:05>
COMMAND: 6
LOGGING OUT FROM NODE


        <COMMAND LEVEL>

<12:17:19>
COMMAND: 7


SYSTEM STATUS

NUMBER OF NODES IN SYSTEM :3
        NODE        1       SELF
        NODE        2       OFF-LINE
        NODE        3       ON-LINE


<12:17:20>
COMMAND: 8
PASSWORD:

        SYSTEM SHUT DOWN
```

```
DECSYSTEM-2050 PSEUDO DISTRIBUTED OPERATING SYSTEM
24-APR-87 12:14:03

*B

NODE NO.:2

FILES LOADED

        PSEUDOS IN OPERATION

PRESS CTRL\X FOR ATTENTION

      <COMMAND LEVEL>

<12:14:10>
COMMAND: 0
```

COMMAND MENU

COMMAND NO.	COMMAND DESCRIPTION
0	PRINT COMMAND MENU
1	CREATE A PROCESS
2	KILL CURRENT PROCESS
3	MIGRATE CURRENT PROCESS
4	RUN SYSTEM PROCESS
5	CONTINUE CURRENT PROCESS
6	LOG OFF FROM THIS NODE
7	SYSTEM STATUS

(PRIVILEGED COMMANDS -- PASSWORD NEEDED)

8	LOG OFF THIS NODE FROM SYSTEM

<END OF COMMAND MENU>

<12:14:10>
COMMAND: 7

SYSTEM STATUS

NUMBER OF NODES IN SYSTEM :3

NODE	1	ON-LINE
NODE	2	SELF
NODE	3	OFF-LINE

<12:14:10>
COMMAND: 4
GET SYSTEM PROCESS
SYSTEM PROCESSES
0 TIMER

PROCESS NO. :0
PROCESS TIMER
REQUEST NO.: 1
MODE WORD : 0
[24-APR-87 12:14:21]
EXIT

<12:14:22>
COMMAND: 6
LOGGING OUT FROM NODE

INTERACTION OF XNODE - NODE 2

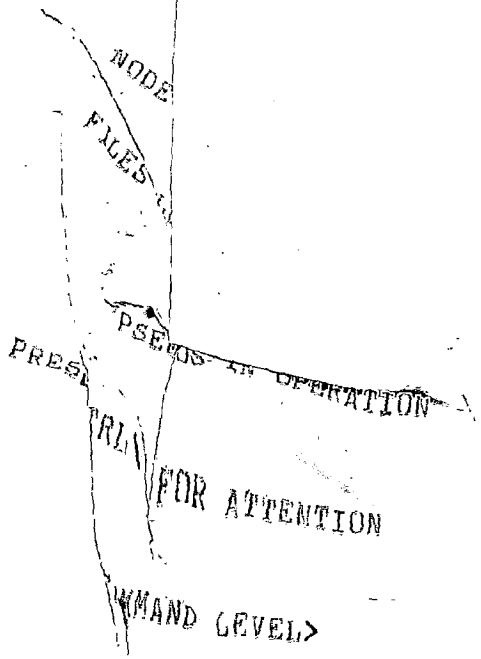
PROCESS STATUS : 425
PROCESS STATUS : 424
PROCESS STATUS : 423
PROCESS STATUS : 422
PROCESS STATUS : 421
PROCESS STATUS : 420
PROCESS STATUS : 417
PROCESS STATUS : 416
PROCESS STATUS : 415
PROCESS STATUS : 414
PROCESS STATUS : 413
PROCESS STATUS : 412
CURRENT PROCESS TERMINATED
<COMMAND LEVEL>

<12:15:35>
COMMAND: 8

PASSWORD:

SYSTEM SHUT DOWN

0 PSEUDO DISTRIBUTED OPERATING SYSTEM
16:38



<12:16:45>
COMMAND: 0

COMMAND MENU

COMMAND NO.	COMMAND DESCRIPTION
0	PRINT COMMAND MENU
1	CREATE A PROCESS
2	KILL CURRENT PROCESS
3	MIGRATE CURRENT PROCESS
4	RUN SYSTEM PROCESS
