University of
Waterloo Research Institute

A SHARED SPOOL SYSTEM FOR A COMPUTER NETWORK

CANADA-DEPARTMENT OF COMMUNICATIONS

WRI Project 1013

THE UNIVERSITY OF WATERLOO RESEARCH INSTITUTE

A SHARED SPOOL SYSTEM FOR A COMPUTER NETWORK

CANADA-DEPARTMENT OF COMMUNICATIONS

July 13, 1972.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

LIST OF FIGURES

1. INTRODUCTION

This document describes a spooling system for shared use among computers on a Computer Network. The questions that first arise when considering such a task are what services does this shared spool offer and why should a subscriber use it rather than or as well as a spool of his own? What special requirements does a shared spool place on the network, if any? What computing equipment is required to to run a shared spooling system and at what cost?

Before answering these questions we must examine the present Computer Network. Early examples of Computing Machines linked to Communications gear are the SAGE (1) and SABRE (2) systems which were dedicated to a specific service. Later networks were formed for service and file sharing such as the Livermore Octopus(3) system with a central file system and time shared services. In these cases specific machines and equipment have been linked together. The next step was to develop a general communications system for data transportation onto which many types and varieties of computing machines could be linked, hopefully with ease. These various machines would provide the services that prior networks had, plus a more flexible system which is easily expanded. The ARPA Network (4) and a system being developed by NPL in England (5) and a communications network being designed by Fraser at Bell Labs (6) are examples of research in this area.

Considering the more flexible communications networks of ARPA, NPL and Bell Labs what services can a shared spool offer other subscribers on the network? The NPL and Fraser systems are conceived as having both local and trunk networks although the concentration has been on the local networks to date. In a local network where there may be smaller special purpose computers and large powerful computers linked to the same network one machine may wish to queue some of his work for a more specialized machine to run at its lesiure . Peripheral sharing is a probability in a local network for economic reasons.Large file transfers to take place at a later time convenient to the destination is another demand on the network. Some networks spool all messages automatically however this places the network under an undesirable burden. If we accept the principle that spooling should be provided on demand then this and the other duties can be provided by a shared spool and more importantly the spool system itself need not then be duplicated . In a trunk network such as ARPA, which is geared to linking powerful computers and perhaps local networks over large distances, a shared spool is perhaps not so obvious. The problem of shipping large files (e.g. an operating system) over the network at peak time however, can be overcome by local spool to local spool transfers at less loaded network hours. Perhaps then the overall view should be one of local networks having shared spooling systems to handle local spooling chores and forward low priority files over the trunk network during off hours.

1. Spool Services

To be more specific we must now examine the services that these concepts demand of a spool. The spool described in this document offers the following services:

1. The spool accepts and places on disk storage sequential data files. It does not distinguish between what is data and what may be a process awaiting CPU time in another machine.

2. The spool forwards these files to destinations. If any file description is needed at the destination it is expected that the sender -receiver protocol will set this up as the first few records of the file, for example commands in a command language. Some information, in the form of a short file descriptor provided by the origin, is available to allow queue management.

3. The spool provides priority queueing for each destination although a mechanism is provided for a physical destination to look like more than one destination if desired.

4. Certain peripheral services are run by the spool such as card readers, and printers, and a subscriber may operate a queue to his own peripheral through the spool if he so desires.

5. Status reports are available on demand at various levels . The status of the spool, the status of a queue or the status of a particular file may be investigated.

6. Although not obvious to the user, certain error precautions are taken and error recovery is provided for.

7. An accounting scheme is provided so that subscribers

may be appropriately charged.

These are the services which any individual system might demand of its spooling service . The problem then of turning a spooling system into a shared spooling system is to assume that the spool knows nothing about its senders andreceivers. This implies isolating all functions and controlling them by messages to and from external and internal processes. This is precisely the way Hansen (7) suggests we look at any opperating system and indeed the philosophy of this design has been influenced by his writings.

The spooling system divides logically into several parts. The file organization and management routines are necessary for keeping track of files and where they are, where they must be delivered, and the order in which they should be delivered. There must also be error precautions and error recovery routines to guard against and recover from internal and external systems crashes from a file point of veiw. There must be a set of file queue scanning routines for collecting status requests when required. This is all included under the file management processor.

There are two other main process areas, the communications and Input Output processors. Input output to all devices including the network and spool pack looks logically the same to most of the spool however certain devices have special characteristics therefor there are some routines which only these devices use, however it is essentially one I/O package. There is a communications processor for communicating with subscribers and the network (for switching requests). There is a master schedular and a Nucleus to the system. These areas are all relatively independant and communicate in an orderly fashion so that a change in one process does not destroy its relationship with another.

II. SPOOL FILE MANAGEMENT AND ORGANIZATION

Perhaps the logical place to start describing the system is with the most central part, in this case the file organization and management. The IBM HASP (8) system provided the starting point for the following organization although the similarities between HASP and the final system are less obvious.

1. Spool Organization

A spooling system demands several things of its local storage and we must choose an organization compatible with these denands.

a) Disk head movement (if a moving head disk is used) and CPU time should be minimized when allocating storage or forwarding a file.

b) The system should not require large amounts of main memory storage for filing.

c) Most important there should be no serious internal or external fragmentation on the spool

In return for these demands we have several features which we may use to our advantage.

a) We know that all files are sequential.

b) We can block these files as we wish providing we forward them with some specific blocksize (either a network or some other device demand).

Since a direct access device must be used I have chosen the disk unit but the same organization principles apply to most access devices used. In fact the actual choice of device is one of the factors which a system simulation must determine. It is not clear that moving head disks would be suitable and fixed head devices may be required.

We will use a uniform record size and chain the records together sequentially by using the last word of each record as an address of the next record. We will only write a record after establishing what the next record is and where it will go. Now we must format the spool pack in terms of these records and find a scheme for allocating space.

Let a record be the fixed length (in words) of an I/O transfer and an allocation block (some internal number of records) be the minimum space allocated in the device. The problem now is to choose the proper length of record and allocation block. A large allocation block costs fragmentation but is paid for by fewer lookups; large records cost in memory buffer space but require fewer I/O operations. I suggest that the problem of an optimum choice depends on the network being served and the nature of the transfer load. I do not attempt to answer this question but keep record and allocation blocks logically distinct so that they may be varied as the system is tuned.

The disk pack is now divided(conceptually) into allocation blocks and a bit map is kept , one bit for each block. Realistically an allocation block should bear some relationship to the physical device and a track would probably be the smallest block one would choose. The bit map then will be a matrix , one row for each cylinder, one column for each track. In the spool I/O package the present head address is noted and when allocation is necessary the bit map is searched from the present head address for the closest block with respect to head movement. The proper bit is switched (say to 0) to indicate the space is used. Corresponding to the Master bit map is a File bit map for each file. When space is allocated to this file the appropiate bit is set on (say to one). When space is freed an "or" operation is performed between the File bit map and the Master bit map to produce the new Master bit map.

2. Spool File Administration

Now that we have planned the spool pack layout we must look at what informatiom is needed to describe the file and its status at any time.

The basic description of the file will be in its Profile Table. There will be one Profile Table for each file with the following format:

1. File identification
2. Origin
3. Number of file elements referencing it, (discussed in next paragraph)
4. File bit map as explained
5. Address of files first record
6. address of files last record so that the file may be

extended if desired.

7. Accounting fields

8. File descriptor for priority allocation (specified by the origin and obeying some origin destination protocol).

The profile table is stored on the spool disk with the file, and its space is noted in the bit map. It is created when the file isreceived, and remains in core whenever a file is active. There is also a file element created, one for each destination. The reference counter in the Profile Table indicates the number of these existant at any moment and no file maybe deleted unless this counter is zero. The file element remains in core (although copies are occassionaly written out on disk) and is moved from queue to queue depending on the status of the file. It has the following format:

1. File identification
2. Destination
3. Priority
4. Status
5. Control processor
6. Pointer to Profile Table
7. Chain to link files together for shipment as one file to the destination if so desired
8. Pointer to next element in queue
9. Accounting information

A file element is considered to be in one of three states:

a) active

b) awaiting action

c) inactive

This state plus information as to what is taking place (file inbound or outbound) or waiting to take place is recorded in the status field. There is a queue for each state and movement between the queues will be discussed later in this section.

The control processor is the processor to whom messages are sent and from whom they arereceived for queue control. Normally it is the destination who controls its queue. Perhaps there is more than one queue for a destination. Occassionally, as in the case of a device such as the printer, the control processor cannot be the destination; in which case, it is a processor designated as control fot the device. If the device is under spool control then the processor is in the spool. The control processor is the only one with permission to change disposition of a file element once it has been created , or to change its queueing order.

A file identification is assigned in various ways when a file arrives at the spool. The origin supplied identification and a unique spool identification are concatenated and placed in the profile table as the profile table's I.D. The destination is concatenated to this and becomes the file element I.D. which is forwarded to the destination or control processor. There can be no conflict within the spool, however if an origin forwards two files with the same I.D. to the same destination the origin will no longer be able to distinguish between them. Protection over file elements is maintained by requiring the various destinations to quote their own version of the I.D.

There are three basic queues corresponding to the three possible states. An element is placed in the active queue whenever it is being transfered and at no other time. At this time the file is open and its Profile Table is in core.

An element is in the awaiting action queue when a required action has been noted and it has been queued to have that function performed. If an elements disposition is uncertain, either because an action has been completed and it awaits acknowledgement, or because the control processor has not given a disposition, it is placed on the inactive queue.

3. File Queue Movement

An element can be moved to the active queue if it has just been created for an inbound file, or taken from the awaiting action queue provided certain requirements are satisfied. Firstly, the I/O processor must indicate it can find buffers for I/O and can gain access to the required data paths. The I/O processor is described later. Secondly, the communication controller must ascertain whether the destination is willing toreceive the file. The highest priority file element which satisfies these requirements (if any) will become active . When a file transfer has been completed then the element is placed in the inactive queue.

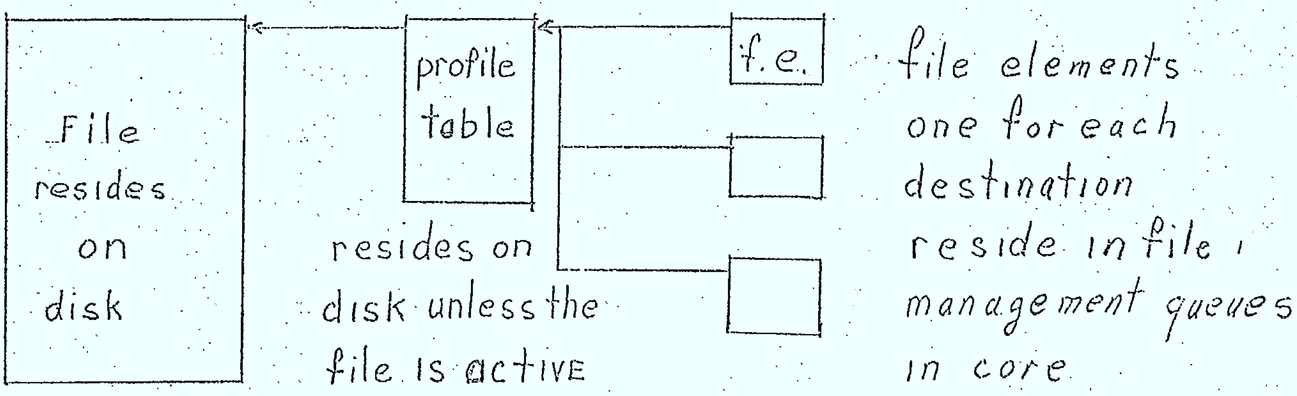
The problem of choosing the next element to be activated is two fold. What destination should be given priority and which of its elements? The problem of choosing the next destination is correctly handled by the Spool processor ; the queuing for that destination should be a function of the processor controling deliveries to the destination in question, hence the Control processor.

