A computer network model using concurrent process synchronization.

Randy Richard Boldosser

Follow this and additional works at: http://preserve.lehigh.edu/etd
Part of the Computer Sciences Commons

Recommended Citation
A COMPUTER NETWORK MODEL USING
CONCURRENT PROCESS SYNCHRONIZATION

by
Randy Richard Boldosser

A Thesis
Presented to the Graduate Committee
Of Lehigh University
in Candidacy for the Degree of
Master of Science
in
Computing Science

Lehigh University
1982
This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

Dec. 6, 1982
(date)

Professor in Charge

Head of Division
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 Introduction</td>
<td>3</td>
</tr>
<tr>
<td>2.0 The Computer Environments</td>
<td>7</td>
</tr>
<tr>
<td>2.1 The RDOS Environment</td>
<td>7</td>
</tr>
<tr>
<td>2.2 The AOS Environment</td>
<td>9</td>
</tr>
<tr>
<td>3.0 The Network Processes</td>
<td>13</td>
</tr>
<tr>
<td>3.1 The MSFIT Process</td>
<td>14</td>
</tr>
<tr>
<td>3.2 The MCAIO Process</td>
<td>26</td>
</tr>
<tr>
<td>3.3 The MCABF Process</td>
<td>37</td>
</tr>
<tr>
<td>3.4 The SEND Process</td>
<td>48</td>
</tr>
<tr>
<td>3.5 The DISPATCH Process</td>
<td>54</td>
</tr>
<tr>
<td>4.0 The Dynamics of the Network</td>
<td>59</td>
</tr>
<tr>
<td>5.0 Conclusion</td>
<td>65</td>
</tr>
<tr>
<td>References</td>
<td>66</td>
</tr>
<tr>
<td>Vita</td>
<td>68</td>
</tr>
</tbody>
</table>
## TABLE OF FIGURES

1. The Computer Network:
   - Process Overview: Page 6

2. The MSFIT Process:
   - MSFIT Task Flow: Page 17
   - RTERM Task Flow: Page 18
   - OSTOP Task Flow: Page 19

3. The MCAIO Process:
   - MCAIO Task Flow: Page 30
   - RDMCA Task Flow: Page 32

4. The MCABF Process:
   - MCATSK Task Flow: Page 40
   - SDASC and SDRND Routine Flow: Page 42

5. The SEND Process:
   - SENDTSK Task Flow: Page 52

6. The DISPATCH Process:
   - DISPATCH Process Flow: Page 57
ABSTRACT

In an integrated circuit manufacturing environment there are many phases of production that require the aid of computers to perform many of the manufacturing processes. To coordinate these phases, computers must be networked to provide information between production processes. This thesis discusses one network that connects several manufacturing processes controlled by minicomputers and a larger computer used for monitoring production flow.

Given an environment where interruption of file transmission is very likely, the network must be designed to include adequate handshaking and error checking to accommodate accurate transmissions. Conflicting with this requirement is the necessity to transmit large amounts of information in short time periods. This thesis describes a network that transmits 100,000 bits of information per second.

The network is designed so that users accessing the network have the capability of real-time file transmission. Since the operating systems used within the network can handle code for the concurrent execution of subtasks within processes, deadlocks are prevented by emulating device timeouts. By using this feature, one subtask can be

Copyright 1982 by Western Electric Company, Inc.
dedicated to monitoring network availability while another subtask can report this information to the network users.

Along with describing the architecture of the network mentioned above, this thesis emphasizes the need for using concurrent processes in this network rather than employing a large single process to handle all network protocols. For example, overlays were not used by the programmer because overlays can not be memory resident simultaneously. Instead, multiple process synchronization is used since the architecture of the operating system allows for concurrent memory resident processes.

The final section of this thesis describes the dynamic aspects of file transmission across the network. Emphasis is placed on exceptional conditions during transmission and how these conditions are handled by the network software.

The network described in this thesis has been in operation at a manufacturing facility since July, 1980. Although certain aspects of the network are tailored to the needs of this specific manufacturing location, the concepts of concurrent process synchronization provide a framework for other applications.
1.0 INTRODUCTION

This thesis describes the computer networking structure designed to transmit files from one central master computer to several slave computers. The computers used for this network are manufactured by Data General Corporation and are all sixteen bit minicomputers. Each minicomputer consists of four registers, two for general purposes and two for indexed addressing, main memory up to 1 megawords, and various peripherals. Processes are limited to 32 kilowords of main memory and larger processes must be broken into overlays. The minicomputers employ a Memory Allocation and Protection unit (MAP) that permits processes to be mapped into any memory segment. All Data General computers use software stacks for process execution. These stacks are contained within the main memory allocated to the process. Since the hardware is designed around the stack concept, all software written for Data General computers is stack oriented. This implies that such languages such as FORTRAN and assembly language can be recursive.

The Central Processing Unit (CPU) of Data General minicomputers handles peripheral devices by using a priority interrupt system. Each peripheral is connected to the interrupt request line of the CPU. When a peripheral requires servicing, it sends a signal across the interrupt request line. If the CPU is servicing an interrupt of lower
priority or handling no interrupt at all, the device driver for the peripheral takes control of the CPU until the interrupt is serviced or another interrupt of higher priority is encountered. On the minicomputers, disk drives are used for backing store and are given the highest interrupt priority. High data rate peripherals such as communication links and secondary disk drives are assigned slightly lower priorities. Human interface peripherals such as terminals and printers are assigned the lowest interrupt priorities.

The master computer of the network uses an operating system called the Advanced Operating System (AOS). This system can schedule 64 memory resident processes. Each of these 64 processes can also be divided into several subtasks which can execute concurrently within different sections of the process spaces. The slave computers use a less powerful operating system called RDOS, or Real-Time Disk Operating System. RDOS can run only two processes at the same time. The RDOS operating system, like AOS, allows these two processes to be divided into several subtasks that can execute concurrently.

All files used during transmission originally reside on disk drives interfaced to the AOS master computer. These files are transmitted to the RDOS slave computer disk drives by using the software described in section three. That piece
of hardware which does the file transmissions is called the MCA, or Multiprocessor Communications Adapter. This MCA is a 6.2 Megabit per second parallel bus configured in a "star" arrangement. Each computer or node has one bus going into a central piece of hardware that directs all communication to the appropriate bus for transmission. If one node is not in operation, the other nodes are not affected (See Figure One).

The entire network is designed so that a number of file transmissions can occur simultaneously. Several machines can receive files from the master computer at once since a unique subtask within an AOS process manages each separate file transmission.

The following section discusses the environments of the AOS and RDOS operating systems in more detail. Also, the design restrictions of each system are discussed in the next section. Section three of the thesis describes the software architecture of the network. Section four describes the dynamics of concurrent process synchronization within the network.
THE COMPUTER NETWORK: PROCESS OVERVIEW

FIGURE 1
2.0 THE COMPUTER ENVIRONMENTS

This section describes each of the operating systems and their capabilities in some detail. The requirements placed on the network are also listed in this section.