5	CONTINUE CURRENT PROCESS
6	LOG OFF FROM THIS NODE
7	SYSTEM STATUS

(PRIVILEGED COMMANDS -- PASSWORD NEEDED)

LOG OFF THIS NODE

<END OF COMMAND MENU>

<12:16:45>
COMMAND: 7

SYSTEM STATUS

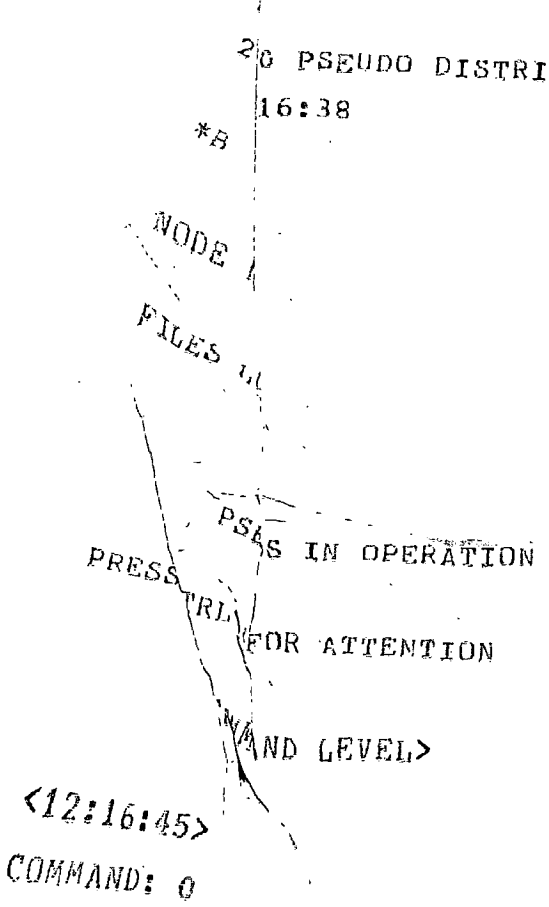
NUMBER OF NODES IN SYSTEM :3

NODE	1	SELF
NODE	2	OFF-LINE
NODE	3	OFF-LINE

<12:16:45>
COMMAND: 4
GET SYSTEM PROCESS
SYSTEM PROCESSES
0 TIMER

PROCESS NO. :0
PROCESS TIMER
REQUEST NO.: 4
MODE WORD : 0
[24-APR-87 12:16:57]
EXIT

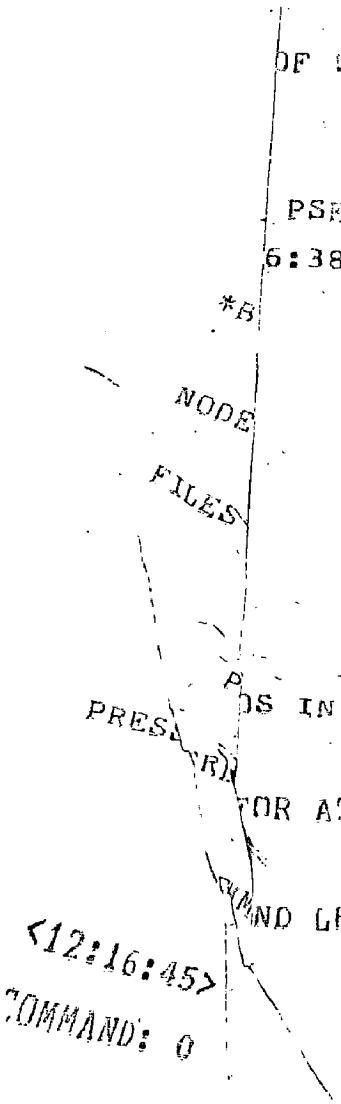
<12:16:57>
COMMAND: 6
LOGGING OUT FROM NODE



COMMAND MENU

COMMAND NO.	COMMAND DESCRIPTION
0	PRINT COMMAND MENU
1	CREATE A PROCESS
2	KILL CURRENT PROCESS
3	MIGRATE CURRENT PROCESS
4	RUN SYSTEM PROCESS
5	CONTINUE CURRENT PROCESS
6	LOG OFF FROM THIS NODE
7	SYSTEM STATUS

(PRIVILEGED COMMANDS -- PASSWORD NEEDED)



OF WNODE - NODE 3 PAGE: 1

PSEUDO DISTRIBUTED OPERATING SYSTEM

6:38

*B

NODE

FILES

P

OS IN OPERATION

PRES R

TOR ATTENTION

ND LEVEL>

<12:16:45>

COMMAND: 0

COMMAND MENU

COMMAND NO.	COMMAND DESCRIPTION
0	PRINT COMMAND MENU
1	CREATE A PROCESS
2	KILL CURRENT PROCESS
3	MIGRATE CURRENT PROCESS
4	RUN SYSTEM PROCESS
5	CONTINUE CURRENT PROCESS
6	LOG OFF FROM THIS NODE
7	SYSTEM STATUS

(PRIVILEGED COMMANDS -- PASSWORD NEEDED)

```
   INTERACTION OF WNODE - NODE 3
          8                LOG OFF THIS NODE FROM


<END OF COMMAND MENU>


<12:16:45>
COMMAND: 7


SYSTEM STATUS

NUMBER OF NODES IN SYSTEM :3
        NODE       1      SELF
        NODE       2      OFF-LINE
        NODE       3      OFF-LINE


<12:16:45>
COMMAND: 4
GET SYSTEM PROCESS
SYSTEM PROCESSES
0        TIMER


PROCESS NO. :0
PROCESS TIMER
REQUEST NO.: 4
MODE WORD : 0
[24-APR-87 12:16:57]
  EXIT


<12:16:57>
COMMAND: 6
LOGGING OUT FROM NODE
```

<COMMAND LEVEL>

<12:17:10>
COMMAND: 8
PASSWORD:

SYSTEM SHUT DOWN