To acheive this, access to the awaiting action queue is through an ordered list of destinations elements, containing a pointer to the first elements in the awaiting action queue, the number of people with that destination, and the Control processor for the queue. Elements in the awaiting action queue with a common destination will be chained together. The destination elements are created dynamically as a file bound for the destination appears in the spool, and disappearing when there are no files left for that destination.

A destination element is assigned an initial priority and then moves up and down in the queue depending on its responses. If it often blocks the dispatch of a file its priority will be decremented and it will move down if it always accepts then it will move up in the queue. The destinations at the top will be polled more frequently for service than those at the bottom, and hence a slow user (such as a printer) would be busy quite frequently and would be polled less often as it dropped in the list. The precise formula for increasing and decreasing priorities will be easily changed so that the system can be tuned without difficulty.

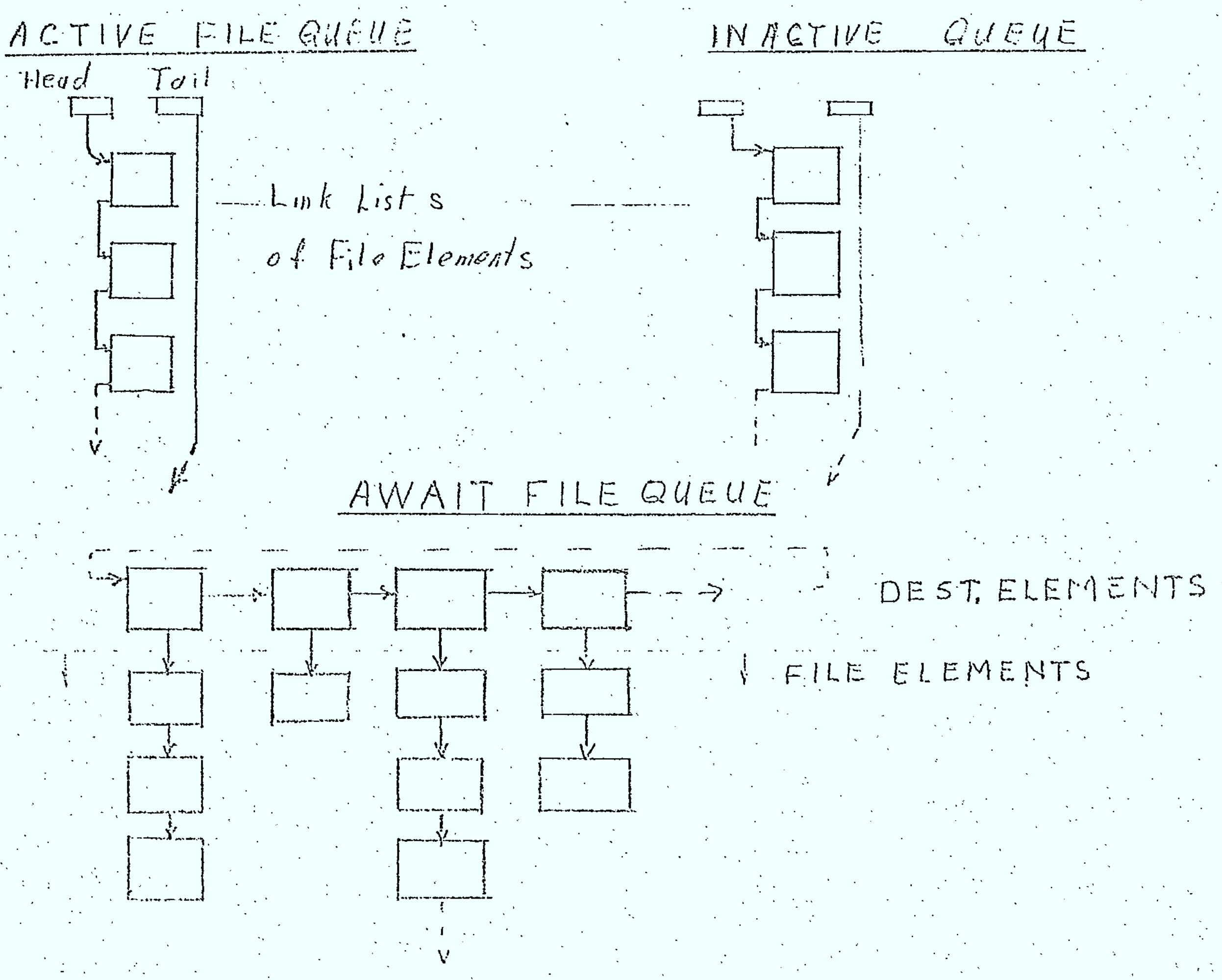
The priority within a destination queue is maintained by the processor designated as control for the destination. Figure 2.1 gives a diagramatical representation of these file queues.

FIGURE 2.1



File Tables

File QUEUES



4. Error Precautions and Recovery

The scheme for error handling and recovery is quite simple. An external error, one that happens to someone else during the course of transmitting a file, is handled in one of two ways. If the file is incoming then the spool assumes that it has never heard of it before. It merely discards its file element, deallocates any spool space allocated and continues. If it is a file being transmitted the file element is simply returned to the awaiting action queue.

Some means of recovery must be provided against internal errors or systems crashes. To allow error recovery, a recovery file is written each time a file element is removed from the active file list.

The recovery file contains a copy of the Master bit map and the file element queues. The version of these is slightly different from that in core and reflects the state that the system should return to in the event of a crash. The philosophy again is that anything active will look as though it had not begun. The routine scans the list of active file elements and places output active file elements in the recovery version of the awaiting queue. Input active files are not queued but an "or" operation is performed between their file bit table and the master bit table, to indicate their space is free; then the awaitng action and the inactive queues as well as the master bit map are written out as the recovery file. If the system goes down while writing this file we cry a great deal and look for the previous copy.

While its is possible to be more sophisticated about keeping track of what stage of activity a file was in and then trying to restart at this point, occasional records will be lost, the recovery therefore will not be reliable and the overhead required increases. The simpler philosophy is reliable and cheap, and at worst will encourage users to pack off smaller files (at least in electrical storms) to the spool.

5. Accounting Method

The accounting formula is not specified but chargeout should be a function of the following parameters:

a) Space used in terms of spool records

b) The length of time the space was used

c) The peripheral services rendered, if any

d) I/O to the network

Other items should also be noted such as the number of destinations for which a file is bound so that the cost may be spread among them. Since time spent waiting may be due to the spool, for instance if a spool supplied peripheral causes a long wait, this time should be recorded. An attempt is made to provide all this information.

The size of a file in terms of spool records is collected in the file profile table by having the spool interpreter bump a counter in the appropriate profile table each time it writes a record. The profile table will also contain the time of the file creation, and themaximum number of destinations that the file has so far been addressed for. The file element contains information appropiate to its destination. It contains the time it was created (which may differ from the profile table), the time it entered the destination queue (it may have been inactive), the number of I/O operations to any peripheral it may have used, and the number of times it wasreceived or shipped out over the network. This I/O information is collected by the device interpreter, first in the appropriate Device Control Table and after I/O completion in the file element. The device control table and device interpreters are explained in the I/O section.

When a file element is being destroyed a record is written on an accounting file against the account of the appropriate destination, or origin in the case of delivery to a spool run peripheral. The accounting record consists of the information previously mentioned and provides the paramenters to a charge-out policy which would be the responsibility of the spool management.

III. THE NETWORK

Before discussing the communications and I/O processors we must be more specific about the network to which the system is interfacing and what services it provides. Starting with the premise that a network user , in this case the Spool, would like to know as little as is possible about the actual network, we need not be concerned with the problems of communications technology. What the Spool must be concerned with is the network interface and how to best use the network. In ARPA and other networks the user is shielded by a network interface from these problems. Fraser has given a relatively complete description of his his network and how to interface to it. I will put forth a users point of view of Frasers network and assume the spool is interfacing to it.

1. Spool's Eye View of the Network

The spool will view the network as offering a number of distinct transmission paths, some of which will be used for data transfer and others for signal transmission to control the data transfer. In particular the user may have many paths open at once and can distinguish logically between data transfer and signal information. One of the transmission lines will terminate at the central office of the network to be used for requesting switching services.

Fraser presents his concept of "full service" to the network subscriber as embodying three central notions. First is the concept of device independant I/O so that while one subscriber will have his own communications procedure it will be relatively independant of those adopted by other subscribers. The second notion is that of a "central office" to whom all requests for switching services may be directed by the user. The third idea is that of an interface between the subscriber and the communications system to provide for the local needs of the subscriber and shield him from the network.

The interface should demand very little in the way of special hardware or software. The network should accept the burden of transmission speed control, protecting thereceiver, via the interface from overloads by holding back incoming information untill thereceive r is ready for it. For a network with limited storage capacity this implies reflecting thereceiver capabilities back to the sender.

The interface unit looks the same to all subscribers. It offers a number of full duplex communications channels each independant of the others except that they go through the same interface. At any given instant only one channel may be accessed however another is easily selected enabling many communications to be carried on at once. Each channel has four data paths which are closely linked and switched in unison. One channel may be chosen to transmit signals which supervise data transmission on the others. Path 0 is used for communications with the local switch and all requests to the local switch are made over this path of any channel. Channel 0 is reserved for communications with the central office. To originate a call a user requests, over data path 0 of the channel he subsequently intends to use, the switch to link him tc another subscriber. The interface passes this request to the central office down data path 0 and the central office in turn passes the request to the receipient over path0 of channel 0. He replies over path 0 of the channel he intends to use and a full duplex connection is formed. The protocol on the various data paths is the responsibility of the subscribers.

The interface appears as two asynchronous devices, the Sender for output and the Receiver for input. Both handle data one 8 bit byte at a time. There are commands for communicating with the Sender and the Receiver; Read/Write one byte of data, Read/Write last byte of message, Select/Read channel and data path, error recovery commands. The I/O controller or a combination af the controller and resident programs must be able to handle these commands. Except for the error recovery commands these are standard I/O initiatives typical of many controllers. To take care of error recovery problems such as restarting transmission of a record after an error or a complete file retransmission there must be some error routines residing within the spool interpreter of the I/O package. It may be desirable to run a powerful programmable controller as a front end to the system to further separate the I/O requirements from the rest of the spool.

Although the interface deals with data in 8 bit bytes,these bits may be assembled into sreams and so in no way restrict the user except that he must be able toreceive these bits. He may of course choose to discard some of them. Protocol between user may involve some restrictions. If someone wishes to send a message to the spool for example he should send the correct number and configuration of bits no matter how he groups them in his machine.

The spool I/O processor multiprocesses its I/O to the network on the buffer level since the interface can bereceiving and transmitting to one channel at any given instant. The I/O processor issues a read or write to the I/O controller specifying a buffer, its length and the channel to which it should be written or read. when this action is completed the next buffer transaction is initiated. The network (Fraser's) is a very high speed device with a maximum effective transmission rate of over 1 mega baud and hence it must be connected to a high speed data highway. Delays due to thereceivers inability to accept data will be filled with I/O to multiplexed peripherals and to and from the spool storage.