2.1 THE RDOS ENVIRONMENT

The slave computers are the machines involved in the manufacturing process. These minicomputers operate several units of hardware by using a process which executes under the RDOS operating system. Since this manufacturing process must execute continuously, the networking process must be designed so that it occupies the only other available process space under RDOS. However, there are many instances when users may want to execute other processes not related to file transmission. Therefore, the master computer cannot assume that the RDOS process is always executing. Also, because of limitations in the RDOS system, the AOS computer cannot start execution of the networking process even when there are no other processes using the available RDOS process space. Therefore, the network depends on users allowing the network process to execute if no other function requires the RDOS process space.

The manufacturing process and the network process can execute under RDOS concurrently since they have no
requirement for simultaneous resource sharing. When the network process has finished receiving a file, this file becomes available for the manufacturing process. After file transmission, the network process does no further manipulation of the file so that the data remains unchanged throughout its existence on the RDOS machine.

Processes on the RDOS system can be interrupted by using a control character sequence entered into the terminal which maintains the input and output of the process. The sequence generates an interrupt that will cause the executing process to stop immediately. This interrupt may damage file transmission integrity. To alleviate this problem, the control character sequence must be captured and the interrupt must be managed so that file transmission is stopped only when the entire contents of the file has been sent to the RDOS machine.

Adequate handshaking is the final consideration on the RDOS computers. If the AOS master computer stops the execution of its MCA handler routines, the necessary handshaking must occur on restart of the AOS program between the AOS and RDOS computers so that the master computer can determine which RDOS slave computers are available for transmission. This also implies that the RDOS process receiving input from the AOS computer must never be in a process state where it cannot restart all file processing.
and remove partially transmitted files.

The above described process which handles all MCA communications on the RDOS slave computers is called MSFIT. MSFIT is a program that can be multitasked when necessary. Since MSFIT allows for terminal I/O, the RDOS user can enter commands that aid in determining the status of the MSFIT process and also provide a means of terminating MSFIT gracefully. By entering the command "STOP," the user starts a sequence of handshaking that not only determines whether files are currently being transmitted, but also sends information back to the AOS computer to notify future network users that file transmission to this RDOS machine is not possible. A complete description of MSFIT can be found in section 3.1.

2.2 THE AOS ENVIRONMENT

The AOS side of the network has many more responsibilities than the RDOS side since the AOS computer must determine which files are to be sent and then start the transmission of these files. The AOS system must know which RDOS machines are available for file transmission as every MCA I/O instruction must run to completion. If the MCA I/O instruction cannot be completed, the process or task handling the MCA I/O suspends execution until the I/O instruction is satisfied. The AOS operating system does not
allow any input or output with timeout capabilities. Since the processes on the AOS system can be multitasked, such limitations can be bypassed by using two tasks to emulate a timeout function.

The scope of the programming necessary to complete this network requires a large amount of computer memory for file transmission. Hence, the AOS process space is not large enough to accommodate all aspects of the network. The use of overlays is necessary since AOS does not allow virtual memory operation. However, overlays require disk to memory swaps each time an overlay is called. As the AOS operating system allows up to 64 processes to be resident in memory at the same time, the network functions are divided into several AOS processes in order that all aspects of file transmission can be resident in memory and thus allowing faster file transmission. Synchronization of AOS processes that are all resident in memory is managed by AOS interprocess communications (IPC).

Because several slave RDOS computers exist for the single master AOS computer, an entire set of networking processes are necessary for file transmission to each individual RDOS machine. Instead of having redundant processes, the network is built using processes that have several subprocesses or tasks executing concurrently within the memory space allocated for a single process. This
mechanism of multitasking the subroutines contained within the processes allows for simultaneous file transmissions to different machines.

Communications between the AOS processes are performed by two methods of interprocess communication. To maintain synchronization of the AOS processes and make common information available to all of them simultaneously, the master computer uses a shared memory facility built within AOS. By using shared memory, several processes can share read and write information in a common region. All locations of this shared memory are available to any process that opens the memory segment for access. Therefore, a lockout scheme must be used so that sensitive data areas allow read and write access by only one process. The architecture of the information system has been designed with this capability. It is not the intention of this thesis to detail this lockout mechanism but merely to mention that the AOS processes do have the capability to synchronize their activities by using this shared memory facility.

The second method used for interprocess communications is the IPC mailbox. An IPC mailbox is simply a temporary disk file that is available for receiving messages from other processes. The IPC mailboxes handle incoming messages by placing each message on a queue. Each IPC receive request is satisfied by the earliest IPC message waiting on the
queue. An AOS process can receive messages from a specific process or from any process. Most AOS processes in the information system use global IPC mailbox reads since several tasks internal to a specific AOS process may be trying to transmit a message.

There are four processes used to transmit files from the AOS computer to the RDOS computer. These processes are: DISPATCH, SEND, MCABF, and MCAIO. Each one of these processes has multitasking capabilities which permit simultaneous file transmission to more than one RDOS machine. The function of each process is described in detail in sections 3.2 through 3.5.
3.0 THE NETWORK PROCESSES

Of the five processes involved with the network, one process is resident on the RDOS computer and four are resident on the AOS computer. The RDOS process MSFIT performs no actions independent of communication protocols with the AOS processes. The four AOS processes become active as a result of a file transmission request entered by an operator. These requests are entered into an AOS computer terminal and sent to the DISPATCH process. The DISPATCH process determines that the operator command is a valid network request and then starts the SEND process. The SEND process parses the operator command and correlates a list of files to be sent to the RDOS machine. After SEND generates this list of files, the process sends an activation message to the MCABF process. This process produces the necessary buffers used by MSFIT to do the disk I/O on the RDOS computer. After MCABF generates a buffer to be sent, an Interprocess Communications (IPC) message containing the buffer is transmitted to MCAIO, the process that handles the MCA communications. The processes remain active throughout this activity until all files contained within the list generated by SEND have been transmitted. On completion of file transmission, all processes suspend execution by waiting on an IPC mailbox receive request.

The above description is a summary of the functions of
the five processes within the network. The following sections discuss each process in detail.

3.1 THE MSFIT PROCESS

The primary function of the MSFIT process is to manage the RDOS disk I/O necessary to load files on the manufacturing machine. MSFIT waits for a buffer to be sent across the MCA. After receiving the buffer, MSFIT determines which disk I/O function to perform by looking at the command word contained within the buffer. The MSFIT buffer contains 261 words. The first word in the buffer is used by MCAIO for AOS addressing purposes. The second word of the buffer contains the disk I/O channel of the RDOS file being accessed. The third word of the buffer is the MSFIT command word that describes commands such as open an RDOS file and return a file channel, close an RDOS file channel, write a line to the RDOS file, write a 256 word buffer to the RDOS channel, and rename the RDOS file. The fourth word of the buffer is used by MSFIT to return the completion status of the MSFIT task. If MSFIT is successful in completing its requested function, MSFIT returns a value of one. If a disk I/O error occurs during the MSFIT function, MSFIT returns the error code associated with the disk I/O error in word four.

