DISTRIBUTED COMPUTER OPERATING SYSTEM
Programming Guide

Version 3 April 1974

Lawrence A. Rowe
William J. Earl
Allan D. Foodym
Frank R. Heinrich

Technical Report #46 - April 1974

This work was supported by the National Science Foundation under Grant GJ-1045.

ACKNOWLEDGEMENTS

This programming guide describes how to write programs for the Distributed Computer Operating System. Many people have contributed to the Distributed Computer Project and the growth of the operating system.

Dave Farber and Julian Feldman, co-principal investigators, provided guidance and an environment in which to develop the system.

Don Loomis designed the communication system interfaces and maintained the hardware used by the project. He also developed the "Quickie Operating System," without which the software development would have been much more painful.

Greg Hopwood developed and maintained the higher level system implementation language (MOLSUE). As manager of the Varian 620/i lab, he has sailed that always difficult course of "satisfying all of the people all of the time."

Bill Crosby modified and maintained much of the file system and has been an invaluable "systems hack," working on QED, QOS-3, PIP, FAILSAFE, and the start-up process for DCOS version 3.

Alan Bell, Ken Larson, Steve Howell and Ed Schwartz have developed many of the components in QOS and DCOS. Alan designed the initial file system in the 620/i. Ken contributed to the early design of the DCOS and implemented an early version of the input/output process. Steve and Ed have worked on a variety of "little" projects which have been useful in a "big" way.

Elaine Gord and Marsha Hopwood have provided a sympathetic ear, and insightful comments on the many occasions when all seemed doomed.

TABLE OF CONTENTS

List of Figures

1. INTRODUCTION

The Distributed Computer Operating System (DCOS) is a multiprogramming operating system distributed over several processors linked together in a network. The particular way of connecting up the machines is transparent to a person writing a program. Each program executing under DCOS is considered to be a process. Processes communicate by sending and receiving messages. All resources are controlled by processes; thus, programs request resource services by sending messages to the processes providing the resource involved.

This programming guide describes how to use the system. It reflects version 3 of the DCOS. The reader is assumed to be familar with the language for the particular machine for which he will be writing programs (presently MOLSUE [1] or LAP [2] for the Lockheed SUE [3]).

2. USING THIS MANUAL

This reference is divided into three main sections: system environment, services and processes, and using the DCS. The section on the system environment describes the details of how a program, or process, interacts with the system. The section on services and processes describes the different processes available on the system. It also describes the command process with which a person sitting at a terminal interacts. The last section, using the DCS, discusses how to go about writing a program to run on the system, and gives an example.

Readers unfamiliar with the system are encouraged to skim through the latter two sections discussed above and to log onto the system and run some processes before digging into the details of the system environment.

3. SYSTEM ENVIRONMENT

This section describes the system environment in which a process executes and the system functions available. Section 3.1, a reprint of an article which was presented at COMPCON 1973 [4], presents a general overview and introduction to DCS. Section 3.2 describes the communication protocol, process names and context blocks. Section 3.3 defines the functions provided by the system, including their calling sequences. Section 3.4 describes the input/output facilities provided. Lastly, section 3.5 describes how a process can initiate and terminate other processes.

3.1 Overview of DCS

Seventh Annual IEEE Computer Society International Conference
February 27028, March 1, 1973

THE DISTRIBUTED COMPUTING SYSTEM

David J. Farber, Julian Feldman, Frank R. Heinrich,
Marsha D. Hopwood, Kenneth C. Larson, Donald C. Loomis, and Lawrence A. Rowe
Department of Information and Computer Science
University of California
Irvine, California 92664

ABSTRACT

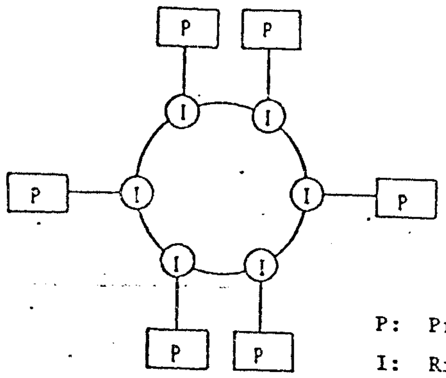
The Distributed Computing System* is an information utility designed to provide reliable, fail-soft service at relatively low cost to a large class of users with modest requirements. The salient feature of this system is the distribution of hardware, software, and system control. Hardware distribution is achieved using a network architecture. Software and control distribution is facilitated through the use of communication by name, rather than address, among the network components. The combined effect of distribution, which provides both redundancy and isolation, and of controlled access, makes total failure of the computing system unlikely. Reliability is achieved by minimizing the probability of total failure, using isolation to keep local failures from spreading and causing global failure, and using redundancy to negate the effects of local failures.

INTRODUCTION

The Distributed Computing System (DCS) is a local network designed to provide reliable, fail-soft service at relatively low cost. The distributed organization of this system incorporates redundancy and isolation in both the hardware and software. The implementation of the DCS organization, with emphasis on how interactions are carried out, is briefly described in this paper.

ORGANIZATION AND COMMUNICATION

The DCS hardware system is a collection of computing system components connected to a digital communication ring by ring interfaces. The communication ring serves as a unidirectional information path and the ring interfaces assist in information routing. Figure 1 shows a DCS configuration with six processors.



P: Processor
I: Ring Interface

Figure 1: A DCS with Six Processors.

The DCS software system is process-oriented, that is, all activities are carried out by processes. Processes interact by sending and receiving messages. Messages are addressed to processes by name, rather than by physical hardware address. A message from one process addressed to another process is placed onto the ring. As the message arrives at each ring interface, the interface compares the destination process name with its list of all processes active in the attached processor. If the destination process name is present, the interface attempts to copy the message into the processor memory. Whether the destination process name is present or not, the interface allows the message to travel on to the next interface on the ring. The message continues around the ring until it arrives at the interface for the processor in which the sending process resides. This interface removes the message from the ring.

Two status bit, MSB1 and MSB2, are set by each ring interface as a message passes. MSB1.is set if the ring interface matches the destination process name to a name in its list of active processes and attempts to copy the message into its attached processor but is unable to do so. MSB2 is set if the message is successfully copied. Figure 2 describes the possible values and meanings associated with the message status bits. Since the ring

*This work has been supported by the National Science Foundation under Grant GJ - 1045.

interface which places the message on the ring is the one which removes the message, the sending process can tell if the message was received by examining the message status bits.