IV. INPUT OUTPUT SERVICES

Data files will be just bit strings as far as the spool processor is concerned (with the exception of unit record device interpreters to be described later). This stream approach to the file implies that buffering is the choice of the spool. All buffers will be in terms of the spool records. The buffer need not be filled but the buffer pool will contain only one size buffer. Messages will also be sent from these although they will be multiples of 256 bits long since messages of this length can be most efficiently transported over the network.

A buffer pool is kept consisting of a chain of free buffers and a pointer in the buffer pool control block pointing to the first buffer. When spooling to or from the network the buffers are allocated, filled, transfered, and freed with only the data in the message buffers being examined. In the case of unit record devices, however, the spool must be sheilded from the necessary protocal. In this case the spool deals with the unit record device interpreters who do all the necessary blocking and deblocking.

Any subscriber using unit record peripherals (card readers, printers, punches, etc.) from the spool will obey the following protocal. The first byte of a record must specify its length, the second is a control character for the device interpreter. For example, the printer interpreter examines the data in the file and using the record length deblocks it for the printer and interprets the control character for the carriage control. The reader interpreter assembles card images, with trailing blanks removed, in this fashion and assembles a full buffer before notifying the spool it has a full buffer for it. The reader interpreter would also send the first record as a message to the spool giving the destination and a file description. If the command interpreter cannot interpret the message an error message is sent back, via the reader interpreter and the file is not accepted.

Each device is controlled by a Device Control Table. Each channel of the network will also have one of these. The number of channels up to the network maximum of 64 is a system parameter. The device control tables are linked together and contain information concerning; the device type, the device name, the device status, a field for control information peculiar to the device, a pointer to the control table, and a pointer to the PCE in the present control.

A Device Control Table is permanently associated with the device it controls and is associated with the file it controls as long as the file element is active. The only devices which are not locked on to an active file are the direct access devices. Before each record can be written on the spool pack the device must be obtained again, however, since this is sequential a device will have just been released by the filereceiving attention just before the present one. There is a device control table for the operators console and a console interpreter which makes the operator appear to the system like any other subscriber. The operator receives notice from the system of all file arrivals and departures as well as their I.D.'s thus giving him access to all files.

The spool I/O interpreter is in control of all spool direct access devices and writes buffers of data on a spool when passed the buffer address and a pointer to its profile table.

The Input Output processing is done on two levels. The main processor locates buffers, checks device control tables, and passes buffered information or accepts it to and from other processors. It passes this buffered information to its own device interpretors or to the command processors. The device interpreters deal with their own device types and their own individual protocals. diagramatical representation of the I/O processor is shown in figure 4.1.

As an example suppose we examine the details of an input service. Uponreceiving notice from the I/O controller to open service on a particular type of device the initialization routine is called. The initialization routine attempts to aquire a device of the correct type via a device control table. If no device is available it asks the command processor to assemble a wait message which it can then return. If a device is available get a buffer and ask the correct type of device interpreter to fill it and indicate whether it is a message or data. The device interpreter calls the I/O controller to write the buffer and then assembles the data if necessary.

The buffer is then marked as data for the spool file or as a message for the communications package. The first buffer should be message, either assembled by a reader interpreter orreceived from a subscriber via the network. If it is not, a file element will not exist, a profile table will not exist , and the spool I/O interpreter will not be able to write the record and an error message will be returned.

The Output process is essentialy the reverse procedure. Now, however, the file dispatcher initiates the search for a device control table, then asks the communications package to assemble a message for the destination (control processor). Then the reverse procedure for forwarding data is initiated.

V. COMMUNICATIONS PROCESSOR

The communicaticns processor consists of a set of routines for encoding and decoding messages to and from the network and placing requests for appropriate actions on the appropriate queues. These command interpreter routines will reside in an expandable library of such routines so that new functions may be conveniently added. The spool will dictate to its subscribers the protocol that messages to the spool should be communicated on data path 1 of the established channel and data transfers over paths 2 and 3.

The spool commands will fall into two categories:

A. Commands to and from the network over data path 0.

These commands will of course be dictated by the network but will contain at least commands to connect via a specific channel to a specified subscriber and toreceive network requests over channel 0 for a channel for communication with a subscriber.

B. Commands to and from the subscriber.

1. File Control Commands

a) Will you accept file x

b) here comes file x

c) Have youreceived file x

Provision to both send andreceive these messages are provided. The designation file x means a file referred to by its subscriber identification or by its place in a subscribers queue.

2. File Status Enquiries and Status Changes

Commands to the spool

a) What priority is file x

b) send the descriptor of file x

c) Change the priority of file x to y

d) Where is file x

Answers to these enquiries are provided by the spool.

Commands from the spool.

a) Specify a priority for file x (the file descriptor is included with this command)

3. Status Reports on Queues and Files

a) How many are in my destination queue

b) How many files are on the queue

c) How many files are awaiting dispatch

d) How many files are inactive

4. File Disposition Commands

Commands to the spool

a) Delete file x

b) Forward file x

c) Chain file x to file y (send it as one file when its turn arrives)

d) send file x to destination y (only the origin or the destination may legally issue this ccmmand)

Commands from the spool

a) Specify a disposition for file x (the present disposition is forwarded along with this message)

VI. SCHEDULING AND INTER-PROCESS COMMUNICATION

The three main areas of the spool processor are, as just described, the input output processor, the communications processor and the file management processor. There is also a master schedular and a nucleus to provide communication between the processors. This communication takes place in a manner similar to that described by Hansen(7) and the technique used on the Titan computer at Cambridge(9). Messages are passed between processors via the nucleus by requesting the nucleus to place a message for one of the three main processors in the appropriate message queue. Then as that processor receives CPU attention the top message from his message queue is acted upon. That processor in turn may request that the nucleus enqueue a message for another processor based on the results of his action.

One of the distinctions between a spool system and a more general system is that no object process ever requires CPU attention for execution of its own code. An object process simply makes system requests. All spool activities therefore are system activities and deserve equal attention. Messages then have no priority and are serviced on a first come first served basis. The same philosophy is extended to the scheduling of the main processors and the master schedular parcels out CPU time in a round robin fashion.

Interrupts are all natural break. Systems routines volutarily give up control after they have performed a task or, if the activity is lengthly they may place a message on the message queue and continue when their turn occurrs again. Since all activities are system ones there is never the problem of an unkind user taking control of the CPU and not returning.

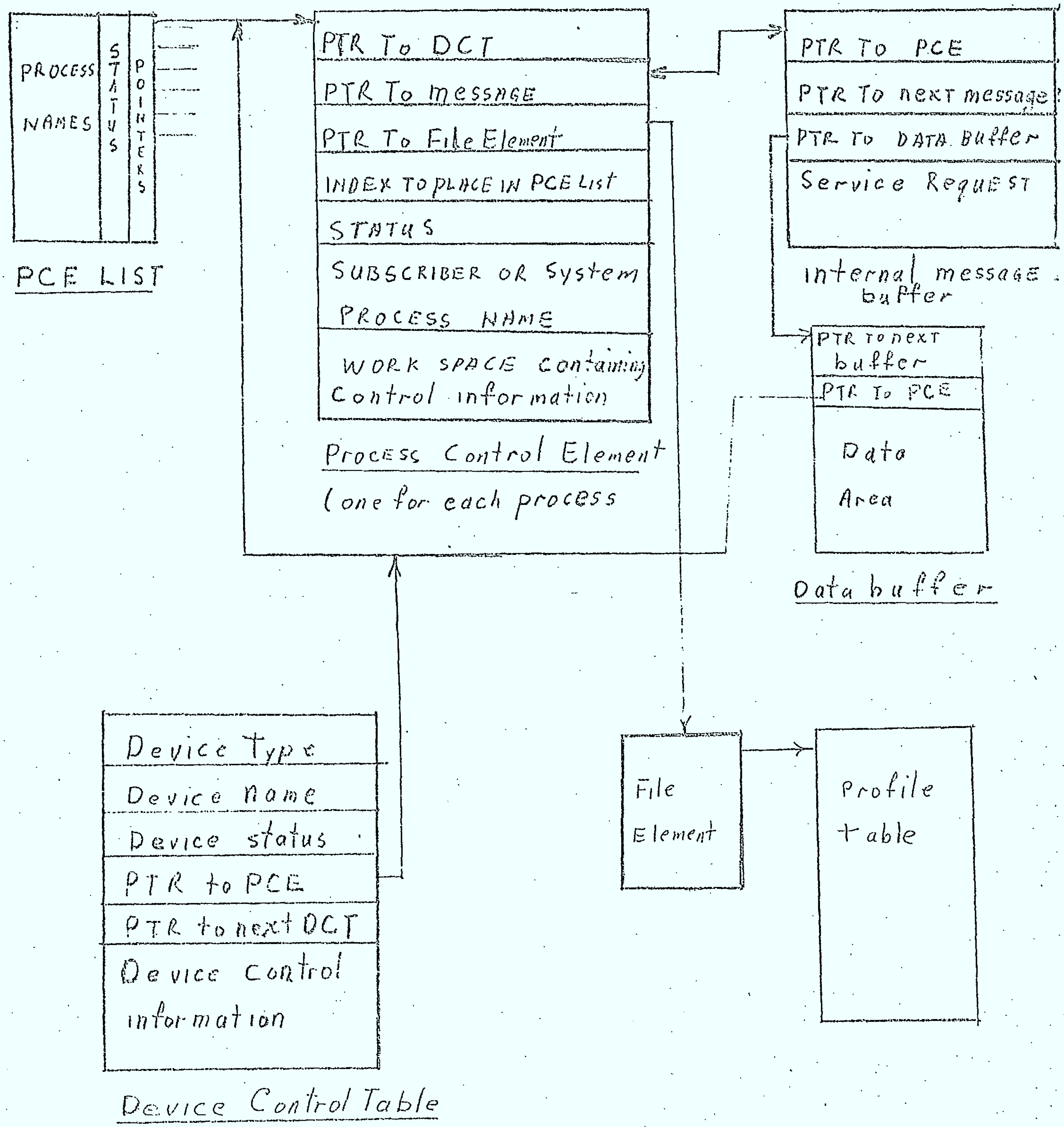
More must now be said about what defines a process and what defines an inter process message. The system is essentialy a special purpose time sharing system with one time shared subscriber process for each subscriber who is logged in to the system. A subscriber is considered logged in to the system if there is an open communication with the subscriber whether the subscriber initiated the conversation or the spool initiated it. There are also certain independant system processes, such as a routine to seek new files for dispatch purposes which remain permanently alive in the system. These independant system processes correctly belong in one of the three main spool processor areas and request services of the other processors by the normal message requests.

Each subscriber and system process is described by a Process Control Element which contains pointers to its various assosciated tables, information concerning the process status, the subscriber or independant system process it is responsible to and a work space to store various control information required for the next time it receives attention from some processor (see figure 6.1).

There is a list of process control elements kept with a pointer to each and its current status. Each process is in one of four states Running, in which case it isreceiving service, Halted if it is waiting for attention and would be running if it wasreceiving the attention, Wait state if it is awaiting the completion of some activity (for example I/O), and Stopped if it may not proceed even though it has work to do.