The remainder of the buffer has a variable format.
depending on the MSFIT function requested. For such functions such as file channel opens, words five through 20 contain the name of the file to be opened plus some additional information about the open accessing mode. The remaining 241 words of the buffer are not used for file channel open functions. For file channel close functions, no additional information is sent in words five to 261. The renaming function uses words 8 to 20 for the original name of the RDOS file and words 21 to 33 for the new name of the file. For RDOS file writes, the text of the file to be written is contained in words 5 to 46 for ASCII file writes or words 5 through 261 for binary file block output.

All 261 words of the buffer are sent even though many words are not used by some commands. This helps maintain integrity during transmissions across the MCA. For most commands, MSFIT reads the buffer, does the operation, and returns the completion status code in the buffer by transmitting the entire buffer back to MCAIO. Some commands have eliminated this additional handshaking by simply requiring that MSFIT send a buffer back to MCAIO when an RDOS disk I/O error occurs (See Figure Two).

Along with the disk I/O and the MCA communications, MSFIT must monitor computer terminal activity. To do this, MSFIT starts a task to handle this terminal I/O. This task, called RTERM, is executing concurrently with the main MSFIT.
THE MSFIT PROCESS

FIGURE 2
MSFIT TASK FLOW

SEND -1 FLAG TO MCAIO

READ MCA

PERFORM REQUESTED RDOS DISK I/O

HANDSHAKING REQUIRED?

ERROR OCCURRED?

WRITE TO MCA

THE MSFIT PROCESS

FIGURE 2
(CONTINUED)
THE MSFIT PROCESS
FIGURE 2
(CONTINUED)
OSTOP TASK FLOW

SEND-2 FLAG TO MCAIO

DETERMINE NUMBER OF RDOS FILE CHANNELS OPEN

YES

FILES OPEN?

NO

NOTIFY OPERATOR

WAIT 30 SECONDS

SEND-3 FLAG TO MCAIO

TERMINATE ENTIRE MSFIT PROCESS

THE MSFIT PROCESS
FIGURE 2
(CONTINUED)
task. RTERM's function is to accept input from the RDOS computer terminal, determine if the input is a valid command, and perform the function requested by the command. The two valid commands are STAT and STOP. When an operator enters the STAT command, RTERM looks in a common memory area in MSFIT where a list of all open RDOS file channel numbers are kept. If any channel numbers are listed, this implies that MSFIT is doing the necessary disk operations for file transmission. Since a file transmission should not be interrupted, the RTERM task returns a message to the terminal stating that MSFIT is busy transmitting a file. If no file channel numbers are listed in the common memory area, the RTERM task indicates that MSFIT is inactive.

The second command accepted by RTERM is STOP. The STOP command does a graceful shutdown of the MSFIT process. When RTERM receives a STOP command, the task starts another concurrent task called OSTOP. The OSTOP task reads the list of file channel numbers in the common memory area and if no channels are open, OSTOP issues a process termination request to kill the entire MSFIT process. If there are file channels open, OSTOP suspends for 30 seconds. After 30 seconds, OSTOP once again checks the common memory area for open file channels. If some files are still open, OSTOP suspends itself again. OSTOP continues until it finds no open file channels. The operator is informed every 30
seconds of the task's findings and is requested to wait until all file channels are closed.

The above STOP sequence is the recommended way to guarantee file integrity on the RDOS disk. If MSFIT were to stop during transmission, incomplete files would exist on the RDOS disk. Since a process interrupt capability does exist under RDOS, it is also necessary that additional steps be taken to prevent premature MSFIT process termination. The RDOS system accepts a Control-A character input from a computer terminal as a valid interrupt. This interrupt is normally passed to the operating system which in turn stops the executing process. To avoid this interrupt, MSFIT captures the Control-A character within a task called CNTRLA. If the operator types a Control-A character, the system has been designed to start the CNTRLA task to handle the interrupt. When started, CNTRLA prints a message on the terminal indicating that the Control-A interrupt has been disabled. This reminds the RDOS operator that he should use the STOP command to terminate MSFIT gracefully.

The main task of MSFIT does the disk operations requested. The MCA buffers are read by this task and execution is directed to the portion of the MSFIT task that handles the received command. After doing the function requested, this main task may return the buffer across the MCA. The task then waits for the next MCA message. Since
this loop is sequentially performed within a single task, MSFIT cannot become confused between requests. Each request is handled sequentially in the order that the request is received. Additional requests made to a particular MSFIT are spooled by the MCA hardware so that no messages are lost.

On initial execution of the MSFIT process, some handshaking is required to notify the MCAIO process of the RDOS machine status. This status is necessary because MCAIO cannot issue an MCA write to an inactive RDOS machine. If this happens, the MCAIO process suspends its execution and cannot handle any more network requests until the waiting MCA write request is satisfied. As stated earlier, the AOS and RDOS systems do not support a system timeout feature. Therefore, it is always the responsibility of MSFIT to notify MCAIO of its status. To do this, MSFIT opens the MCA channel and transmits a 261 word buffer immediately to MCAIO. This buffer contains two important pieces of information. First, the RDOS machine identification is sent to inform MCAIO which RDOS machine is available for transmission. Since MCAIO has only a global MCA read outstanding, this process has no way of knowing which RDOS machine has performed the initial handshaking. This is a limitation of the two operating systems and because of this limitation, all messages transmitted across the MCA must contain a return address within the body of the message. The
other piece of information passed during initial execution of MSFIT is the "RDOS machine available" status code. This code, represented by a -1, directs MCAIO to set the proper flags within its common process region so that further network requests can be handled accordingly.

Other handshaking is also necessary during MSFIT process termination. MCAIO requires that MSFIT inform MCAIO when MSFIT is about to terminate. MSFIT does this within the OSTOP task. After OSTOP is started, the task transmits a -2 flag back to MCAIO. This informs MCAIO that the MSFIT on this particular RDOS machine is attempting to shut down. Therefore, the -2 flag indicates that any future network requests that involve opening another RDOS file channel on the MSFIT concerned should be prevented. MCAIO does allow other types of file transmission requests to continue since MSFIT cannot terminate until all current files have been fully transmitted. When OSTOP determines that all file channels are closed on the RDOS side, OSTOP transmits a -3 flag to MCAIO directing MCAIO to prevent any further MCA communications with this particular MSFIT. By using these three flags, MCAIO is always informed of the activities affecting MSFIT execution.