A process normally sends a message to another specific process. However, there are occasions when a process wishes to send a message to a set of processes. This is called message broadcasting. An example of message broadcasting is given below in the discussion of resource allocation.

MSB1	MSB2	Meaning
0	0	The destination process name was not matched by any ring interface.
0	1	The message has been matched and copied by at least one ring interface.
1	0	The destination process name was matched by at least one ring interface, but not copied by any of them.
1	1	The destination process name was matched by at least two ring interfaces. At least one ring interface copied the message and at least one did not.

Figure 2: Possible Values of Message Status Bits.

RESIDENT SYSTEM SERVICES

Each processor on the ring has a resident software system called the nucleus. The nucleus schedules processes running in the processor and provides message transmission facilities.

The nucleus schedules user processes in a first-come, first-serve round-robin order. Each processor has a time-out mechanism which prevents a user process from holding the processor indefinitely. Within each nucleus there are some system processes which have priority over all user processes. These system processes, such as the output message routine described below, are scheduled when their services are needed.

When a process wants to send a message, it issues a system call with two arguments: the destination process name and the message. A routine for servicing system calls in the nucleus copies the message and other message protocol information into a system buffer. This routine is also responsible for placing the originating process name into the message.

The formatted message is placed on a queue for the output message routine. The output message routine takes messages from the queue and places them on the ring. When the message has been copied into the receiving processor, that processor is interrupted signaling that a message has been received. The interrupt routine places the message on the general input message queue and reinitializes the ring interface to receive the next message.

The message is moved from the general input message queue to the input message queue for the particular process to which the message is addressed. The next time the addressed process is scheduled, a special message receiving routine copies the message into a buffer within the process data space. It is the responsibility of the process to check its message buffer periodically for an incoming message.

Because the majority of activities carried out between processes require sending a message and then waiting for a reply, a process may block itself waiting for a reply. This block is usually requested along with a time limit so that a process is not suspended indefinitely.

Before placing the formatted message on the output message routine queue, the routine for servicing system calls looks to see if the addressed process resides in the same processor. If it does, the message is placed on the general input message queue rather than the queue for the output message routine. In this way, a message is not needlessly transmitted around the ring.

In summary, the nucleus provides facilities for scheduling processes and transmitting and receiving messages. Other system functions, such as resource allocation, device input-output, and file system services[4], are provided by processes executing in the DCS. Because the nucleus is the only software unit bound to a particular machine, these other system services may be executing in any machine in the ring and are accessed from user processes by sending and receiving messages. A process requesting service does not need to know where in the DCS the service process resides, because messages are addressed to processes rather than processors.

PROTECTION

The functions one needs to protect a system, isolation and controlled access[1,2], are explicitly included in the DCS design. In addition to the standard machine-level protection mechanisms, the DCS must also employ network-level protection. Because the DCS is process-oriented and process access is done by sending and receiving messages, protection is achieved by insuring the integrity of messages. In particular, protection is achieved by insuring that the sending process name placed in a message by the resident nucleus is actually the name of the process sending the message. If forgery of the sending process name has been prevented, a receiving process can discriminate in the services it provides depending on the source of the request.

RESOURCE ALLOCATION

Resources in the DCS are treated as processes. Each resource is associated with a process and all use of that resource is through the associated process. Resource allocation can then be achieved through the process management mechanism. The mechanism by which processes are initiated is called request-bid or request for quotation. Each processor in the network has a resource allocator, which does not necessarily reside on that processor, controlling resources, e.g., memory and peripheral devices, connected to that processor. To initiate a process, a user sends a broadcast message to all resource allocators requesting a bid on the services desired. The resource allocators reply with bids to perform the requested service. The requestor receives these bids and selects the best according to its own criteria. A message accepting the bid is then returned to the offering resource allocator, but this does not yet complete the agreement. Bids are not binding and the resource allocator may have accepted other work between the time the allocator sent the bid and the requestor sent the acceptance. Thus the allocator may not be able to satisfy the request, in which case the requestor is notified and must renegotiate a contract. If, on the other hand, the resource allocator can still honor the bid, it returns a message to the requestor acknowledging the contract. At this point the agreement is complete and the requestor can assume that the resource has been assigned to him.

There are three points to notice about this scheme: control is distributed, load balancing is carried out implicitly by the relationships between the different bids, and expansion or contraction of available resources requires little or no programming effort. In this resource allocation scheme, control is distributed throughout the system. No central authority controls the relationship of the bids which determines which processor will provide the service. Load balancing, dynamically distributing the load so that each processor receives some portion of the total work, is being effected every time a new process is created. The processor with the most unused capacity, or best able to provide the requested service, will return a low bid, while processors that are relatively full will bid high. This insures the distribution of work. A change of configuration, thereby altering available resources, is achieved with little programming effort, since all that is required is the addition, deletion, or change of the appropriate resource allocators.

CONCLUSION

The Distributed Computing System is a reliable, fail-soft information utility with a network architecture. Reliability is achieved through redundancy, isolation, and controlled access. Redundancy allows the DCS to continue to function with no loss of capability but some loss of capacity if a component fails. Isolation and controlled access prevent the failure of one component from causing the failure of other components and thus precipitating a total system failure.

Several other papers give greater detail about certain aspects of the DCS, in particular, the communication system[6], the communications protocols[7], the file system[4], and the fail-soft behavior of the software system[5]. For purposes of comparing the DCS to other networks, [3,8] may be helpful.

REFERENCES

1. Brinch Hansen, P. "The Nucleus of a Multiprogramming System." Comm. ACM 13, (April 1970), 238-241, 250.

2. Denning, P. J. "Third Generation Computer Systems." Computing Surveys 3, (Dec. 1971), 175-216.

3. Farber, D. J. "A Survey of Computer Networks." Datamation 18, 4 (April 1972), 36-39.

4. Farber, D. J. and F. R. Heinrich. "The Structure of a Distributed Computer System--The Distributed File System." Proc. International Conference on Computer Communications, (Oct. 1972), 364-370.

5. Farber, D. J., M. D. Hopwood, and L. A. Rowe. "Fail-Soft Behavior of the Distributed Computer System." Technical Report #24, Department of Information and Computer Science, University of California, Irvine, California, (November 1972).