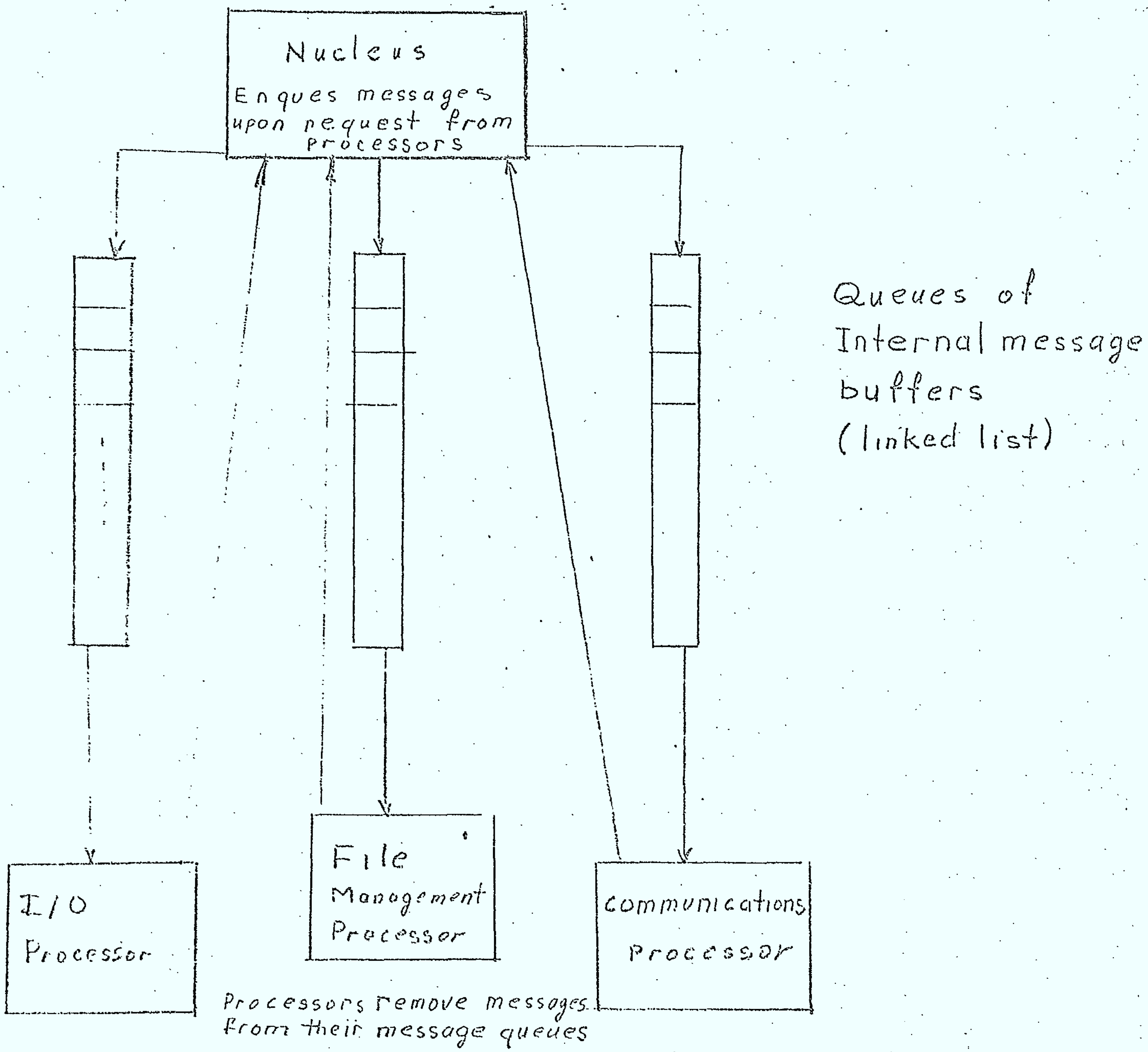
Messages are passed between processors in special message buffers which are queued in the appropriate queue by the nucleus upon request. They contain a service request, pointer to the process control element it is associated with, a pointer to the next message buffer in its queue and a pointer to any data buffer associated with it. Data in this case could be a message from an external process waiting to be translated by the communications processor. Actually while the message buffer is logically distinct from the process control element they are in one to one correspondance since the system does not allow more than one service request from a process at any one time, and may be implemented as one table to cut down on pointers and searching. A diagramatical representation of the system, its tables and inter-process communications are presented in figures 6.1,6.2,and6.3.

FIGURE 6.1



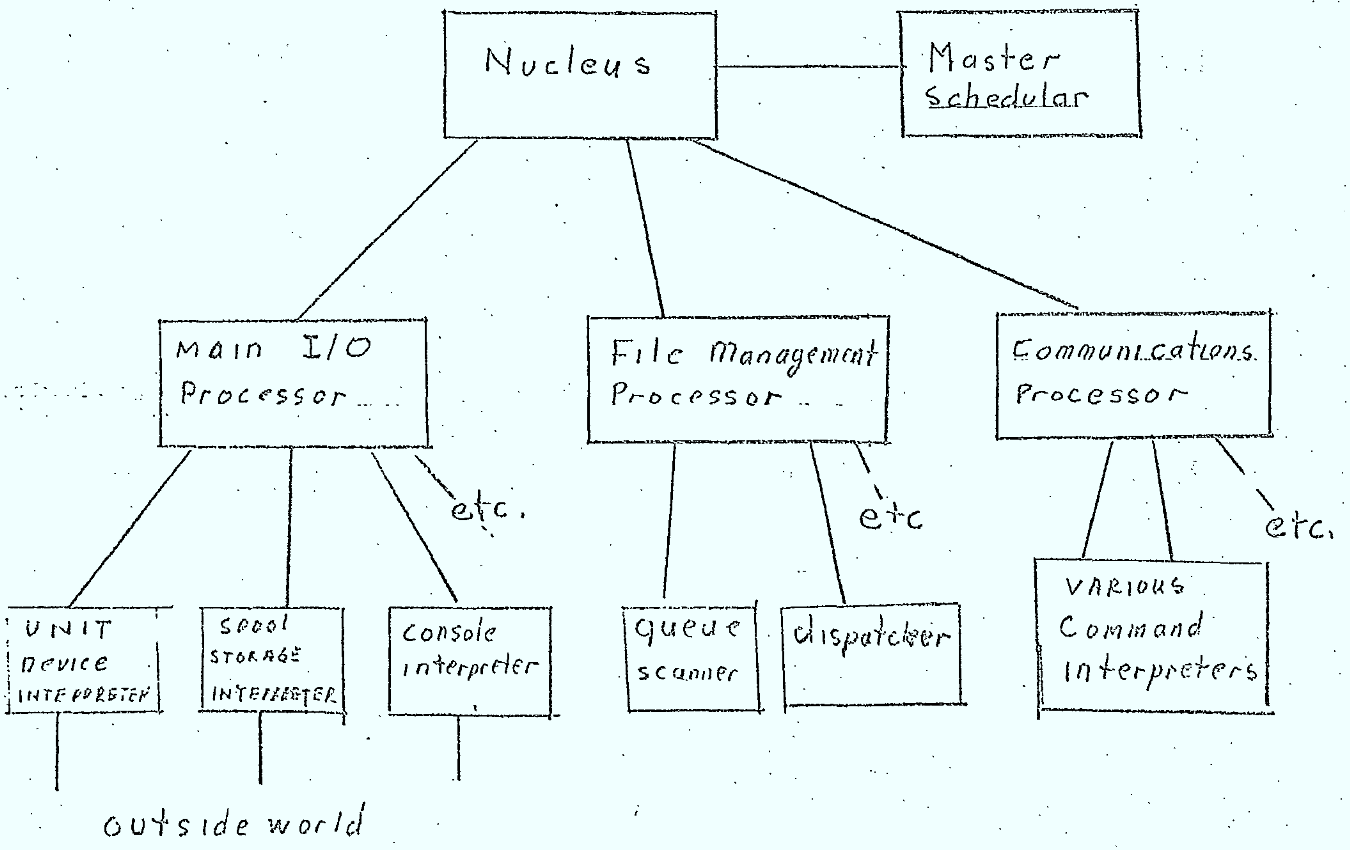
SYSTEM TABLES

FIGURE 6.2



INTER-PROCESS COMMUNICATIONS

FIGURE 6.3



SYSTEM PROCESSOR HIERARCHY

VII. CONCLUSIONS

In setting down a system design for a shared spooling service for a local network I have answered only some of the questions originaly posed. Some of the others could be answered by a system simulation and still others (such as whether you could sell the services) could be answered only by offering such a service.

Concerning the system demands on the network we have discovered at least that the system is not particularly unusual and except for increased traffic, demands no extra network services. The increased traffic problem is one that can be answered by network simulation.

It is clear that we may indeed build and operate a shared spooling system and provide the services outlined in the introduction. It is also clear that these services are desirable on a network (though to what extent we cannot tell). Merely designing the system however cannot answer the most important questions concerning equipment, cost and hence feasability.

A simulation of the system could perhaps answer the following questions. What hardware and hardware configuration is necessary to support the system? More specifically can moving arm disks provide the spool storage facilities and under what loads or must we use more expensive fixed head devices? How does the very high network transmission rate effect this problem? Should the I/O functions rest mostly in a large and more powerful CPU or should they be separated off into an independant programmable front end control unit? How many subscribers could the system support logged in at any one time given certain transmission rates?

Only after answering the above questions via a system simulation can any attempt be made to estimate the cost and feasability of this type of spool service.

VIII. REFERENCES AND BIBLIOGRAPHY

1. Everett, R.R., Zraket, C.A. and Benington, H.D., "Sage a Data Processing System for Air Defense", Proc.EJCC, 1957.

2. Evans, J. "Experience Gained from the American Airlines SABRE System Control Program", Proc. ACM National Meeting August 1967.

3. Fletcher, J.G. "PDP 6 System at LRL, Livermore", Report No 70185, November 1966 Univ. of Calif., Lawrence Radiation Lab.

4. Roberts, L."Computer Network Development to Achieve Resource Sharing", Proc. AFIPS SJCC, vol 36, 1970, P543-550.

   Heart, F. kahn, R., Ornstein S., Crowther, W., Walden D., "The Interface message Processor for the ARPA Computer Network" as above, p551-568

   Kleinrock L., "Analytic and Simulation Methods in Computer Network design", as above, p569-580

   Frank H., Frisch, Chou W., "Topological considerations in the Design of the ARPA Computer Network", as above, p581-588

   Carr S., Crocker S., Cerf V. "Host-Host Communications Protocol in the ARPA Network" as above, p589-596.

5. Davies, D.W., "Communications Networks to serve Rapid Response Computers", IFIP Conf. Proc., IFIP Congress Edinburgh, Aug. 1968. (also various other papers in the same proceedings)

6. Fraser A.G. "A Communications Network for Computers" Bell Labs. Technical Memo. , March 1970

7. Hansen B., "The Nucleus of a Multiprogramming System", CACM, Vol 13, number 4, april 1970 p238-241, 250

8. "The IBM Hasp System", IBM Hasp Manual, August 1969.

9. Hartley, D.F., Landy, B. and Needham, R.M. "the Structure of a Multi-Programming Supervisor", Computer Journal Vcl 11, number 3, Nov 1968.