There is one final aspect of MSFIT that requires discussion. There are many times when termination of the information system occurs at intervals when MSFIT is still
operational. To allow currently operational MSFIT processes to continue their execution after the information system on the AOS computer has been stopped and restarted, the initial handshaking required of MSFIT by MCAIO must be repeated. Because MSFIT does not know when the AOS information system is stopping, MSFIT is in a loop where it is waiting for a valid RDOS disk I/O request. To force the handshaking between MCAIO and MSFIT, MCAIO sends MSFIT an automatic restart request. When MSFIT receives such a request, it closes any file channels currently open and sends the initial -1 flag back to MCAIO. The MSFIT restart includes closing of the file channels currently open because the AOS information system is in a state where it does not remember which files were being transmitted. If these files are forgotten and the MSFIT file channels left open, all future operator requests for a MSFIT shutdown cannot be performed because the forgotten file channels are permanently left open. If the AOS information system terminates during file transmission, it is probable that serious problems have developed on the AOS computer. In this case, the files transmitted are incomplete, and therefore, the best method to eliminate these files is to close their file channels and delete the incomplete files. Because of the structure of the network, it is MCABF's responsibility to delete these incomplete files. The purpose of MSFIT is to perform the disk I/O functions without further interpretation of the
MCABF requests. This increases the execution speed of MSFIT and reduces the CPU demand on a computer whose primary responsibility is manufacturing and not computer networking.

Since the automatic restart function is handled in the context of regular RDOS disk functions, the MSFIT main task is given the responsibility of performing automatic restarts. No other tasks within MSFIT are affected by MSFIT restarts. Note that RDOS operating system limitations prevent initiation of MSFIT directly from a disk image. The automatic restart capability only exists for MSFIT processes that have been executing under RDOS before the AOS information system restart. If the MSFIT process on a particular RDOS machine is not active, MCAIO can assume that the RDOS machine is not available and maintains a default -3 flag for that particular RDOS machine.
3.2 THE MCAIO PROCESS

The AOS process handling all MCA communications is the MCAIO process. This process is designed to provide fast throughput of MSFIT command information. One design constraint of this process is that MCAIO must be a memory resident process even when it is inactive. The AOS operating system requires that all processes communicating with the MCA hardware be memory resident. This restriction seems necessary because frequently AOS must service the fast data rates of the MCA hardware. Since MCAIO must always be memory resident, it becomes desirable to limit the amount of memory used by MCAIO. Therefore, the capabilities of the MCAIO are limited. The contents of the MSFIT command buffer are not generated in MCAIO. This function has been ported to an intermediate process called MCABF. MCAIO is only given the intelligence necessary to be able to direct the MSFIT command buffer to the appropriate RDOS machine and additionally to determine whether an RDOS machine is available for network communications.

The above simple responsibilities reduce MCAIO's network role to that of the AOS computer "gate-keeper." All communications received by MCAIO are simply directed to the appropriate task in MCABF. Therefore, MCAIO has a bidirectional capability. Furthermore, some MSFIT buffers sent through MCAIO do not have an accompanying response.
buffer. This implies that all communications through MCAIO should be handled asynchronously.

Asynchronous MCA communications are provided by MCAIO by using the following method. Since multitasking capabilities exist on AOS two tasks can be executing concurrently. One task handles buffer transmissions from the AOS environment to the MCA output channel and the other task handles communications from the MCA input channel to the AOS environment. The main task within MCAIO manages the MCA output communications and a task called RDMCA deals with messages received from the MCA. Each task can work independently and requires no intertask synchronization.

The RDMCA task uses a global MCA receive request to accept MCA input. This enables RDMCA to accept input from any RDOS MSFIT program in the network. Because RDMCA does not know which MCA channel has sent the MSFIT responses, RDMCA must determine this information from the context of the received buffer. The information contained within this response buffer is examined later.

The activities of the main task of MCAIO are discussed initially since information must be passed to the RDOS machines before any responses can be received by RDMCA. When the AOS information system becomes active, the MCAIO process is scheduled for execution. The main task within MCAIO
becomes active and immediately opens the MCA input and output channels. After opening the MCA channels, the main task starts another independent task for each RDOS machine located on the network. These tasks are called RESTART. A RESTART task has the job of sending the automatic restart command to the RDOS MSFIT program. If the MSFIT program is not executing on a particular RDOS machine, the MCA output request from RESTART does not get satisfied. Since there are no timeout capabilities on AOS, the RESTART tasks suspend themselves until the MSFIT programs in question become active. If the transmission of the MSFIT restart command were sent in the main body of the MCAIO process, the entire process would be suspended. Since the network has more than one RDOS link, MCAIO must remain executable for all active links in the network. By having MCAIO initiate several RESTART tasks concurrently, the main task can remain executable even if the RESTART tasks become suspended.

After starting the RESTART tasks, the main task in MCAIO initiates the RDMCA task. The RDMCA task begins concurrent execution with MCAIO's main task. Finally, the main task in MCAIO initiates an IPC receive request. Since the primary function of the main task in MCAIO is to obtain MSFIT buffers from the AOS environment, the main task reads IPC messages that have been directed to it. After reading an IPC message, the main task sends the message across the MCA
output channel to the appropriate RDOS machine. This loop of reading the IPC mailbox and writing the buffer to the MCA output channel is continued by MCAIO's main task until the AOS information system is stopped (See Figure Three).

When the RDMCA task becomes active, the process immediately initiates an MCA read request. After receiving a buffer from the MCA, RDMCA first determines whether the buffer contains MCA status information or network data. The first word of the buffer is used to determine the buffer contents. If the value contained in this first word is either a -1, -2, or -3, RDMCA uses the buffer to determine MCA status. If an MCA status value exists in the first word, RDMCA uses the second word of the buffer to determine which RDOS machine has changed its MCA operating status. After finding the RDOS machine identification, RDMCA changes the MCA status flags located in a common region accessible by both RDMCA and the main task in MCAIO.

If the first word of the buffer is not a -1, -2, or -3, RDMCA assumes that the value contained in this word is the IPC address to be used to transmit the MCA buffer received by RDMCA to the AOS environment. This address is generated by the main task in MCAIO. When MCAIO receives an IPC message, the task also has access to the IPC address of the AOS process that sent the IPC message. This address is used if an IPC message is to be returned to the process sending
THE MCAIO PROCESS

FIGURE 3
MCAIO TASK FLOW

OPEN MCA I/O CHANNELS

INITIATE "RESTART" TASKS

INITIATE "RD MCA" TASK

READ IPC MAILBOX

DETERMINE MCA STATUS FOR REQUESTED RDOS MACHINE

MCA AVAILABLE

NO

SEND MCA STATUS ACROSS IPC TO MCABF

YES

SEND BUFFER TO MSFIT

THE MCAIO PROCESS

FIGURE 3
(CONTINUED)
RDMCA TASK FLOW

READ MCA

MCA STATUS CHANGE

YES

CHANGE INTERNAL MCA STATUS FLAGS

SEND INTERTASK MESSAGE TO ANY "NOTIFY" TASKS

NO

SEND BUFFER ACROSS IPC TO MCABF

THE MCAIO PROCESS
FIGURE 3
(CONTINUED)
the initial IPC message. The main task stores this address in the first word of the MSFIT buffer before sending the buffer across the MCA. Since MSFIT does not access this first word during its normal RDOS disk I/O functions, any response generated by MSFIT leaves the first word of the buffer intact for transmission back to the AOS machine. When the response buffer is transmitted back to AOS, the RDMCA task accepts the buffer and can then read the first word and use the address to determine the calling AOS process. This eliminates any handshaking or intertask communications between the two concurrent tasks within the MCAIO process.