6. Farber, D. J. and K. Larson. "The Structure of a Distributed Computer System--The Communications System." Proc. Symposium on Computer-Communications Networks and Teletraffic, Microwave Research Institute of Polytechnic Institute of Brooklyn, (April 1972).

7. Loomis, D. C. "Ring Communication Protocols." UC Irvine Distributed Computer Project, Memo 46-A, (May 1972).

8. Roberts, L. G. and B. D. Wessler. "Computer Network Development to Achieve Resource Sharing." Proc. AFIPS 1970 SJCC, Vol. 36, AFIPS Press, Montvale, N. J., 543-549.

3.2 System Organization

3.2.1 Message Protocol. The message protocol has the following fields defined:

DPN	OPN	LEN	TEXT

where:

DPN	Destination process name (one word),
OPN	Originating process name (one word),
LEN	Length of text in characters (one word), and
TEXT	Text of the message (byte string).

There is no restriction on message length. Messages may be packeted by the system for transmission, but are reassembled before being passed to the destination process. DCOS handles all aspects of message communication, error recovery and retry. Thus, a process need not concern itself with the problem of message communication, only with passing the correct arguments to the send and receive procedures described in section 3.3.

Two types of interprocess communication are supported by DCS. The first is process to process. The second type is one process to many processes, called broadcast messages. Broadcast messages are used only in special circumstances, mostly by system processes, so that, in general, users need not be concerned with this type of communication.

3.2.2 Process Names. Processes communicate by sending messages back and forth. To send a message to a particular process all that must be known by the sending process is the name of the receiver. Process names are 16 bits long. The format of a name is:

	bits
class	15-12
machine	11-8
sequence	7-0

the class definitions are:

0	not used
1	system processes
2	reserved for resource allocators
3-8	reserved for system classes
9	user processes
A-F	reserved for user classes

Machines are numbered 1 to N and are marked on the front of each machine (N currently is 3). Sequence numbers must be greater than 1. The pair (machine number, sequence number) ensures that process names are unique throughout the DCOS. The machine number does not always indicate on which machine the process is currently executing, only the machine on which the process was first initiated. (The current version of the operating system does not support process migration so the machine number does reflect what machine the process is executing on.) When a process is initiated by the nucleus, the generated hardware name and start address are returned to the requestor. In this way the initiator knows how to send messages to the process it created.

Certain system processes have predefined names:

1X01	Nucleus Process
1X02	I/O Handler Process
1X03	Command Interpreter Process
1X04	Sequence Bit Process
1X05	reserved for Status Checker Process

where X is the machine number. This implies that system process names are generated starting at sequence number 6. The existence of the reserved system process names does not imply that all of these processes exist. Different configurations of the system may include some but not all of the processes. For example, 1203 would be a command interpreter process first intiated on machine 2. However, any given DCS configuration may not have a command process with that name.

3.2.3 Process Context Block. Process dependent information is maintained in a context block (CB). The format of a context block is shown in figure 3-1. The symbol ":CB:" is defined as the address of the context block by the system procedures. Processes may access values in the context block by declaring ":CB:" to be an external and referencing context block entries relative to it.

On the SUE, program execution begins at the address specified in CBSR0 (program location counter) which is defined when the CB is created. A procedures library ("PROC.OB[DCS]") must be linked with each user process. Included in the library are MOLSUE procedures for calling the system functions and a context block for the process. The context block definition establishes location "START" as the process start address. Hence, every procedure must define a variable "START" and declare it as internal. Furthermore, a user process must not declare a start location using the "END" statement in LAP or the "DONE" statement in MOLSUE, since this transfer address is used to pass the address of the context block to the system when the process is loaded.

Items of interest to the programmer in the context block are: CBINI, CBIOH, CBTNO and CBNTF. CBINI is the name of the process which initiated this process. If a terminal was connected to the initiated process when it was created, CBIOH is the name of the I/O Handler for the terminal, and CBTNO is its logical file number (see section 3.4). The logical file number is a byte value. If a terminal was not connected to the process when it was initiated these two fields are zero. CBNTF is the process which will be notified when this process terminates. The terminal, if one is connected to the terminating process, is connected to the process named in CBNTF. Initially CBNTF is set equal to CBINI; however, a process may create arbitrary structures of processes by changing CBNTF.

Figure 3-1 describes all the fields in a context block, and their offsets from ":CB:". A definition file ("CB.DE") defines the MOLSUE offsets for a context block and users are encouraged to append it to the front of their programs. Thus, if the context block definition changes, all that must be done to a program is to recompile/reassemble using the updated definition file.

Figure 3-1
Context Block Format

symbol	type	meaning
CBSR0	word	register 0 save word
CBSR1	word	register 1 save word
CBSR2	word	register 2 save word
CBSR3	word	register 3 save word
CBSR4	word	register 4 save word
CBSR5	word	register 5 save word
CBSR6	word	register 6 save word
CBSR7	word	register 7 save word
CBMBF	word	message buffer address
CBRSB	word	receive sequence bit table address
CBSSB	word	send sequence bit table address
CBIME	word	illegal mdf message error
CBSNM	6 words	symbolic name for process
CBINI	word	name of initializing process
CBIOH	word	name of I/O handler for process´s terminal (0 if no terminal connected
CBTNO	byte	logical file number for terminal
CBNTF	word	process to be notified when this process terminates

3.3 System Functions

There are four system functions provided: send a message (SEND), receive a message (RECV), exit a process (EXIT), and read the time (CLCK). These procedures are defined in the procedures file which must be linked with every process run on DCS. To be used the procedure names must be declared as externals in the user process. The procedures are written using MOLSUE argument and calling conventions. The examples in this section are sample MOLSUE calls. MOLSUE conventions are described in section 5.2.2.

3.3.1 Send a Message. The arguments necessary to send a message are: 1) the destination process name, 2) message length, and 3) the text of the message. Message sending is synchronous, which means only one message may be sent at a time. Upon completion of a send request, a status is returned to indicate whether the message was received at the destination. The status is returned as the value of the send request function call, and has the following possible values:

status	meaning
0	Addressed process is not known in the system.
1	Message was successfully received by all machines on which the addressed process is executing.
2	Addressed process exists but the machine it is executing on was not ready to receive messages.
3	Message was successfully received by at least one machine on which the destination process exists. However, at least one machine which is executing the process was not ready to receive the message. (This happens only with broadcast messages.)