10. Farmer, W.D. and Newhall, E.E. , "An Experimental Distributed Switching System to Handle Bursty Computer Traffic", Bell Tel. Labs. , Holmdel, New Jersey

11. Fraser A. G., "Coordination of Communicating Processes", Bell Tel. Labs. technical memo, Jan 1970.

APPENDIX A

```
;*****************************************************************************
;*                                                                          *
;*                                                                          *
;*     N E T S P O O L   C O M M   P R O C E S S O R                         *
;*                                                                          *
;*                                                                          *
;* DATE:MAY 24TH,1972                                                       *
;* AUTHOR: MARC DUFRESNE                                                    *
;* VERSION: 1   LEVEL: 0          LAST UPDATE: MAY 24TH,1972                *
;*                                                                          *
;*                                                                          *
;*****************************************************************************
;
;
        R0=%0                      ;REGISTER 0, WORK REG.
        R1=%1                      ;REGISTER 1 ,WORK REG.
        R2=%2                      ;REGISTER 2 ,WORK REG.
        R3=%3                      ;REGISTER 3 ,NOT USED
        R4=%4                      ;REGISTER 4 ,NOT USED
        R5=%5                      ;REGISTER 5 ,NOT USED
        SP=%6                      ;REGISTER 6, STACK POINTER
        PC=%7                      ;REGISTER 7, PROGRAM COUNTER
;
;
        .CSECT
        .TITLE  CMPROC
;
        MOV     CMQHED,R0          ;LOAD HEAD OF MESSAGE QUEUE
        MOVB    8(R0),R1           ;LOAD REQUEST TYPE
        MOVB    9(R0),R2           ;LOAD ACTUAL REQUEST
        JMP     JTABL3(R1)         ;JUMP TO REQUEST TYPE HANDLER
;
DECODE:JSR      PC,JTABL2(R2)      ;CALL TO THE SUITABLE COMMAND DECODER
        RTS     PC
;
;******* ACCEPT FILE  DECODER  ROUTINE
;
ACKFIL:MOV      @6(R0),R1          ;LOAD PTR TO PCE AND ACCESS IT
        MOV     4(R1),R1           ;GET ADDR OF FILE ELEMENT
        BNE     ERR1               ;IF NOT EQUAL TO NULL : ERROR
        MOV     #2.,(+R0)          ;LOAD ID OF FILE MANAGER IN REQUEST
        CLR     3(R0)              ;SET REQUEST TYPE & ACTUAL REQUEST
                                   ;(REQTYP=0,ACTREQ=0)
        RTS     PC                 ;AND RETURN
;
ERR1:   MOV     #2.,(R0)           ;SET ADDR OF I/O PROCESSOR
        MOV     4(R0),R1           ;GET ADDR OF MSG BUFFER
        MOV     #25.,4(R1)         ;PUT NAK IN MSG BUFFER
        RTS     PC                 ;AND RETURN
;
;******* HERE COMES FILE DECODER ROUTINE
;
COMFIL:MOV      #4.,(R0)           ;MOVE I/O PROC ID TO MSG BUFFER
        CLRB    8(R0)              ;SET REQUEST TYPE TO 0
        MOVB    #2.,9(R0)          ;SET ACTUAL REQUEST TO 2
        RTS     PC                 ;AND RETURN
```

```
;******* HAVE YOU RECEIVED FILE  DECODER ROUTINE
;
RECFIL:MOV     #4.,(R0)           ;MOVE FILE MAN. PROC ID TO MSG BUFFER
       CLRB    8(R0)              ;SET REQUEST TYPE TO 0
       MOVB    #4.,9(R0)          ;SET ACTUAL REQUEST TYPE TO 4
       RTS     PC                 ; & RETURN
;
;******* GIVE PRIORITY OF FILE DECODER ROUTINE
;
PRIFIL:MOV     #4.,(R0)           ;MOVE FILE MAN. PROC ID TO MSG BUFFER
       CLRB    8(R0)              ;SET REQUEST TYPE TO 0
       MOVB    #6.,9(R0)          ;SET ACTUAL REQUEST TO 6
       RTS     PC                 ; & RETURN
;
;******* SEND DESCRIPTOR DECODER ROUTINE
;
SNDDES:MOV     #4.,(R0)           ;MOVE FILE MAN. PROC ID TO MSG BUFFER
       CLRB    8(R0)              ;SET REQUEST TYPE TO 0
       MOVB    #10.,9(R0)         ;SET ACTUAL REQUEST TO 8
       RTS     PC                 ;&RETURN
;
;******* CHANGE PRIORITY OF FILE DECODER ROUTINE
CHGPRI:MOV     #4.,(R0)           ;MOVE FILE MAN. PROC ID TO MSG BUFFER
       CLRB    8(R0)              ;SET REQUEST TYPE TO  0
       MOVB    #12.,9(R0)         ;SET ACTUAL TYPE TO 10
       RTS     PC                 ;&RETURN
;
;******* WHERE IS FILE (STATUS) DECODER ROUTINE
;
WHEFIL:MOV     #4.,(R0)           ;MOVE FILE MAN. PROC ID TO MSG BUFFER
       CLRB    8(R0)              ;SET REQUEST TYPE TO 0
       MOVB    #14.,9(R0)         ;SET ACTUAL REQUEST TO 12
       RTS     PC                 ;& RETURN
;
;******* GIVE # OF FILES IN DESTINATION DECODER ROUTINE
;
NUMDST:MOV     #4.,(R0)           ;MOVE FILE MAN. PROC ID TO MSG BUFFER
       CLRB    8(R0)              ;SET REQUEST TYPE TO 0
       MOVB    #16.,9(R0)         ;SET ACTUAL REQUEST TO 14
       RTS     PC                 ;& RETURN
;
;******* GIVE # OF FILES ON ALL QUEUES DECODER ROUTINE
;
NUMQUE:MOV     #4.,(R0)           ;MOVE FILE MAN. PROC ID TO MSG BUFFER
       CLRB    8(R0)              ;SET REQUEST TYPE TO 0
       MOVB    #20.,9(R0)         ;SET ACTUAL REQUEST TO 16
       RTS     PC                 ;& RETURN
;
;******* GIVE # OF FILES IN WAIT QUEUE DECODER ROUTINE
;
NOWDIS:MOV     #4.,(R0)           ;MOVE FILE MAN. PROC ID TO MSG BUFFER
       CLRB    8(R0)              ;SET REQUEST TYPE TO 0
       MOVB    #22.,9(R0)         ;SET ACTUAL REQUEST TO 18
       RTS     PC                 ;& RETURN
;
;******* GIVE # OF FILE IN INACTIVE QUEUE DECODER ROUTINE
;
NOINAC:MOV     #4.,(R0)           ;MOVE FILE MAN. PROC ID TO MSG BUFFER
       CLRB    8(R/)              ;SET REQUEST TYPE TO 0
       MOVB    #24.,9(R0)         ;SET ACTUAL TYPE TO 20
       RTS     PC                 ;& RETURN
```

```
;
;******* DELETE FILE DECODER ROUTINE
;
DELFIL:MOV      #4.,(R0)            ;MOVE FILE MAN. PROC ID TO MSG BUFFER
       CLRB     8(R0)               ;SET REQUEST TYPE TO 0
       MOVB     #26.,9(R0)          ;SET ACTUAL REQUEST TO 22
       RTS      PC                  ;& RETURN
;
;******* FORWARD FILE DECODER ROUTINE
;
FORLFIL:MOV     #4.,(R0)            ;MOVE FILE MAN PROC ID TO MSG  BUFFER
       CLRB     8(R0)               ;SET REQUEST TYPE TO 0
       MOVB     #30.,9(R0)          ;SET ACTUAL REQUEST TO 24
       RTS      PC                  ;& RETURN
;
;******* SEND FILE TO DESTINATION Y DECODER ROUTINE
;
SENFIL:MOV      #4.,(R0)            ;MOVE FILE MAN. PROC ID TO MSG BUFFER
       CLRB     8(R0)               ;SET REQUEST TYPE TO 0
       MOVB     #32.,9(R0)          ;SET ACTUAL REQUEST TO 26
       RTS      PC                  ;&RETURN
;
JTABL3:.WORD    DECODE              ;ADDR OF DECODER ROUTINE HANDLER
;
JTABL2:.WORD    ACKFIL              ;ADDR OF 'ACCEPT FILE?' DECODER
       .WORD    COMFIL              ;ADDR OF 'HERE COME FILE' DECODER
       .WORD    RECFIL              ;ADDR OF 'RECIEVED FILE?' DECODER
       .WORD    PRIFIL              ;ADDR OF 'GIVE PRIORITY' DECODER
       .WORD    SNDDES              ;ADDR OF 'SEND DESCRIPTOR' DECODER
       .WORD    CHGPRI              ;ADDR OF 'CHANGE PRIORITY' DECODER
       .WORD    WHEFIL              ;ADDR OF 'WHERE FILE' DECODER
       .WORD    NUMDST              ;ADDR OF '# IN DEST. QUEUE' DECODER
       .WORD    NUMQUE              ;ADDR OF '# IN QUEUE(TOTAL)' DECODER
       .WORD    NOWDIS              ;ADDR OF '# IN WAIT QUEUE' DECODER
       .WORD    NOINAC              ;ADDR OF '# IN INACTIVE Q' DECODER
       .WORD    DELFIL              ;ADDR OF 'DELETE FILE' DECODER
       .WORD    FORFIL              ;ADDR OF 'FORWARD FILE' DECODER
       .WORD    SENFIL              ;ADDR OF 'SEND FILE ' DECODER
;
       .END
```

```
;*******************************************************************
;*                                                                 *
;*                                                                 *
;*    N E T S P O O L   F I L E   M A N A G E M E N T              *
;*                   P R O C E S S O R                             *
;*                                                                 *
;*                                                                 *
;* DATE:MAY 24TH,1972                                              *
;* AUTHOR: MARC DUFRESNE                                           *
;* VERSION: 1   LEVEL: 0        LAST UPDATE: JUNE 11TH,1972        *
;*                                                                 *
;*                                                                 *
;*******************************************************************
;
;
      R0=%0                      ;REGISTER 0, WORK REG.
      R1=%1                      ;REGISTER 1 ,WORK REG.
      R2=%2                      ;REGISTER 2 ,WORK REG.
      R3=%3                      ;REGISTER 3 ,NOT USED
      R4=%4                      ;REGISTER 4 ,NOT USED
      R5=%5                      ;REGISTER 5 ,NOT USED
      SP=%6                      ;REGISTER 6, STACK POINTER