The only information needed by the main task of the MCAIO process is the MCA status. This status information is received by the RDMCA task. However, the main task in MCAIO must know the MCA status before sending a buffer across the MCA. If such a buffer is sent to an inactive RDOS link, the main task in MCAIO becomes suspended until MSFIT is executed on that RDOS machine. To circumvent this, the main task in MCAIO examines a common region where the MCA status flags are stored. If the RDOS machine is determined to be inactive, the main task of MCAIO sends a message back to the AOS process initially sending the IPC message. The returned message indicates that the MCA link requested is inactive. When the MSFIT machine becomes available, the MCA status flag is changed in the common memory region within MCAIO to
show an active status.

The RESTART tasks started by the main task in MCAIO have the function of sending the restart command to the MSFIT program located on the machine appointed for that task. After a successful transmission across the MCA, the RESTART task terminates. If MSFIT is not executing on the RDOS side, the RESTART task suspends execution until the MSFIT program becomes active. When MSFIT does become active, the RESTART task can satisfy the MCA I/O request and RESTART can terminate. Note that MSFIT does not require an automatic restart on initial execution. If such a restart request is outstanding, this request is the first command serviced by MSFIT. No other associated network commands can be overridden. (For example, a command to open an RDOS file channel cannot be received before the MSFIT automatic restart has finished closing all open file channels.) Should the MSFIT program be terminated while the AOS information system is active, the normal -2 and -3 status flags are sent across the MCA and received by RDMCA. When MSFIT terminates, no further communications can occur across that particular MCA channel.

The final aspect of MCAIO is its ability to inform other processes in the AOS environment of the establishment of MCA links. If the AOS process has learned that the MCA link is unavailable to the machine desired, the AOS process
has the opportunity to ask MCAIO to inform it when the MCA link is active once again. To do this, a buffer is sent to MCAIO from the AOS process with the command word in the buffer set to -99. When MCAIO receives the buffer, it checks the MSPIT command word for this possibility. If a -99 is found, the main task in MCAIO starts another concurrent task called NOTIFY. The purpose of the NOTIFY task is to send an IPC message back to the AOS process requesting notification. The NOTIFY task does not have the responsibility of determining this information. Rather than having these NOTIFY tasks remain active, each NOTIFY task requests an intertask message from RDMCA signaling an MCA status change. To be more specific, only the NOTIFY tasks associated with a particular RDOS machine are informed of the status of the MCA link. Therefore, RDMCA has the additional responsibility of restarting the NOTIFY tasks during these MCA status changes. After the NOTIFY task is restarted, it sends an IPC message back to the AOS process requesting the information. It is the responsibility of the calling AOS process to maintain a vigil task waiting for the notification from MCAIO's NOTIFY task. After NOTIFY has sent the IPC message, the task terminates.

In summary, MCAIO has two primary concurrent tasks, the main task and RDMCA. The main task deals with communications from the AOS environment to the MCA, and RDMCA handles...
communications from the MCA to the AOS processes. MCAIO must determine the status of the MCA before doing its transmission. This is done in the main task but the status flags are changed by RDMCA. MCAIO can be requested to notify an AOS process of MCA availability with the NOTIFY task. Finally, the MCAIO process initiates a MSFIT restart request on initial execution of MCAIO. This allows MCAIO to determine that MSFIT processes are active and informs MSFIT that any current RDOS file channels should be closed so that future network requests can clean up these incomplete RDOS disk files.
3.3 THE MCABF PROCESS

The AOS process MCABF is responsible for the creation of the MSFIT command buffers filled with data obtained from within the AOS system. By generating the command buffers to handle the RDOS disk I/O functions, MCABF emulates a resident RDOS program. The buffers created by MCABF are transmitted to MCAIO by using the IPC facility within AOS. When MCAIO receives these buffers, it transmits the buffers across the MCA to MSFIT. All necessary MCA I/O is handled by MCAIO enabling MCABF to deal only with the generation of the MSFIT buffers.

When the AOS information system is started, the main task of the MCABF process becomes active. This task creates a local IPC mailbox connected to the AOS information system. It is the responsibility of the information system to start the MCABF process at the proper time. The main task initiates an IPC receive request after creating the mailbox. At this point the MCABF process suspends execution. When the information system starts the main task in MCABF, the task starts another concurrent task called MCATSK. The main task returns to another IPC receive request while MCATSK handles the recent activation request.

When MCATSK becomes active, it is allocated a shared memory resident buffer accessible to all processes in the
information system. This buffer uses a segment of the shared memory starting at a location defined by the synchronization routines within the information system. MCATSK uses various information contained within this buffer. First, the name of the temporary disk file generated by the network processes is located within the buffer. Contained within this file are the names of all files being sent to a specific RDOS machine during this network request. Each file name contained within the temporary disk file, called a PKT file, has an associated command word. This command word is used by MCATSK to determine what type of file is being sent across the network. Other information contained within the information system buffer received by MCATSK includes the number of files contained in the PKT file and the RDOS machine destination for all files contained within the PKT file.

After MCATSK determines the name of the PKT file, the task opens the file for reading. MCATSK goes into a loop that reads a filename in the PKT file, does the necessary MSFIT buffer generation and transmission, and goes back to the beginning of the loop. The MCATSK task continues in this loop until all entries in the PKT file have been read. The loop consists of calling the proper subroutine in MCATSK to generate the MSFIT buffers for the particular file type read from the PKT file. There are two types of files that can be
transmitted across the network. The first file type is a text file and the second is a binary file. The text file is a line-oriented file where all bytes contained within the file are printable ASCII characters. The binary file contains bytes that represent octal integer values and do not have any corresponding printable ASCII text. These binary files are used by the RDOS manufacturing machines exclusively and are designed for mass storage of data. The text files are designed for easy modifications using a standard text editor. The two subroutines associated with these two file types are SDASC for text files and SDRND for binary files. MCATSK calls subroutine SDASC for each text file found in the PKT file. Likewise, SDRND is called for each binary file in the PKT file (See Figure Four).

The structure of these two subroutines are the same. Both subroutines generate the same sequence of MSFIT command buffers with only minor differences in the open and write RDOS file commands. The following paragraphs include a detailed description of the SDASC and SDRND functions.

When MCATSK calls SDASC, the subroutine is not being executed as a task. The process execution has been passed from MCATSK to SDASC with no concurrent execution of routines. When SDASC starts, it first determines if the text file to be sent exists on the AOS computer. If the text file does not exist, there is no need to continue accessing the
THE MCABF PROCESS

FIGURE 4
MCATSK TASK FLOW

OPEN "PKT" FILE

FILES LEFT TO SEND?

YES

GET NEXT FILE NAME

TEXT FILE TYPE?

CALL "SDASC"

CALL "SORND"

MCA WAS AVAILABLE?