If an unsuccessful transmission condition (0, 2 or 3) is returned, transmission was actually tried a couple of times (approximately 5) and a user can usually assume that the process or processor has failed.

The format for a send message function call is:

```
SEND(DPN,MA,LEN)
```

where:

DPN	destination process name.
MA	message address, if a length is not specified (i.e. LEN=0) then the length of the message is in the first word of the message.
LEN	if 0 then the length is in first word of the message; otherwise, the length is in the argument list. The length is the number of characters in the message not including the length field.

The message to be sent at the location given by MA must begin on a word boundry.

3.3.2 Receive a Message. A process receives a message by requesting to receive one. If a message is available it is copied into the process address space. If a message is not available, the process may wait for the message, i.e. be blocked until a message arrives, or it may continue execution. The receive function has three arguments:

RECV(BUF,BBIT,TIME)

where:

BUF	address of the buffer in which the message is returned, the first word of the buffer is the length of the buffer in bytes, not counting the length word.
BBIT	block bits. If this argument is 0, control is returned to the process whether a message was copied or not. If the argument is 1, the process is blocked until a message is received. User processes may only block on a message or time. System processes may block on any of the conditions in the system status word (SS). The conditions defined are described elsewhere [6].
TIME	number of clock ticks (a clock tick is about 1/60-th of a second) to wait before the process resumes execution. Time wait is an "or" condition with wait for message. This means the process waits until a message is received or for the length of time specified, whichever occurs first. A zero argument means no wait on time.

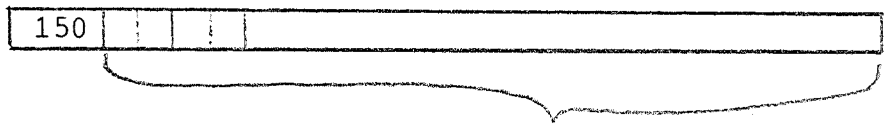
RECV returns 0 if no message was copied, and 1 if a message was copied. A complete message, as shown in section 3.2.1 with DPN, OPN and LEN, is copied into the buffer.

If the buffer address given in the call is 0, then a message is not copied. Thus, a process may wait until a message is available, without it being copied, by doing a receive with BBIT=1 and BUF=0. Or a process may wait for some length of time by calling RECV with BUF=0, BBIT=0 and TIME#0.

Figure 3-2 shows an example of a receive message buffer and a sample call to receive a message.

Figure 3-2
Message Buffer and Receive Message Call

ALPHA

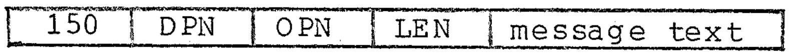


150 bytes

RECV(@ALPHA,1,120)

Wait for message or 2 seconds, whichever occurs first. When the message arrives it will look like:

ALPHA



3.3.3 Exit the Process. Another system function provided allows a task to terminate itself. The format of an exit process call is:

CALL EXIT(TERM)

where:
TERM is the termination code, >=0.

The termination code is returned to the notification process on termination (CBNTF in the context block). The format for this notification message is:

80	RESP	IOH	LFN	0	PNM	DIR

where:

RESP	is the termination code,
IOH	is the I/O handler name for the terminal which was connected to the terminated process (note that the terminal is connected to the process which receives this message),
LFN	is the logical file number for the terminal,
PNM	is the name of the process which was terminated, and
DIR	is the 6 byte directory name for the terminating process.

The termination code is forced positive since negative codes are system detected errors, such as unimplemented instruction trap, receive message buffer overflow or bad arguments to a system function. Figure 3-3 lists the system detected error codes. A zero code means normal termination.

Figure 3-3
System Detected Error Codes

Code	Meaning
0	Normal termination
-1	Illegal memory reference
-2	Unimplemented instruction
-3	Illegal message definition field in user send request (must be less than or equal 2)
-4	Illegal system request function code
-5	Receive sequence bit table overflow, unrecoverable error
-6	Send sequence bit table overflow, unrecoverable error
-7	Receive message buffer overflow
-8	Message buffer address not in process address space
-9	Message to be sent not within process address space (either a bad address or length)
-10	Message to be sent not on word boundry

3.3.4 Read the Time. Each processor on DCS has a clock. Two time functions are available, return the system clock and return the current date-time block. The format for a read clock call is:

```
CALL CLCK(FCT,ADDR)
```

where:

FCT	function requested: 0 return clock variable 1 return date-time block
ADDR	address of an area to return the value in.

A clock variable is a 2-word integer value incremented each time a clock interrupt (60 cycle clock) is received. The format for a clock value is:

```
15 14 . . .       0 15 14 . .          0
|   high order     |    low order      |
```

the sign bits in both words are always off (0).

The date-time block is returned as a 10 byte representation of the current time:

```
| D D | M M | 7 Y | H H | M´ M´ |
```

where:

DD	Day of the month (1-31)
MM	Month of the year (1-12)
7Y	Year (0-9)
HH	Hour (0-24)
M´M´	Minute (0-60)

3.4 Input/Output Facilities

The DCOS I/O Handlers are system processes which supervise all input/output devices and provide I/O services to other programs. these services are requested by messages sent to an I/O Handler which replies to the requests by sending messages to the requestors. In addition to responding to specific requests, the I/O Handler also provides the terminal user with certain special services; one is the recognition of Control-C on input, which causes the I/O Handler to send a special message which results in the process being interrupted. a user process finds the name of the I/O Handler process for its terminal (if one is connected at initiation time) in the "CBIOH" word of its context block.