      PC=%7                      ;REGISTER 7, PROGRAM COUNTER
;
;
      .CSECT
      .TITLE  FMPROC
;
      MOV     FMQHED,R0          ;LOAD HEAD OF MESSAGE QUEUE
      MOVB    9(R0),R1           ;LOAD ACTUAL REQUEST
      JSR     PC,JTABL4(R1)      ;& PROCESS IT.
;
GETSPC:MOV    @6(R0),R1          ;ACCES PCE
      MOV     4(R1),R1           ;LOAD ADDR OF FE
      MOV     DSKHED,R2          ;LOAD DISK HEAD ADDR
      MOV     BITMAP,R3          ;LOAD ADDR OF MASTER BIT MAP
      MOV     1(R2),R4           ;LOAD CYLINDER ADDR
      MOV     2(R2),R2           ; LOAD TRACK ADDR
;
;  BIT MAP SEARCH ROUTINE TO BE CONTINUED LATER
;
```

```
GIVDES:MOV      @4(R0),R1         ;ACCESS DATA BUFFER
       MOV      10(R1),R2         ;ACCESS DESTINATION
       MOV      DESTHED,R3        ;GET A DESTINATION ELEMENT
       MOV      DESTHD,R4         ; LOAD PTR FOR END OF D.E LIST
;
DESRCH:CMP      R2,R3             ;SEE IF WE HAVE SAME DESTINATION
       BEQ      FESRCH            ;YS?: THEN LOOK FOR F.E.
       MOV      (R3),R3           ;NO?: THEN ACCESS NEXT D.E.
       CMP      R3,R4             ;HAVE WE GONE AROUND THE LIST
       BEQ      NOFIND            ;YS?: THE WE HAVE AN ERROR
       BR       DESRCH            ;NO?: THEN COONTINUE THE SEARCH
;
FESRCH:MOV      @2(R3),R3         ;ACCESS F.E.
LOOP1: CMP      4(R2),(R3)        ;TEST FILE ID (1ST PART)
       BNE      NEXTFE            ;WRONG ONE?: THEN GET NEXT F.E.
       CMP      6(R2),2(R3)       ;TEST FILE ID (2ND PART)
       BNE      NEXTFE            ;WRONG ONE?: THEN GET NEXT F.E.
       MOV      12(R3),R2         ;GET ADDR OF PROFILE TABLE
       BMI      ACTVFE            ;IF ADDR NEG:PROFILE TABLE NOT IN CORE
       MOV      PTNDEX(R2),R2     ;LOAD BEGINING OF DESCRIPTOR TEBALE
       ADD      #4.,R1            ;SET ADDR OF DATA AREA
       MOV      DESLEN,R3         ;MOV DESCRIPTOR TABLE LENGTH
LOOP2  MOV      (+R2),(+R1)       ; MOVE DESCRIPTOR TABLE TO DATA AREA
       DEC      R3                ;DECREMENT COUNTER
       BNE      LOOP2             ;AND LOOP UNTIL EQUAL TO 0
       MOV      #2.,(R0)          ;SET ADDR OF I/O PROC
       CLR      8(R0)             ; SET REQUEST TYPE & ACTUAL REQUEST TO 0
       RTS      PC                ;& RETURN
;
NEXTFE:MOV      16(R3),R3         ;LOAD  ADDR OF NEXT F.E
       BEQ      NOFIND            ;PTR = NULL?: THEN EN D OF LIST
       BR       LOOP2             ;OTHERWISE WE TEST AGAIN FOR FILE ID.
;
NOFIND:MOV      #2.,(R0)          ;SET ADDR OF I/O PROC
       CLR      8(R0)             ;SET RREQUEST TYPE & ACTUAL REQUEST TO0
       MOV      #25.,4(R1)        ;MOVE NACK TO DATA BUUFFER
       RTS      PC                ;AND RETURN
;
ACTVFE:MOV      R2,4(R1)          ;LOAD ADDR OF PROFILE TABLE IN DATA BUF.
       MOV      #2.,(R0)          ;SET ADDR OF I/O PROC
       CLRB     8(R0)             ;SET REQUEST TYPE TO ZERO
       MOVB     #4.,(9(R0)        ;SET ACTUAL REQUEST TO 4
       RTS   C  PC                ; & RETURN
```

```
;
EOFTST:MOV      8(R0),R1            ;LOAD REQUEST TYPE
       JSR      JTABL5(R1)          ;AND EXECUTE APPROPRIATE ROUTINE
       RTS      PC
;
FEACTV:MOV      @6(R0),R1           ;ACCESS PCE
       MOV      @4(R1),R1           ;ACCESS FE
       MOV      @4(R0),R2           ;ACCESS DATA BUFFER
       CMP      4(R2),(R1)          ;TEST FILE ID (1ST PART)
       BNE      SEARCH              ;IF NOT EQUAL THEN SEARCH FE LISTS
       CMP      6(R2),2(R1)         ;TEST FILE ID (2ND PART)
       BNE      SEARCH              ;IF NOT EQUAL THEN SEARCH FE LISTS
LOOP:  MOV      10(R1),R3           ;LOAD ADDR OF PROFILE TABLE
       BMI      SETMSG              ;IF NEG SET MSG TO ACTIVATE FILE
       MOVB     PTNDEX(R3),R3       ;LOAD 1ST BYTE OF DESCRIPTOR TABLE
       BHI      ALL-OK              ;IF POS FILE IS COMPLETE
FINRTN:MOV      #2.,(R0)            ;SET ADDR OF I/O PROC
       CLR      8(R0)               ;SET REQUEST TYPE & ACTUAL REQUEST TO 0
RETURN:RTS      PC                  ;&RETURN
ALL-OK:MOV      #7.,4(R2)           ;MAVE ACK TO DATA BUFFER
       BR       FINRTN              ;& RETURN
SEARCH:MOV      #4.,INDEX           ;SET UP INDEX FOR SEARCH
       MOV      FELIST,R4           ;LOAD LIST OF FE LISTS HEAD POINTERS
LOOP2: MOV      INDEX(R4),R1        ;LOAD ADDR OF FIRST LIST
LOOP1: CMP      4(R2),(R1)          ;TEST FIRST PART OF FILE ID
       BNE      RESET               ;IF NOT EQUAL THEN GET NEXT ONE
       CMP      6(R2),2(R1)         ;TEST FILE ID (2ND PART)
       BNE      RESET               ;IF NOR EQUA; THEN GET NEXT ONE
       BR       LOOP                ;WE HAVE IT! END
RESET: MOV      16(R1),R1           ;GET PTR TO NEXT LIST
       BEQ      NXTLST              ;IF = NULL THEN WE GET NEXT LIST
       BR       LOOP1               ;OTHERWISE RESUME SEARCH
NXTLST:SUB      #2.,INDEX           ;DECREMENT INDEX
       BMI      NOFIND              ;IF NEG WE FOUND FILE: ERROR
       BR       LOOP2               ; OTHERWISE CONTINUE SEACH ON NEXT LIST
NOFIND:MOV      #25.,4(R2)          ;MOVE NACK IN DATA BUFFER
       BR       FINRTN              ;& RETURN
SETMSG:MOV      R3,10(R2)           ;MOVE AADDR OF PROFILE TABLE TO DATA
                                    ;BUFFER
       MOV      #2.,(R0)            ;SET ADDR OF I/O PROC
       CLRB     8(R0)               ;SET REQUEST TYPE TO 0
       MOVB     #4.,9(R0)           ;SET ACTUAL REQUEST TO 4
       RTS      PC                  ;& RETURN
;
JTABL5:.WORD    FEACTV              ;FILE ELEMENT ACTIVATE
       .WORD    TEST                ;TEST (FE ALREADY ACTIVE)
```

```
;
FILPRI:MOV     @4(R0),R1        ;ACCESS DATA BUFFER
       MOV     10(R1),R2        ;GET DESTINATION OF FILE
       MOV     DESTHD,R3        ;GET A DEST ELEMENT
       MOV     DESTHD,R4        ;SET PTR TO END OF LIST
LOOP:  CMP     R1,R3            ;FTEST IF WE HAVE DESTINATION
       BEQ     FESRCH           ;YS?: THEN LOOK FOR FILE ELEMENT
       MOV     (R3),R3          ;NO?; THEN GET NEXT D.E.
       CMP     R3,R4            ;BACK TO BEGINNING?
       BEQ     NOFIND           ;YS?: THEN WRONG DESTINATION
       BR      LOOP             ;NO?: THEN CONTINUE TESTING
FESRCH:MOV     @2(R3),R3        ;ACCES F.E.
LOOP1  CMP     4(R1),(R3)       ;TEST FIRST PART OF FILE ID
       BNE     NEXTFE           ;IF NOT EQUAL THEN GET NEXT F.E.
       CMP     6(R1),2(R3)      ;TEST 2ND PART OF FILE ID
       BNE     NEXTFE           ;IF NOT EQUAL THEN GET NEXT F.E.
       MOVB    2(R3),4(R2)      ;MOVE PRIORITY TO DATA BUFFER
FINRTN:MOV     #2.,(R0)         ;SET ADDR OF I/O PROC
       CLR     8(R0)            ;SET RREQUEST TYPE & ACTUAL REQUEST TO 0
       RTS     PC               ;& RETURN
NEXTFE:MOV     16(R3),R3        ;GET PTR TO NEXT FE
       BEQ     NOFIND           ;IF NULL END OF LIST
       BR      LOOP1            ;OTHERWISE CONTINUE SEARCH
NOFIND:MOV     #25.,4(R2)       ;MOVE NACK TO DATA BUFFER
       BR      FINRTN
```

```
;
PRICHG:MOV     @4(R0),R1          ;ACCESS DATA BUFFER
       MOV     10(R1),R2          ;GET DESTINATION
       MOV     DESTHD,R3          ;GET A DESTINATION ELEMENT
       MOV     DESTHD,R4          ;SET UP PTR FOR END OF LIST
LOOP:  CMP     R2,R3              ;SEE IF SAME DESTINATION
       BEQ     FESRCH             ;YS?: THEN LOO  FOR F.E.
       MOV     (R3),R3            ;NO?: THEN ACCESS NEXT D.E.
       CMP     R3,R4              ;GONE AROUND LIST?
       BEQ     NOFIND             ;YS?: THEN ERROR
       BR      LOOP               :NO?: THEN CONTINUE
FESRCH:MOV     @2(R3),R3          ;ACCESS F.E.
LOOP1: CMP     4(R2),(R3)         ;TEST FILE ID (1ST PART)
       BNE     NEXTFE             ;IF NOT EQUAL THEN GET NEXT ONE
       CMP     6(R2),2(R3)        ;TEST FILE ID (2ND PART)
       BNE     NEXTFE             ;IF NOT EQUAL THE GET NEXT ONE
       MOVB    13(R1),7(R3)       ;SET NEW PRIORITY
       MOV     #7.,4(R1)          ;MOVE ACK TO DATA BUFFER
FINRTN:MOV     #2.,(R0)           ;SET ADDR OF I/O PROC
       CLR     8(R0)              ;SET RREQUEST TYPE & ACTUAL REQUEST TO 0
       RTS     PC                 ;& RETURN
NEXTFE:MOV     16(R3),R3          ;LOAD ADDR OF NEXT F.E.
       BEQ     NOFIND             ;IF = NULL THEN END OF LIST
       BR      LOOP1              ;OTHERWISE CONTINUE
NOFIND:MOV     #25.,4(R2)         ;MOVE NACK TO DATA BUFFER
       BR      FINRTN             ;& RETURN
```

```
;
WHEFIL:MOV     @4(R0),R1          ;ACCES DATA BUFFER
       MOV     #4.,INDEX          ;SET INDEX
       MOV     FELIST,R2          ;ACCESS HEADS OF F.E. LISTS
LOOP:  MOV     INDEX(R3),R4       ;ACCES A LIST DEPENDING ON INDEX
       MOV     #6.,INDEX1         ;SET INDEX FOR DATA BUFFER
LOOP1: CMP     4(R1),(R3)         ;TEST FILE ID (1ST PART)
       BNE     NEXTFE             ;IF NT EQUAL THEN GET NEXT ONE
       CMP     4(R1),(R3)         ;TEST FILE ID (2ND PART)