YES

NO

NOTIFY OPERATOR

DELETE "PKT" FILE

RELEASE AOS SYSTEM RESOURCES

TERMINATE TASK

NOTIFY OPERATOR

OPERATOR CHOOSES TO WAIT FOR MCA

WAIT FOR NOTIFICATION FROM MCAIO

THE MCABF PROCESS

FIGURE 4

(CONTINUED)
THE MCABF PROCESS
FIGURE 4
(CONTINUED)
network. If the text file does exist, SDASC will open a file channel to it and continue executing. The next step is to determine whether the text file being sent does not already exist on the RDOS machine. SDASC creates a MSFIT buffer containing a command to do an RDOS file status call. SDASC sends the created buffer to the IPC mailbox for MCAIO. The SDASC routine then waits for a response from MCAIO. When a response is received, SDASC knows two pieces of information. First, SDASC knows whether the link to the RDOS machine in question is available. Second, if the RDOS machine is available for network use, SDASC knows if the file being sent is already there. The results of the RDOS file status call are returned through MCAIO back to the SDASC routine. If the file already exists on the RDOS side, the SDASC routine is finished. If the file does not exist, SDASC continues to the next step. At this point, SDASC creates a MSFIT buffer requesting that MSFIT create a file and open a file channel on the RDOS machine. The filename used is not the name of the file being sent but rather a singular temporary file name generated by the task identification of the MCATSK calling SDASC. SDASC sends this MSFIT buffer to MCAIO and waits for MCAIO to return the results. SDASC becomes active once again when the buffer returns with the channel number for the RDOS file. All further write requests on the RDOS disk can use this channel number rather than the file name.
SDASC has opened files on both of the AOS and RDOS systems. The reading of the AOS file and writing to the RDOS file is the next step in transmission. Up to this point, all MSFIT buffers created by SDASC required an additional response buffer. SDASC would send a MSFIT command buffer and then wait for the buffer to be returned so that SDASC could determine if the MSFIT command has been executed successfully. This handshaking requires too much interaction across the MCA link for data transmission. To reduce the handshaking and maintain file transmission reliability, SDASC sends MSFIT RDOS disk I/O buffers that require no handshaking. The MSFIT process returns a buffer only if an RDOS disk file error occurs. To catch the possible error on the RDOS side, the SDASC subroutine starts a concurrent task called MCAREC. When this task becomes active, SDASC reads lines from the AOS file and places the text in a buffer to be sent to MCAIO. Each buffer contains one line of the text file. The buffers are sent sequentially to MCAIO which transmits the buffers to MSFIT. Before sending each buffer to MCAIO, SDASC determines whether any RDOS write errors occurred within MSFIT. If no errors resulted during the transmission of all buffers, SDASC kills the MCAREC task, closes the AOS file channel, and sends another buffer to the RDOS machine to close the RDOS channel. Now that the file has been transmitted, the final step is the renaming of the RDOS file from its
temporary name to the real name of the AOS file. After a successful renaming, SDASC returns execution back to MCATSK.

When SDASC starts MCAREC, the MCAREC task performs one function. MCAREC generates an IPC receive request waiting for any error messages returned through MCAIO from the MSFIT process. Under normal file transmission, no messages are received and the MCAREC task is terminated by SDASC. However, if the MSFIT process encounters an RDOS disk write error, MSFIT sends the buffer back to MCAIO which in turn sends the buffer to the calling AOS process. The outstanding IPC receive request generated by MCAREC allows MCAREC instead of SDASC to receive the buffer. If MCAREC should receive such a buffer, MCAREC writes the error number generated on the RDOS side into a common memory location that both SDASC and MCAREC can access. The SDASC routine checks this memory location before each buffer transmission. SDASC stops sending buffers if MCAREC has received an error message. After MCAREC writes the error number for SDASC, the MCAREC task terminates. If an RDOS error occurs, SDASC must close the file channels on the AOS and RDOS machines before reporting the error to the network user and returning control to MCATSK.

The subroutine SDRND transmits binary files similar to SDASC. Differences include the mode of opening the file used by MSFIT to access the file on the RDOS side, and the type...
of write request sent to MSFIT. Binary files are configured as 256 word blocks and do not have any delimiting characters. These blocks are read sequentially on the AOS computer and placed in buffers bound for MSFIT. When the last block is read on the AOS side, an end of file error is encountered telling SDRND to jump out of the read/write loop. SDRND does not use handshaking for the disk writing in MSFIT. Instead, SDRND starts an MCAREC task to monitor transmission errors.

If an MCA shutdown request is entered during file transmission, some additional IPC communications are necessary. This is discussed as part of the network dynamics in Section 4.0.

For each file name in the PKT file, MCATSK calls either SDASC or SDRND. When the end of the PKT file is reached, the file channel associated with the PKT file is closed, and the PKT file is deleted. These PKT files are internal files used by the network and are not intended to be viewed as a record of transmitted files. During all file transmissions within SDASC and SDRND, results are displayed on the network user's terminal. Also, the results displayed on the terminal are spooled for output both to a line printer and a disk file for future reference. During normal file transmission, no operator interaction is necessary if the destination machine has MSFIT running.

. 46 .
In summary, the MCABF process generates the buffers necessary to control the actions of a particular MSFIT process. These buffers are generated by two possible subroutines located within a concurrent task called MCATSK. The two subroutines differ only in the type of file being transmitted to the RDOS machine. The subroutine used to send ASCII text files is SDASC, and the subroutine used to transmit binary files in block mode is called SDRND. Both routines require no handshaking after initial conditions are set because another task called MCAREC is initiated. MCAREC monitors the transmission and becomes active when an RDOS disk write error occurs within MSFIT. The files to be sent are listed in a PKT file and this file is created by another information system process. The PKT file is deleted on completion of the network request.
3.4 THE SEND PROCESS

The SEND process is the AOS process that is linked to other data bases contained within the information system. This process uses the data stored within the information system to generate the list of file names used by MCABF to send a package of files to an RDOS machine. The SEND process is activated by the information system when an operator has requested a network service.

When the SEND process is initiated during the AOS information system startup, the main task of the process begins execution. This task creates a local IPC mailbox that is connected to the information system. After creating the mailbox, the main task generates an IPC receive request and suspends execution until the AOS information system activates SEND. When SEND's main task receives the IPC message, a shared memory buffer common to the AOS information system and SEND is made available for SEND's use. The buffer contains the operator command originally entered on an AOS computer terminal. The main task of SEND calls a subroutine called SENDMSG to parse the operator command in the buffer and return several flags back to SEND's main task detailing the structure of the files being transmitted across the network. The manufacturing facility that uses the AOS information system needs several varieties of file combinations including a complete package of...
information necessary to produce the product. These file packages generally contain one text file and one or more binary files. The name of the text file is contained within the text of the operator's SEND command. The binary file names are found by using the text file name as a key to an AOS data base containing the manufacturing information for the product associated with the text file. This manufacturing information includes a list of all binary files needed to manufacture the product. When the operator requests that an entire product package be transmitted to an RDOS machine, the AOS data base is accessed and the text file name plus all binary file names are gathered. These file names are then placed into the PKT disk file. As mentioned in the previous section, the PKT disk files are used by MCABF. When the SEND process writes the file names into the PKT disk file, it also writes a flag showing the file type for each file name entered into the PKT file.