3.4.1 Message Formats. Requests to the I/O Handler are in the general format of other DCOS messages, with some additional fields. Request messages to and response messages from the I/O Handler have the following general format (all offsets and values hereafter are in hexadecimal, unless otherwise noted):

FIELD	OFFSET	DESCRIPTION
IODPN	0	Destination Process Name (DPN).
IOOPN	2	Originating Process Name (OPN). Note that the value of this field in response messages permits the user process to distinguish between responses from two different I/O Handlers. (The LFN returned is not sufficient, since LFNs are not unique among I/O Handlers.)
IOLEN	4	Message Length (LEN). This is the length of the entire message, including any arguments or returned values.
IOLFN	6	Logical File Number. This byte serves as a logical name for communications between the requestor and the I/O Handler about a particular file. The requestor may match replies to requests by examining this field.
IOOP	7	Operation Code (request messages only). This byte is one of several codes described below, specifying what the

		request is.
IORSP	7	Response Code (response messages only). This byte is the error number if any (0 indicates successful completion of the request). The meanings of the various error codes are listed below.
IOARG	8	Argument Block (request messages only). The nature of the arguments depends on the particular function to be performed, as described below.
IORVL	8	Returned Value (response messages only). This field constitutes the balance of the message and depends on the particular request, as described below. Its length may be derived from the message length (IOLEN).
IOCNT	8	Byte Count for Read (request only). Count of the maximum number of bytes to be returned in response to a READ operation.
IOFLG	8	Open Flags. Flags used in RESERVE and OPEN operations and returned as values.
IOFFS	A	Free Format Specification (request only). Start of a free format file specification supplied as an argument to a RESERVE or OPEN request.
IODEV	A	Device Name. Start of the device name field in a fixed format file specification used or returned by a RESERVE or OPEN request.
IOFLN	10	File Name. Start of the file name field in a fixed format file specification used or returned by a RESERVE or OPEN request.
IOEXT	16	File Name Extension. Start of the file name extension field in a fixed format file specification used or returned by a RESERVE or OPEN request.
IODIR	18	Directory Name. Start of the directory name field in a fixed format file specification used or returned by a RESERVE or OPEN request.
IOWLD	20	Wild Character Mask. Wild character mask word in a fixed format file specification used or returned by a RESERVE or OPEN request.
IORFN	22	Returned LFN (response only).

		The LFN returned by a RESERVE or OPEN operation.
IOPOS	8	Position Number (response only). Position number returned by a SENSE POSITION request.
IOHNM	8	Process Hardware Name. The hardware name of a process used or returned by various requests, such as LIST LOGICAL FILE NUMBERS.
IOCLS	8	Close Argument (request only). One word argument to a CLOSE request.
IOSDS	C	Set Status Argument (request only). One word status argument to the SET DEVICE STATUS request.
IORDS	8	Read Status Value (response only). One word value returned by the READ DEVICE STATUS request.
IOTRT	8	Transfer To Name (request only). The hardware name of the process to which the device should be transferred on a TRANSFER DEVICE request.
IOTRF	A	Transfer From Name (request only). The hardware name of the process from which the device should be transferred on a TRANSFER DEVICE request.
IORDN	8	Read Status Name (request only). On a READ DEVICE STATUS request, the hardware name or the process which has reserved the device.
IORDL	A	Read LFN (request only). The LFN for which a READ DEVICE STATUS request should be executed.
IOLSN	8	List Name (request only). The hardware name of the process in a LIST LOGICAL FILE NUMBERS request.
IOLSL	A	List LFN (request only). The LFN argument to a LIST FILE NAME operation.

A Logical File Number (LFN) is a "name" for the message dialogue between the I/O Handler and a user process in regard to the reservation of a specific device or an opening of a particular file on a device. Most operations concerning a given LFN are synchronous. That is, a process must wait for a response to a request associated with a particular LFN before it sends the I/O handler another request associated with that LFN. Requests may, however, be pending for several different LFNs at the same time. Also, LFNs assigned by an I/O Handler are unique among user processes; in the present implementation, a specific device will in general always have the same LFN. User processes should not, however, assume that the assignment of LFNs to devices is fixed; it may often change with new versions of the I/O Handler. The LFN for a device is returned to the user when he requests that a device be reserved for his use. See the description of the reserve operation code in the next section.

3.4.2 Operation Codes. The following are the currently available operation codes (codes preceded by an asterisk have not been implemented):

NAME	CODE	DESCRIPTION
OPRSM	00	READ SYMBOLIC. The only argument is the IOCNT field (one word), the maximum number of bytes to be read. Returned value is the string of bytes read.
OPRB1	10	READ BINARY I. Same argument and value as READ SYMBOLIC.
OPRB2	20	READ BINARY II. Same argument and value as READ SYMBOLIC.
OPRVR	30	READ AND VERIFY. Same argument and value as READ SYMBOLIC.
OPWSM	01	WRITE SYMBOLIC. The only argument is the string of bytes to be written. No value is returned in the response message.
OPWB1	11	WRITE BINARY I. Same argument and value as WRITE SYMBOLIC.
OPWB2	21	WRITE BINARY II. Same argument and value as WRITE SYMBOLIC.
OPWVR	31	WRITE AND VERIFY. Same argument and value as WRITE SYMBOLIC.
OPWEF	41	WRITE EOF. There are no arguments or value. A physical EOF mark will be written on devices which use such marks.
OPWER	51	ERASE. Same argument and value as WRITE SYMBOLIC.
OPSBS	02	SENSE BUSY. There is no argument or returned value. The returned response code (ECBSY or ECNRM) indicates if the device was busy.
OPSPS	12	SENSE POSITION. There are no arguments. The returned value is the IOPOS field (one word), the position, which is device dependent.
OPLPR	22	LIST PROCESSES HAVING RESERVED DEVICES. There are no arguments. The value is a string of words, each of which is the name of a process which has reserved at least one I/O device.
OPLNO	32	LIST LOGICAL FILE NUMBERS. The argument is the IOHNM field (one

word), the hardware name of a process. The value is a string of the LFNs in use by that process, one byte each. If the argument is zero or absent, the list returned will constitute the LFNs associated with devices not reserved by any process (where these LFNs are arbitrarily assigned by the I/O Handler).

OPLNM	42	LIST FILE NAME. The arguments are the IOLSN field (one word), which is the name of a process, and the IOLSL field (one byte), which is an LFN in use by that process. The value is the fixed format file specification associated with the pair of arguments. A zero process name is interpreted as in LIST LOGICAL FILE NUMBERS.
OPPRW	03	REWIND. There is no argument or value. The effect is to rewind the device if that operation is defined for it; if it is not, this is a no-operation.
OPPRU	13	REWIND AND UNLOAD. Same as REWIND, except that the device will be unloaded, if that is meaningful.
OPPFF	23	POSITION FILE FORWARD. There are no arguments. The value is the position after the operation. The effect is to skip to the start of the next file, for sequential access devices; for direct access devices, it positions to the end of the current file.
OPPFR	33	POSITION FILE REVERSE. There are no arguments. The value is the position after the operation. The effect is to position to the beginning of the file preceding the current one, for sequential access devices which can be backspaced; for other devices, it is a no-operation.
OPPRF	43	POSITION RECORD FORWARD. There are no arguments. The value is the position after the operation. The effect is to position the start of the next physical record after the current one.
OPPRR	53	POSITION RECORD REVERSE. There are no arguments. The value is the position after the operation. The effect

is to position the file at the start of the record before the current one, for devices which can be backspaced; for other devices, it is a no-operation.