       BNE     NEXTFE             ;IF NOT EQUAL THEN GET NEXT ONE
       MOV     6(R3),INDEX1(R1)   ;MOVE STATUS &PRIORITY TO DATA BUFFER
       ADD     #2.,INDEX1         ;INCREMENT INDEX FOR NEXT TIME
NEXTFE:MOV     16(R3),R3          ;ACCESS NEXT F.E.
       BEQ     NXTLST             ;IF =NULL END OF LIST GET NEXT ONE
       BR      LOOP1              ;OTHERWISE CONTINUE SEARCH
NXTLST:SUB     #2.,INDEX          DECREMENT INDEX TO ACCESS NEXT LIST
       BMI     END                ;IF NEG THE WE ARE DONE
       BR      LOOP1              ;OTHERWISE WE CONTINUE ON NEXT LIST
END:   MOV     2 #2.,(R0)         ;SET ADDR OF I/O PROC
       CLR     8(R0)              ;SET REQUEST TYPE & ACTUAL REQUEST TO 0
       RTS     PC                 ;& RETURN
```

```
;
DSTNUM:MOV      @4(R0),R1       ;ACCESS DATA BUFFER
       MOV      10(R1),R2       ;GET DESTINATION
       MOV      DESTHD,R3       ;GET A DESTINATION ELEMENT
       MOV      DESTHD,R4       ;SET PTR FOR END OF LIST
LOOP:  CMP      R2,R3           ;TEST FOR DEST
       BEQ      SETNUM          ;IF EQUAL THEN DONE
       MOV      (R3),R3         ;GET NEXT DESTINATION
       CMP      R3,R4           ;BACK TO BEGINNING?
       BEQ      NOFIND          ;YS?: THEN BAD DEST ID.
       BR       LOOP            ;NO?: THEN COMTINUE
SETNUM:MOV      4(R3),4(R2)     ;MOVE # OF F.E TO DATA BUFFER
FINRTN:MOV      #2.,(R0)        ;SET ADDR OF I/O PROC
       CLR      8(R0)           ;SET ACTUAL REQUEST & REQUEST TYPE TO 0
       RTS      PC              ;& RETURN
NOFIND:MOV      #25.,(4(R2)     ;MOVE NACK TO DATA BUFFER
       BR       FINRTN          ;& RETURN
```

```
;
QUENUM:MOV      @4(R0),R1           ;ACCESS DATA BUFFER
       MOV      NOWAIT,(+R1)        ;MOVE # OF F.E. IN WAIT QUEUE
       MOV      NOINAC,(+R1)        ;MOVE #OF F.E.IN INACTIVE QUEUE
       MOV      NOACTV,(+R1)        ;MOVE # OF F.E. IN ACTIVE QUEUE
                                    ;TO DATA BUFFE
       MOV      #2.,(R0)            ;SET ADDR OF I/O PROC
       CLR      8(R0)               ;SET REQUEST TYPE & ACTUAL REQUEST TO 0
       RTS      PC                  ;& RETURN
```

```
;
NOWAIT:MOV      @4(R0),R1          ;ACCESS DATA BUFFER
       MOV      NOWATQ,(+R1)       ;MOVE # OF F.E. IN WAIT QUEUE TO DATA
                                   ;BUFFER
       MOV      #2,,(R0)           ;SET ADDR OF I/O PROC
       CLR      8(R0)              ;SET RQUEST TYPE & ACTUAL REQUEST TO 0
       RTS      PC                 ;& RETURN
```

```
;
NOINAC:MOV      @4(R0),R1       ;ACCESS DATA BUFFER
       MOV      NOINAQ,(+R1)    ;MOVE # OF F.E. IN INACTIVE QUEUE TO
                                ;DATA BUFFER
       MOV      #2.,(R0)        ;SET ADDR OF I/O PROC
       CLR      8(R0)           ;SET REQUEST TYPE & ACTUAL REQUEST TO 0
       RTS      PC              ;& RETURN
```

```
;
FILDEL:MOVB     8(R0),R1            ;LOAD REQUEST TYPE(=0 1ST PASS,=2 PT
                                    ; IN CORE)
        JMP     JTABL6(R1)
FIRST:  MOV     @4(R0),R1           ;ACCESS DATA BUFFER
        MOV     @6(R0),R2           ;ACCESS PCE
        MOV     @4(R2),R3           ;ACCESS FE
        CMP     4(R1),(R2)          ;TEST FILE ID (1ST PART)
        BNE     SEARCH              ;IF NOT EQUAL FILE NOT ATTACHED
        CMP     6(R1),(R2)          ;TEST FILE ID (2ND PART)
        BNE     SEARCH              ;IF NOT EQUAL FILE NOT ATTACHED
        MOV     12(R2),R3           ;GET ADDR TO PROFILE TABLE
        BMI     FEACTV              ;IF NEG: TABLE NOT IN CORE
        ADD     #4,(R3)             ;ACCESS FILE BIT MAP
        MOV     =MAPLEN,R5          ;LOAD LENGTH OF BIT MAP
        MOV     BITMAP,R4           ;LOAD ADDR OF MASTER BIT MAP
LOOP:   BIS     (+R3),(+R4)         ;OR BOTH MAPS TOGETHER
        DEC     R5                  ;DECREMENT LENGTH LEFT TO DO
        BMI     DONE                ;IF NEG THEN WE ARE DONE
        BR      LOOP                ;OTHERWISE WE CONTINUE
        SUB     MAPLEN+4,R3         ;GO BACK TO BEGINNING OF P.T.
        MOV     R2,(+SP)            ;STACK ADDR OF FE
        JSR     PC,FREEFE           ;&CALL ROUTINE TO FREE FILE ELEMENT
        MOV     R3,(+SP)            ;STACK IT'S ADDR
FREPT:  JSR     PC,FREEPT           ;& CALL ROUTINE TO FREE PROFILE TABLE
SEARCH:MOV      #4,INDEX            ;LOAD INDEX LOCATION
        MOV     FELIST,R2           ;LOAD ADDR OF HEADS OF F.E. LISTS
        MOV     INDEX(R2),R3        ;GO TO FIRST LIST
LOOP1:  CMP     4(R1),(R3)          ;TEST FILE ID (1ST PART)
        BNE     NEXTFE              ;IF NOT SAME THEN GET NEXT ONE
        CMP     6(R1),2(R3)         ;TEST FILE ID (2ND PART)
        BNE     NEXTFE              ;FI NOT SAME THEN GET NEXT ONE
        MOV     12(R3),R4           ;LOAD ADDR OF P.T.
        BMI     FEACTV              ;IF ADDR IS NEG:P.T. IS NOT IN CORE
        MOV     R4,(+SP)            ;STACK ADDR OF PROFILE TABLE
        JSR     PC,FREEPT           ;CALL ROUTINE TO FREEIT
        MOV     R3,(+SP)            ;STACK ADDR OF F.E.
        JSR     PC,FREEFE           ;CALL ROUTINE TO FREE F.E.
        TST     R3                  ;TEST ADDR OF F.E.
        BEQ      NXTLST             ;IF NEG: GET NEXT LIST
        BR      LOOP1               ;OTHERWISE CONTINUE
FEACTV:MOV      R3,4(R1)            ;MOVE ADDR OF PROFILE TABLE TO DATA
                                    ;BUFFER
        MOV     #2.,(R0)            ;SET ADDR OF I/O PROC
        CLRB    8(R0)               ;SET REQUEST TYPE TO 0
        MOVB    #6.,9(R0)           ;SET REQUEST TYPE TO 6
        RTS     PC                  ;& RETURN
NEXTFE:MOV      12(R3),R3           ;GET ADDR OF NEXT F.E.
        BEQ     NXTLST              ;IF = NULL: GET NEXTLIST
        BR      LOOP1               ;OTHERWISE WE CONTINUE
NXTLST:SUB      #2.,INDEX           ;DECREMENT INEDX
        BMI     DONE                ;IF NEG: ALL DONE
        BR      LOOP1               ;OTHERWISE WE SEARCH NEXT LIST
DONE:   CLR     (R0)                ;SET DUMMY MSG
        RTS     PC                  ;& RETURN
SECOND:MOV      @4(R0),R1           ;ACCESS DATA BUFFER
        MOV     10(R1),(+SP)        ;STACK ADDR OF P.T.
        BR      FREPT               ;FREE PORFILE TABLE & SEARCH FOR NEXT
                                    ;F.E.
;
JTABL6:.WORD    FIRST               ;1ST PASS THROUGH THE ROUTINE
       .WORD    SECOND              ;SECOND PASS THROUGH:P.T. IN CORE
```

```
;
FORFIL:MOV    8(R0),R1          ;LOAD REQUEST TYPE
       JMP    JTABL7(R1)        ;AND PROCESS ACCORDINGLY
FILFOR:MOV    @4(R0),R1         ;ACCESS DATA BUFFER
       MOV    10(R1R1),R2       ;ACCESS DESTINATION
       MOV    DESTHD,R3         ;ACCESS A DEST. ELEMENT
       MOV    DESTHD,R4         ;SET PTR FOR END OF LIST
;
LOOP:  CMP    R2,R3             ;SEE IF SAME DESTINATION
       BEQ    FESRCH            ;YS?: THEN GET THE F.E.
       MOV    (R3),R3           ;NO?: THEN GET NEXT D.E.
       CMP    R3,R4             ;TEST IF BACK TO BEGINNING
       BEQ    NOFIND            ;YS?: THEN WE HAVE AN ERROR
       BR     LOOP              ;NO?: THEN CONTINUE SEARCH
;
FESRCH:MOV    @2(R3),R3         ;ACCESS F.E.
LOOP1: CMP    4(R2),(R3)        ;TEST FILE ID (1ST PART)
       BNE    NEXTFE            ;IF NOT SAME: GET NEXT ONE
       CMP    6(R2),2(R3)       ;TEST FILE ID (2ND PART)
       BNE    NEXTFE            ;IF NOT SAME: GET NEXT F.E
       MOV    12(R3),R2         ;LOAD ADDR OF PROFILE TABLE
LOOP2: BMI    ACTVFE            ;IF NEG: TABLE NOT IN CORE
       MOV    R2,4(R2)          ;MOVE P.T. TO DATA BUFFER
       MOV    #2.,(R0)          ;SET ADDR OF I/O PROC
       CLRB   8(R0)             ;SET REQUEST TYPE TO 0
       MOVB   #10.,9(R0)        ;SET ACTUALTREQUEST TO 8
       RTS    PC                ;& RETURN
;
ACTVFE:MOV    R2,4(R1)          ;LOAD ADDR OF P.T. INTO DATA BUFFER
       MOV    #2.,(R0)          ;SET ADDR OF I/O PROC
       MOVB   #2.,8(R0)         ;SET RQUEST TYPE TO 2
       MOVB   #10,9(R0)         ;SET ACTUAL REQUEST TO 8
       RTS    PC                ;& RETURN
;
NEXTFE:MOV    16(R3),R3         ;GET ADDR OF NEXT FE
       BEQ    NOFIND            ;IF = NULL THEN END OF LIST
       BR     LOOP1             ; OTHERWISE CONTINUE SEARCH
NOFIND:MOV    #2.,(R0)          ;SET ADDR OF I/O PROC
       CLR    8(R0)             ;SET REQUEST TYPE & ACTUA; REQUEST TO 0
       MOV    #25.,4(R1)        ;MOVE NACK TO DATA BUFFER
       RTS    PC                ;& RETURN
;
FELINK:MOV    @4(R0),R1;ACCESS DATA BUFFER
       MOV    4(R1),R2          ;GET PTR TO F.E.
       MOV    14(R2),R2         ;GET LINK TO NEXT FILE
       BEQ    DONE     ;IF = NULL:NO OTHER FILE
       MOV    @12(R2),R2        ;OTHERWISE GET THE F.E.
       BR      LOOP2            ;AND RESUME PROCEDURE
;
JTABL7:.WORD  FILFOR            ;MAIN ROUTINE TO FORWARD FILE
       .WORD  FELINK            ;ROUTINE FOR FORWADING LINKED FILES
```