The above procedure does not occur as part of the main task of SEND. After the command has been parsed and has been determined to be syntactically correct, the subroutine SENDMSG returns process execution back to the main task with no error flags set. The main task initiates a new task called SENDTSK at this point. After starting SENDTSK, the main task loops back to the coding that requests another IPC message. The concurrent SENDTSK takes the command line
entered by the operator and analyzes it. The necessary data bases are accessed and one PKT file is created for the entire network request (See Figure Five). The PKT file can contain up to 512 file names. PKT files containing less than 512 file names only occupy the disk space necessary to store the given number of files. An end of file mark is placed after the record containing the last file name of the package.

After generating the PKT file, SENDTSK closes its file channel and loads the PKT file name into the shared memory buffer common to the information system. SENDTSK sets up the protocol necessary to start the MCABF process. After sending the information system protocols to MCABF, SENDTSK terminates.

Since the information system receives network requests asynchronously, the SEND process must be able to accommodate more than one request concurrently. Each operator command is allocated a singular information system buffer and the SEND process starts a unique SENDTSK task to service the operator's network command. Each SENDTSK can activate MCABF individually since the MCABF process also has the multitasking capability and a unique information system buffer structure to handle individual network requests. The information system buffers are returned automatically to a "least recently used" chain of free buffers when the . 50 .
THE SEND PROCESS
FIGURE 5
SENDTSK TASK FLOW

1. Obtain Operator Command
2. Create a unique "PKT" file
3. Generate filenames associated with the operator command and place them in the "PKT" file
4. Store "PKT" filename in AOS shared memory
5. Signal MCABF process
6. Release AOS system resources
7. Terminate task

The SEND process Figure 5 (continued)
information system is informed that the SEND or MCABF processes no longer require their use.

In summary, SEND is the information system process responsible for accessing data stored within the information system and generating a list of files associated with a network request. This list is written into a PKT file for MCABF's use. The information system starts the SEND process when an operator has entered a network request. After the concurrent SENDTSK task generates the necessary PKT file, SENDTSK also conducts the proper information system protocol to start MCABF and its tasks. The buffer used by the information system to pass the operator's command is also used by SENDTSK to store the PKT file's name. Control of the buffer is passed to MCABF on activation of MCABF. SEND can service more than one request simultaneously, and therefore, MCA data transmissions can be concurrent.
3.5 THE DISPATCH PROCESS

The final process necessary for network communications across the MCA is the DISPATCH process. This process is the parent process of the information system. When the information system is brought into operation, the DISPATCH process begins execution. DISPATCH initializes the shared memory buffers and other disk files used by the other processes of the information system. After DISPATCH does this housekeeping, the remaining AOS processes associated with the information system are started. Currently, there are 30 processes for the entire manufacturing facility, including the four network processes. These four network processes have been discussed in the previous sections. Other aspects of the DISPATCH process are not discussed in this thesis except for the information relating to the AOS-RDOS network.

After starting all information system processes, DISPATCH initiates terminal read requests from every terminal attached to the information system on the AOS computer. These terminals are now available for entry of operator commands. The DISPATCH process suspends execution until an operator types a SEND command (or any other valid system command). DISPATCH takes the operator's command and compares the first word of the command with words listed in a command table. The command table contains each valid
information system command along with the identification of the information system process associated with that particular command. Therefore, if the command SEND is entered, DISPATCH sends an activation signal to the SEND process.

Before signaling the information system process found in the command table, the DISPATCH process must allocate a shared memory buffer. As stated earlier, this buffer is used to pass information conveniently between the various AOS processes. For the SEND process, DISPATCH loads the operator's command line into the shared memory buffer. This buffer is passed to the SEND process when DISPATCH signals SEND. If another command is entered and DISPATCH determines that SEND is again requested, the DISPATCH process signals the SEND process again after allocating an additional buffer for SEND. The SEND process can handle all these activations because it is multitasked. The main SEND task deals with these signals sequentially, but schedules the SENDTSK tasks to execute concurrently.

Since all commands within DISPATCH are handled one at a time, no multitasking of this function is necessary. DISPATCH can handle all terminals within the information system by using a single task process (See Figure Six).

In summary, the DISPATCH process maintains the
THE DISPATCH PROCESS
FIGURE 6
DISPATCH PROCESS FLOW

ACTIVATE ALL INFORMATION SYSTEM PROCESSES

INITIALIZE AOS SHARED MEMORY AREAS

INITIATE TERMINAL READ REQUESTS

WAIT FOR INPUT

COMPARE INPUT TO COMMAND TABLE

YES

INPUT VALID?

ALLOCATE AOS SHARED RESOURCES

SIGNAL APPROPRIATE AOS PROCESS

NO

REPORT ERROR

THE DISPATCH PROCESS

FIGURE 6

(CONTINUED)
activation and termination of the AOS information system. It
does all necessary file and buffer housekeeping for the
system. All information system processes are sent IPC
activation messages through DISPATCH whenever DISPATCH
receives an operator command with an associated command
table entry and process identification. The SEND process is
signaled by DISPATCH after an operator enters the SEND
command. Before signaling SEND, DISPATCH allocates a shared
memory buffer and places the operator's command within this
buffer. Finally, DISPATCH sends the IPC activation message
to SEND and returns to the beginning of the loop waiting for
the next terminal input.
4.0 THE DYNAMICS OF THE NETWORK

Section three of the thesis described each network process in detail. Since the processes must be synchronized for successful file transmission, some discussion is appropriate regarding resource management and process synchronization.

The network described in this thesis has been in use at a manufacturing location since July, 1980. This location has one AOS computer connected to three RDOS computers. All the RDOS computers are slaves to the AOS computer. The following discussion describes a scenario where one or more RDOS machines are not communicating to the network.

Because of the MSFIT protocols, the AOS information system is always aware of which RDOS computers are active. After sending a buffer to MCAIO, the MCABF process may receive an immediate response buffer when the RDOS machine associated with the file transmission has been determined to be inactive. If MCABF should find the RDOS machine unavailable, several functions must be performed.

In the existing network, the AOS information system processes give a network operator three options when file transmission to an RDOS machine is impossible. The first option allows the operator to wait for the MSFIT process to be executed on the RDOS machine. Because of the local nature
of the network, the operator may have access to the RDOS machine concerned in many instances. In this circumstance, the operator can physically walk to the RDOS machine and execute the MSFIT program. When MSFIT is started, the status buffer with the -l flag is sent across the MCA. The RDMCA task contained within the MCAIO process receives the status buffer and sends an intertask message to all NOTIFY tasks under MCAIO. The NOTIFY tasks send an IPC message to each process waiting for that particular RDOS machine. The processes waiting for notification are MCATS task in MCABF. If the operator decides to wait for MSFIT execution, MCATS sends a buffer containing the -99 MSFIT command buffer to MCAIO. The MCATS task waits for a response from the NOTIFY task in MCAIO. When NOTIFY signals MCATS that the RDOS machine is available, MCATS starts sending the MSFIT buffers needed to transmit the files.