OPPPA 63 POSITION ABSOLUTE.

The arguments are device dependent and consist of one or more words specifying an absolute position on the device. On magnetic tape, only one word is used; it indicates the number of the file on the tape, counting from 1. On disk, only one word is used; it indicates the block number in the file on which the next read or write operation should be executed.

OPRDV 04 RESERVE DEVICE.

The argument and the first value are file specifications, as described below. This operation reserves a device without opening a file on it. Thus, if the file specification is a disk file, the file is not opened. Is reserved to the exclusive use of the reserving process. The IOLFN field in the response message will be set to the LFN supplied in the request; the LFN of the device, which must be used in any further requests related to that device, will be returned in the IORFN field (the first byte after the file specification). For disk, the reservation is for a software channel, rather than the entire device.

OPOFL 14 OPEN FILE.

The argument is a file specification as described below; it is optional if the file was previously OPENed and CLOSEd without RELEASing the device, as described below. The values, if any, are the fixed format version of the file specification for the device and file, with changes made to reflect the values of unspecified parameters, such as the file position for disk files or the file number for magnetic tape files, and the IORFN field, which will contain the LFN for the device (which may differ from the one in the IOLFN field, if the device was not already reserved).

OPFDV 05 RELEASE DEVICE.

The arguments are the IOHNM field, which contains the hardware name of a process, and the IOLNO field, which contains an LFN in use by that process. If the arguments are absent, they default to the values of the IOOPN and IOLFN fields respectively. The effect is to release the device, if any, associated with the LFN given. If no device is associated with the LFN, this is a no-operation. If a file is open on the device, it is closed.

OPFAD 15 RELEASE ALL DEVICES.

The argument, which defaults to the requesting process's name if not supplied, is a string of one or more hardware process names. For each name supplied, all devices reserved by that process will be released. If files are open on any such devices, they are closed. There is no value.

OPCFL 25 CLOSE FILE.

The argument is one word, the value of which determines certain parameters of the close process; if not present, it is assumed to be zero. The only value currently available is 1, which indicates that no EOF mark should be written on devices which use them. There is no value. The effect is to close the file. Note that the LFN and the file are not released. A subsequent open may use just the LFN rather than the complete file specification.

OPRSD 35 RESET DEVICE.

ECIOC will be returned if the RESET operation is not implemented for the device specified by the LFN. If RESET is implemented for the device and an operation is in progress, the operation will be terminated, an ECBSY response will be returned for it, and an ECNRM response will be returned for the RESET operation. If no operation is in progress, an ECNRM response will be returned.

* OPRDS 06 READ DEVICE STATUS

The arguments, which are optional, are as for the OPLNM operation; if they are not supplied, the requestor's own name and LFN will be used. The value, returned in the IORDS field (one word), is the Device Status Word for the device specified by the LFN. Bit assignments for the Status Word are given below.

OPSDS 16 SET DEVICE STATUS
There are three arguments, the first two of which are the same as for the OPRDS operation. The third is the IOSDS field (one word), which is the word which should be used to replace the user-accessible bits in the Device Status Word. The value, returned in the IORDS field (one word), is the previous value of the Status Word.

* OPSYN 07 SYNCHRONIZE.
There is no argument or value. The effect is to direct the I/O Handler to wait to transmit the response message for any I/O operation in progress on the specified device until the next request message for that device arrives (this does not apply to the response to the SYNCHRONIZE operation itself). This constraint applies only until the next request message arrives; that is, it affects only one operation. If an I/O operation is in progress, the ECSYN error code is returned. If SYNCHRONIZE is illegal for the device specified, ECIOC is returned; otherwise, the ECISY error code is returned. Note that this is a second exception to the message protocol. This operation may be used to resolve the ambiguity noted above; if the user process first sends a request for this operation, and then notes whether the response code in the next response message is one of the two SYNCHRONIZE response codes (no other operation ever returns them), it can decide whether the last previous operation is still in progress (and whether its response has been arrested by the SYNCHRONIZE operation).

OPTDC 08 TRANSFER DEVICE.
The first argument, IOTRT (one word), is the name of the process to which the device is to be transferred; the second

argument, IOTRF (one word), is the name of the process which currently has the device assigned. There is no value. If the requesting process currently has the device reserved, the second argument is optional; if not, it is required. If the requestor is privileged to make this request, the device is transferred. User processes may only transfer devices which they currently have reserved; certain system programs may use the general form. If any file is currently open on the device, its status is unchanged.

OPCNV 09 CONVERSE.

The arguments are the same as for a WRITE BINARY I command. The value is the same as for a READ SYMBOLIC command. The effect is to execute a WRITE BINARY I followed by a READ SYMBOLIC. The length of the READ SYMBOLIC is the maximum defined for the device. This operation may be used to prompt and then read a line from a terminal.

3.4.3 File Specification Format. A file specification as supplied to the I/O Handler may be in either a free or a fixed format. The free format consists of one "FSFLG" word followed by an arbitrary length byte string of ASCII characters, constituting a file specification having the general form

```
<device>:<filename>.<extension>[<directory>]
```

where the default "<device>:", if it is not specified, is "DSK:", and the default "<directory>" is the requesting process's LOGIN directory. The "." is optional if the "<extension>" is not supplied, and the "[" and "]" are illegal if the "<directory>" is not supplied. The "]" is optional if the "<directory>" is supplied. "*" Is supplied as the "<filename>" if it is omitted, as the "<extension>" if both it and the "." are omitted, and as the "<directory>" if it and the "[" are omitted. The fixed format, which is used by the I/O Handler whenever it returns a file specification as a value, is as follows:

FIELD	OFFSET	DEFINITION
FSFLG	0	Open Flags (1 word).
FSDEV	2	Device Name (6 bytes).
FSFLN	8	File Name (6 bytes).
FSEXT	E	Extension (2 bytes).
FSDIR	10	Directory Name (6 bytes).
	16	(Reserved) (2 bytes).
FSWLD	18	Wild Characters Mask (1 word).
	1A	(Reserved) (6 bytes).