A SHORT DESCRIPTION OF THE SPOOL ALGORITHMS

I- FILE CONTROL

***-ACCEPT FILE X ?

Data buffer contains File Id & File Descriptor.
Implementation: only one file can be sent per virtual sign-on

-Ask File Management processor for some space on disk(1 logical block).

-If none available :END

-Otherwise send addr of block to I/O processor, along with addr of File Element & addr of Profile Table.

-I/O processor sends ACK to suscriber, write Profile Table on file & puts the File Element on the wait queue.

-END.

***-HERE COMES FILE X !

-File Management processor puts File element on the active queue (if it is not already there).

-I/O processor gets a D.A.D Device Control Table and opens the file.

```
    -It then reads the Profile Table & chains it to the File Element
    -Does it need a new block?
    LOOP: -No: then write data
          -Yes: then ask File management processor for one
                & put File Element on the wait queue
```

-File Manager tries to get a block of storage

```
    -Got it?
       Yes:then return to I/O processor
       NO: delete the file and send ABORT msg to suscriber
```

-Return to I/o processor

```
    -go to LOOP
```

-Data read done:

```
      -Deallocate Device Control Table
      -Send ACK to suscriber
      -Write Profile Table onto file
      -Set it's addr. in File Element.
      -Put File Element on inactive queue
```

-END.

```
***-HAVE YOU RECEIVED FILE X ?

-File Element active ?
LOOP:YES?:-Access Profile Table
        -Get Descriptor Table .
        -EOF tag on?
            -YES?: Send ACK to suscriber
            -NO ?: Send NACK to suscriber

   NO ?:-Give addr of Profile Table to I/O processor
        -It reads Profile Table from disk and returns it's addr
         to the File Manager.
        -Go to LOOP.

-END.

-END.
```

II- FILE STATUS ENQUIRIES AND STATUS CHANGES

***-PRIORITY OF FILE X ?

-File Manager:
- -If given destination of file: Scan Destination Element queue for File Element.
- -If not given destination: Search all queues (active, inactive, or wait) for file.
- -If file not found: send NACK to suscriber
- -If file is found: send priority to suscriber

-END.

***-SEND DESCRIPTOR OF FILE X.

-File Manager:
- -Scan Destination Element queue then the File Element queue for file
- -File active ?: -Yes: -Get ptr to Profile Table
  - -Get Descriptor Table
  - -Give it to the I/O processor for send off
- -No : -Get addr of Profile Table on disk
  - -Have I/O processor read it into core
  - -On return, get decriptor Table
  - -Give it to I/O processor for send off

-END.

***-CHANGE PRIORITY OF FILE X TO Y.

-File Manager:
- -Scan Destination Element queue then the File Element queue for file.
- -Change priority of File Element.
- -Send ACK to suscriber

-END.

***-WHERE IS FILE X (STATUS REPORT ON FILE)

-File Manager:
- -Scan Destination Element queue then the File Element queue for file.
- -If not found: END , send NACK to suscriber
- -If found: put status in data buffer

-END.

III- STATUS REPORTS ON QUEUES AND FILES.

***-HOW MANY FILES ON DESTINATION QUEUE ?

-File Manager:
- -Search for Destination Element.
- -If not found: send NACK to suscriber
- -If found : Get File Element count
  Give it to I/O processor for send off

-END.

***-HOW MANY FILES ON QUEUES ?

-File Manager:
- -Move number of elements on -inactive queue
  -active queue
  -wait queue
  to data buffer.
- -Have I/O processor send it off to suscriber

-END.

***-HOW MANY FILES ON WAIT QUEUE ?

-File Manager:
- -Move number of File Elements on wait queue to data
  data buffer
- -Have i/O processor send it off to suscriber.

-END.

***-HOW MANY FILES ON THE INACTIVE QUEUE ?

-File Manager:
- -Move Number of File Elements on inactive queue to
  data buffer
- -Have I/O processor send it off to suscriber.

-END.

IV- FILE DISPOSITION COMMANDS.

***-DELETE FILE X.

-This algorithm is in the process of being redesigned !!!

***-FORWARD FILE X.

-Search Destination Element queue and File element queue for file.

-if not found: send NACK to suscriber .

-if found :-File active?
LOOP :-YES?:-Get addr of file through Profile Table
-Give it to I/O processor for send off
-Put File Element on wait queue on return

-NO ?:Give addr of Profile Table to I/O processor
-It activates file by reading Profile Table into core. Returns addr of Profile Table to File Manager.
-Go to LOOP.

-File link .eq. null ?
-YES?: -END.
-NO ?: -Update ptr to next File Element
-Go to LOOP

***-ROUTE FILE X TO DESTINATION Y.

-Given old destination.

-Search Destination element queue and File Element queue for old destination

-If not found : Send NACK to suscriber.
-If found : -Put it on new destination element queue
-Generate a 'FORWARD FILE' msg

-END

```
;***********************************************************************
;*                                                                     *
;*                                                                     *
;*     N E T S P O O L   M A S T E R   S C H E D U L E R               *
;*                                                                     *
;*                                                                     *
;* DATE:MAY 15TH,1972                                                  *
;* AUTHOR: MARC DUFRESNE                                               *
;* VERSION: 1   LEVEL: 0        LAST UPDATE: MAY 15TH,1972             *
;*                                                                     *
;*                                                                     *
;***********************************************************************
;
;
        R0=%0                           ;REGISTER 0, WORK REG.
        R1=%1                           ;REGISTER 1 ,NOT USED
        R2=%2                           ;REGISTER 2 ,NOT USED
        R3=%3                           ;REGISTER 3 ,NOT USED
        R4=%4                           ;REGISTER 4 ,NOT USED
        R5=%5                           ;REGISTER 5 ,NOT USED
        SP=%6                           ;REGISTER 6, STACK POINTER
        PC=%7                           ;REGISTER 7, PROGRAM COUNTER
;
;
        .CSECT
        .TITLE  MASTER
MASTER:MOV      LSTONE,R0               ;LOAD LAST PROCESSOR SCHEDULED
        ADD     #2.,R0                  ;SCHEDULE THE NEXT ONE
        CMP     #6.,R0                  ;TEST IF IT IS INVALID ID
        BEQ     RESET                   ;IF ID > 4 THEN RESET
;
BACK:   MOV     R0,-(SP)                ;OTHERWISE PUSH ID ON STACK
        MOV     R0,LSTONE               ;AND UPDATE THE CHECK
        RTS     PC                      ;AND RETURN
;
RESET:  CLR     R0                      ;CLEAR THE RESULT
        BR      BACK                    ;AND GO TO THE END ROUTINE
;
LSTONE:.WORD    0.
        .END
```

```
;**********************************************************************
;*                                                                    *
;*                                                                    *
;*            N E T S P O O L     N U C L E U S                       *
;*                                                                    *
;*                                                                    *
;* DATE:MAY 13TH,1972                                                 *
;* AUTHOR: MARC DUFRESNE                                              *
;* VERSION: 1    LEVEL: 2          LAST UPDATE: MAY 15TH,1972         *
;*                                                                    *
;*                                                                    *
;**********************************************************************
;
;
        R0=%0                           ;REGISTER 0
        R1=%1                           ;REGISTER 1
        R2=%2                           ;REGISTER 2
        R3=%3                           ;REGISTER 3
        R4=%4                           ;REGISTER 4
        R5=%5                           ;REGISTER 5
        SP=%6                           ;REGISTER 6, STACK POINTER
        PC=%7                           ;REGISTER 7, PROGRAM COUNTER
;
;
        .CSECT
        .TITLE  NUCLUS
        .GLOBL  IOQPTR,IOPROC
        .GLOBL  FMQPTR,FMPROC           ;GLOBAL DECLARATIONS
        .GLOBL  CMQPTR,CMPROC
        .GLOBL  MASTER
;
SCHDUL:JSR      PC,MASTER               ;CALL MASTER SCHEDULER TO
                                        ;SCHEDULE NEXT PROCESSOR
        MOV     (+SP),R0                ;PICK UP IT'S ID OF THE STACK
        JSR     PC,JTABL0(R0)           ;AND CALL IT (0=I/O PROC, 2=FILE
                                        ;MANGEMENT PROC, 4=COMMUNICATION
                                        ;PROC)
        MOV     (+SP),R0                ;GET ADDR OF MSG RETURNED BY PROCESSOR
        MOV     (R0),R1                 ;AND SAVE THE PROC ID IN R1
        BEQ     FREMSG                  ;IF ID=2 RETURN TO THE FREE MSG POOL
        JMP     JTABL1-2(R1)            ;OTHERWISE WE BRANCH TO THE
                                        ;SUITABLE ROUTINE TO PLACE THE
                                        ;MESSAGE ON THE RIGHT REQUEST QUEUE

SETIOQ:MOV      IOQPTR,R1               ;MOVE ADDR OF LAST ELEMENT IN QUEUE TO
                                        ;R1
        MOV     R0,IOQPTR               ;AND RESET THE CHAIN POINTER
        MOV     (R0),(+R1)              ;AND CHAIN THE MSG TO IT(2ND WORD IN)MSG
        BR      SCHDUL                  ;AND WE RESCHEDULE OURSELVES
;
SETFMQ:MOV      FMQPTR,R1               ;LAST ELEMENT OT FILE MAN QUEUE LOADED
        MOV     R0,FMQPTR               ;AND RESET THE CHAIN POINTER
        MOV     (R0),(+R1)              ;CHAIN MESSAGE TO IT(2ND WRD IN MSG)
        BR      SCHDUL                  ;AND RESCHEDULE
;
SETCMQ:MOV      CMQPTR,R1               ;LAST ELEMENT OF COMM QUEUE LOADED
        MOV     R0,CMQPTR               ;AN RESET CHAIN POINTER
        MOV     (R0),(+R1)              ;WE CHAIN THE MSG TO IT(2ND WORD IN MSG)
        BR      SCHDUL                  ;AND WE RESCHEDLUE
```

```
;
FRQPTR:.WORD   FRQHED              ;PTR TO HEAD OF FREE MSG. BUFFER QUEUE
IOQPTR:.WORD   IOQTL               ;PTR. TO TAIL OF I/O PROC. MSG QUEUE
FMQPTR:.WORD   FMQTL               ;PTR. TO TAIL OF FILE MAN. PROC MSG QUEU
CMQPTR:.WORD   CMQTL               ;PTR TO TAIL OF COMM. PROC. QUEUE
;
JTABL0:.WORD   IOPROC,FMPROC,CMPROC     ;JMP TABL TO ACCESS MASTER SCHE.
JTABL1:.WORD   SETIOQ,SETFMQ,SETCMQ     ;JMP TABL FOR MSG PROCESSING
       .END    SCHDUL              ;BEGIN EXEC AT SCHDUL.
```

DIAGRAM FOR INTERPRETATION OF CODES