The first command sent to MCAIO from MCATS after the initial restart of MSFIT is always the file status or open RDOS disk file command because the MSFIT process never allows termination before closing all its open RDOS disk file channels. Therefore, no single file transmission can continue over later executions of MSFIT. Also, the MCABF process has an inherent means of insuring that NOTIFY signals sent to MCATS are valid. Since the first attempts at transmission are RDOS file status or open commands rather
than data transfers, channel save pointers (the current AOS file input location pointers) are not affected if another response buffer is returned to MCATSK from MCAIO flagging that the MCA is not available again. The software is designed to continue sending the file status or file open MSFIT command until a successful response buffer is received from an active MSFIT. Once a successful file channel open has been done on the RDOS computer, MSFIT cannot terminate until the MCATSK on the AOS computer closes the channel. Since there is no chance that MSFIT can terminate while receiving RDOS disk write command buffers, it is not necessary for MCATSK to be concerned with receiving asynchronous MCA status buffers from MCAIO during AOS file reading and RDOS file writing.

If the RDOS machine crashes, a serious deadlock can occur. The MSFIT program does not get a chance to send any status buffers across the MCA to MCAIO. This implies that MCAIO is not aware of the inactive RDOS status and, therefore, the status flag in MCAIO is kept at -1 (or flagged as "active"). Unfortunately, any future accesses to this RDOS machine are permitted. The next time MCAIO is requested to transmit to this MCA channel, MCAIO encounters an MCA write command that cannot be satisfied. At this point, the main task MCAIO is suspended and all communications are stopped. Fortunately, this occurs
infrequently since the RDOS machines are reliable. The situation is mentioned because current operating system limitations within AOS allow such a problem to result. This is a case where MCA write timeouts would enable MCAIO to ascertain the MSFIT process status and update the status flag to -3 (or "inactive") until further status buffers have been received by the MSFIT in question.

The discussion now returns to the operator options available when file transmission is requested to an inactive MSFIT process. If the operator cannot execute the MSFIT program on the RDOS machine, he may choose the second option. The information system has access to several magnetic tape units on the AOS computer. If the user chooses, he may send all files associated with his network request to a magnetic tape instead of transmitting the files across the MCA. If this option is selected, the MCATSK that discovered the inactive MSFIT must direct process execution to another information system process. The magnetic tape capability does not exist as part of the MCABF process. To accomplish the tape output, MCATSK must request that the information system start the MTAOUT process. The MTAOUT process receives the shared memory buffer originally allocated to MCABF plus the name of the PKT file that MCABF was using. If file transmission is interrupted in the middle of a package, only the remaining files listed in the PKT
file are sent to magnetic tape. After sending all remaining files to tape, MTAOUT removes the PKT file from the AOS disk and the network request is considered to be satisfied. After dumping the files to tape, it is the responsibility of the operator to transport the tape to the appropriate RDOS machine and have the contents of the tape loaded onto that RDOS machine. An independent utility program not associated with the AOS-RDOS network is used to enter the files onto the RDOS disk from the magnetic tape.

The magnetic tape option is also used as the backup medium if the MCA hardware fails. All AOS-RDOS communications can be done through magnetic tapes should the need arise. Also, these magnetic tapes can be used to transport information between separate AOS information systems. Currently, dedicated high speed links do not exist between the two manufacturing locations using the information system. Therefore, the magnetic tape medium is the only available means of transporting large volumes of information between these two locations.

The final option permitted by MCATSK during file transmission failure is the ability to cancel the network request altogether. When this option is selected, MCATSK returns the shared memory buffer back to the information system's "least recently used" free buffer chain, the PKT file associated with MCATSK is deleted, and the associated...
MCATSK terminates.

Whether the network link requested is available or not, the system always uses IPC flags rather than polling loops to determine the status of any portion of the network. No additional CPU time is wasted when some nodes of the network are unavailable. By multitasking the routines containing semaphores, the system never reaches a state where communications are deadlocked. Also, by associating a restart request to each node on the network, the system allows communications to proceed on those nodes that are available during restart.

The design of the information system restricts any AOS-RDOS network use except when the information system is available. The network is started when DISPATCH is executing and stops when DISPATCH is terminated. All network processes on the AOS computer are started by the DISPATCH process. The RDOS MSFIT processes do not terminate when DISPATCH terminates. These processes can be stopped only by the RDOS operators on their local RDOS manufacturing machine. Because the MSFIT processes have restart capabilities, they may continue execution indefinitely through several activations of the information system.
5.0 CONCLUSION

The purpose of this thesis has been to describe the functionality of the computer network that uses a master-slave configuration. The primary points stressed by the discussion include the importance of allowing communications to be unaffected by any singular nodes on the network. Dividing the coding into separate processes allows more of the network to be resident in computer memory concurrently. By using semaphores instead of wait loops, AOS CPU time wasted during incomplete transmissions is kept at a minimum. The reduction of handshaking is done by scheduling a sentry task during data transmission. This concurrent task insures that files are transmitted without error. Finally, concurrent process synchronization has been demonstrated to be an effective means of overcoming many possible limitations inherent in minicomputer operating systems.
REFERENCES


(2) Data General Corporation, Westboro, Massachusetts. The following publications were used as references and were copyrighted 1976 to 1981:

  69-016 Introduction to the Advanced Operating System.
  69-018 Learning to Use Your Advanced Operating System.
  69-020 AOS Software Documentation Guide.
  93-193 AOS System Manager's Guide.
  93-217 How to Load and Generate Your AOS System.
  93-154 FORTRAN 5 Programmer's Guide.

VITA

Randy R. Boldosser, son of Randall K. and Regina S. Boldosser, was born 5 July 1957 in Pottstown, Pennsylvania. He was salutatorian of his graduating class from Pottsgrove High School in 1975. He attended Lehigh University from 1975 to 1979 and graduated with high honors with a Bachelor's Degree in Electrical Engineering. He also minored in Economics as an undergraduate. Since 1981, he has been pursuing his Master's Degree in Computer and Information Sciences at Lehigh University. He was a member of the Phi Eta Sigma Honor Society, a member of I.E.E.E., a member of the Tau Beta Pi honor society, and the treasurer of the Eta Kappa Nu honor society. He was also the recipient of a Firestone Tire and Rubber Company Scholarship.

Randy Boldosser has been employed by Western Electric Company of Allentown, Pennsylvania, since June, 1979, as a software systems engineer. He has been involved in the development of a production facility control and information system. Also, he has designed a computer network associated with this information system. Since 1981, he has lived in Allentown with his wife, Nancy Shilay Boldosser.