FSDEV, FSFLN, FSEXT, and FSDIR are the left-justified ASCII representations of the device, file name, and directory name. The device name must be of the form described below. The response from the I/O Handler to a RESERVE or OPEN request includes the fixed format specification for the file. The contents of the file specification returned will correspond to the name of the device as reserved and the file as opened. (On a RESERVE operation, the file name, extension, and directory will returned as supplied.) An OPEN operation will return a file specification with the reserved and FSWLD fields cleared; a RESERVE operation will return them as supplied in a fixed format specification or as filled in by the I/O Handler's file scanning routines. In the latter case, FSWLD will form a left-justified bit mask for the FSFLN, FSEXT, and FSDIR fields indicating wild characters in the specification. (Wild characters are "?"

in place of specific letters and "*" in place of entire elements; wild devices are not allowed.) The user process may make use of this wild character information in opening a file if it first calls the I/O Handler to reserve the device. FSFLG determines the specific nature of certain operations and is returned, appropriately modified, in the response. Possible values for the FSFLG word are as follows:

NAME	CODE	FUNCTION
OCORD	0	Open file for reading.
OCOWR	1	Open file for writing.
OCOWN	2	Do not deallocate unused blocks at the end of the file when it is closed.
OCDIR	4	Open directory file for reading.
OCOUP	8	Open file in update mode.
OCOAP	10	Open file in append mode.
OCDEL	20	Delete file.
OCREN	21	Rename file (unimplemented).
OCRIB	40	Open RIB of file for reading.

Code OCREN, when implemented, will require that the file first be OPENed under the old name and then reOPENed under the new name, with this code and the new name specification as the arguments; this will have no effect except to rename the file. In addition to the above, the high-order bit of the FSFLG word (bit 15) indicates whether the free or fixed format is being used. If zero, the free format is indicated and if one, the fixed format is supplied.

In converting from free format to fixed format, the I/O Handler ignores all characters lower in the ASCII collating sequence than a blank, except to treat them as separators. All lower case characters (including the "lower case" equivalents of "[" and "]") are converted to upper case. The only characters which may be used in names are letters ("A" - "Z") and digits ("0" - "9") and the wild characters ("?" and "*"). Other characters, however, may be used if each is preceded with a "´" character, which indicates that the following character should be taken literally (this applies to "´" itself and all other characters as well, including lower case and control characters). Furthermore, any sequence of characters not including ´"´ may be enclosed in a matching pair of ´"´ characters, and will be interpreted as a single name. If a name supplied is longer than permitted by the equivalent field in the fixed format, it will be truncated on the right to fit.

The symbol FSSIZ gives the size in bytes of a complete fixed format file specification.

3.4.4 Device Names and Specifications. Device names are up to six characters long and may be either the name of a specific physical device or simply a generic name for a certain class of devices. A generic name is the same as a physical name, except one or more of the trailing letters is omitted. Thus, in the table below, the device "DSKA00" appears. This denotes disk channel 0 for disk pack 0 of disk file structure A. Hence, the generic "DSK" (or "DSKA") refers to any one of the available channels, "DSKA00" through "DSKA0B". The following are the currently available devices for the I/O Handler on machine 1:

NAME	DESCRIPTION
DCS	System device (equivalent to "DSK:[DCS]").
DSKA0n	CALCOMP Disk, Channel n (0 <= n <= B).
LPT0	CENTRONICS Lineprinter.
MTA0	KENNEDY Magnetic Tape Drive.
NULA0n	Null device (write-only memory), channel n (0 <= n <= 1).
PLT0	CALCOMP Plotter.
PTR0	REMEX Paper Tape Reader.
SYS	System device (equivalent to "DSK:[SYS]").
TTX1	TELETYPE Model 35.
TTY2	TELETYPE Model 33 (near Machine 1).
TTY4	DATAPOINT 3300 terminal.
TTX50	MICROSWITCH Keyboard and TEKTRONIX Scope.

The following devices are those currently available from the I/O Handler on machine 3:

NAME	DESCRIPTION
NULB0n	Null Device, channel n (0 <= n <= 1).
TTY0	DATAPOINT 3300 terminal.
TTY3	DATAPOINT 3300 terminal.

Disk channels are software constructions used for buffering purposes. The number of disk files which may be open at any one time is constrained to be at most the number of available channels. Absolute position numbers on disk reference the block number of the next disk block to be read, relative to the start of the file (the first block is block 1).

The CENTRONICS Lineprinter accepts only a limited character set (no lower case). Maximum line length is 80 characters.

The KENNEDY Magnetic Tape Drive reads and writes 9-track, industry-compatible 1/2-inch magnetic tape. If a file name is specified on an open command, the software will assume that the tape is in QOS3-DCOS directory format; that is, file 1 on the tape is a directory file, the first block of which is the directory. If in directory format, the tape may be treated much as a disk, except that only one file may be open at one time, and space occupied by deleted files is not reclaimed until there are no useful files beyond them on the tape. A directory holds a maximum of 32 files, including deleted but unreclaimed files. If no file name is specified, then the position number is taken to be the number of the file desired, and the tape is positioned to that file.

The driver for "TTY???" devices provides limited editing of input during READ SYMBOLIC operations. The following characters are interpreted by the driver:

↑U Delete current line.

↑R or ↑Y Retype current line.

RUBOUT Delete last character on current line.

↑C Terminate input operation and send Control-C Interrupt control message to the process controlling the terminal. (Presently, this message will cause the process to be terminated.)

On input, the high-order bit of all characters will be set to "1". If the DSLCP in the device status word is not set, characters input will be echoed to the terminal as received, except that Control-R (↑R), Control-Y (↑Y) and RUBOUT are not echoed, ESCAPE is echoed as "$" and all other control characters are echoed as "^" followed by the corresponding ASCII letter. (For example, Control-A (↑A) is echoed as "↑A".)

For detailed descriptions of the actions of the READ and WRITE commands, see the Lockheed IOCS specifications [5].

3.4.5 Device status Word Bit Assignments. In the following table, names followed by a "*" may not be set or reset by the user via the OPSDS operation; any attempt to do so will be ignored.

NAME	CODE	DESCRIPTION
DSBSY*	8000	I/O REQUEST IN PROGRESS. An I/O request relating to this device is currently in progress.
DSUWP*	4000	WRITE PROTECT. The device or file is write-protected.
DSNAC*	2000	RESPONSE MESSAGE NOT ACCEPTED. The last response message for the device was returned "MATCH--NO ACCEPT" and is presently waiting for retransmission.
DSOPN*	1000	FILE OPEN. If set, the device is currently in "OPEN" status.
DSWDR*	800	WAITING FOR DIRECTORY RESPONSE. If set, processing of a request requiring the reserving process's LOGIN directory has been suspended, pending receipt of a response to a message requesting that name.
DSLCP	400	LOCAL COPY (TERMINALS ONLY). If set, there will be no echo on input for the device; if cleared, the device will echo input in its normal manner.
DSHDX*	200	HALF-DUPLEX (TERMINALS ONLY). If set, DSLCP will be ignored, as the device supports only "echo" mode.
DSSYS*	200	SYSTEM FILE (DIRECTORY DEVICES ONLY). If set, the file currently open is from logical device "SYS:" or "DCS:".
DSRSM*	100	READ SYMBOLIC STARTED (TERMINALS ONLY). If set, on completion of the current read request, the input will checked for a Control-C; if one is found, a Control-C Interrupt message will be sent to the requesting process.
DSWIO*	80	WAITING FOR I/O OPERATION. If set, processing of a request has been suspended pending completion of a physical i/o operation.
DSRLS*	20	RELEASE OPERATION STARTED. If set, a release operation is in progress on the device.

DSOPC*	10	OPEN OPERATION STARTED. If set, an open operation is in progress on the device.

Figure 3-4
I/O Handler Error Codes

NAME	CODE	CONDITION
ECNRM	0	Normal Completion.
ECDNR	1	Device Not Reserved By Requestor. No I/O operations except RESERVE, OPEN, and, for privileged processes, TRANSFER are permitted before the device in question has been reserved by the requestor.
ECBCN	2	Byte Count Non-positive. The byte count supplied as an argument to a READ operation was non-positive.
ECIOC	3	Illegal Operation Code.
ECNRS	4	Device Requested could not be Reserved.
ECBSY	5	Device Busy.
	6	(Reserved code).
ECPRT	7	Access Protection Failure. A WRITE operation failed because the device or file in question was write-protected or an OPEN operation failed because the requestor was not privileged to access that file.
ECNRD	8	Device Not Ready.
ECTRN	9	Transmission Error.
ECEOT	A	End-of-Medium or End-of-Tape.
ECEOF	B	End-of-File.
ECBOT	C	Start-of-Medium of Beginning-of-Tape.
ECCNV	D	Conversion Error.
ECNBF	E	Buffer Not Available. The I/O Handler could not allocate a buffer it required to complete the request.
ECPRM	F	Parameters Out of Range. This error should never occur under DCOS; it reflects an internal error in the I/O Handler.
ECLFN	10	Illegal Logical File Number.
ECBOV	11	I/O Device Buffer Overflow. On a READ operation, the data block read overflowed the input buffer; on a WRITE or POSITION operation, the data supplied in the request message overflowed the device buffer.
ECIRQ	12	Invalid Requestor.

		The requestor was not privileged to execute the operation requested.
	13	(Reserved code).
ECSYN	14	Synchronize Succeeded.
ECISY	15	Synchronize Failed. A SYNCHRONIZE operation was requested when no I/O operation was in progress.
ECNOP	16	File Not Open. An I/O transfer or position operation was attempted on a directory device before any file had been opened.
	17-1F	(Reserved codes).
ECOPN	20	Open Attempted While File Open. An OPEN operation was attempted when a file was already open on the specified device.
ECRDE	21	Read after End-Of-File. This error should never occur under DCOS: "ECEOF" should be returned instead.
ECFRF	22	File RIB Full. A disk operation could not be completed because all pointers in the file's Retrieval Information Block (RIB) had been used.
ECFNF	23	File Not Found. The file specified in an OPEN operation could not be found.
ECDFF	24	Directory Full. The file specified in an OPEN operation could not be entered in the directory on that device because the directory was full.
ECDRF	25	Directory RIB Full.
ECDKF	26	Disk Full.
ECFWP	27	File Write Protected.
ECDNF	28	Directory Not Found. The directory specified in an OPEN operation could not be found.
ECPNM	29	Pack Not Mounted.
ECSDS	2A	System Stopped in Disk Software.
ECRST	2B	System Reset since Open.
ECODM	2C	ODCB Modified since Open.
ECFEI	2D	File Entered Improperly into MFD.
ECFOE	2E	File Open Elsewhere. A file may not simultaneously be opened for writing on one channel and opened for anything on another.
ECIRS	2F	Improper Read/Save Format.

		This error should never occur under DCOS.
ECWIC	30	Write on Illegal Cylinder. This error should never occur under DCOS.
ECUKC	31	Unknown Command. Generally, this error indicates an illegal OPEN parameter was supplied.
ECDWP	32	Device Write Protected. This error should never occur under DCOS; "ECPRT" should be returned instead.
ECE20	33	Disk Error 20. This error should never occur under DCOS.

3.4.6 Definition Files. All symbols listed above are defined in the standard system definition files "CB.DE" and "IOC.DE", which may be placed on the front of a MOLSUE program. Users are encouraged to make use of these definition files as much as possible, in order that any changes in the values of the above symbols may easily be incorporated in programs referencing them. Changes are not anticipated, but are possible.

3.5 Process Initiation and Termination

See the command process documentation (section 4.1) for a description of facilities for initiating and killing processes.

4. SERVICES and PROCESSES

This section describes some of the processes which are available on the system.

4.1 Command Interpreter Process

The Command Process is a system procedure operating under DCOS. This process services requests from terminals and from other processes running under DCOS. Requests come in the form of "command messages". Each command message is acknowledged either by a response message indicating that the service has been performed, by an error message, indicating why the service could not be performed, or, in the case of the command 'QUEUE', which requests service at a later time, an acknowledgement of the request.

4.1.1 Service Requests. In the description that follows requests from a terminal are made by entering the appropriate command string in response to the prompt character ("."). For example, to run the mail process (described in section 4.4) one enters the command string

```
.RUN MAIL
```

where stands for carriage return. Terminal input is done in symbolic mode so the standard editing characters are meaningful (↑U - delete current line, ↑Y or ↑R - retype current line, and rubout - delete last character).

Requests from other processes use the higher level protocol:

LEN	ID	0	command

where
- LEN is the length of the message in bytes,
- ID is the request identifier, and
- command is the command string.

A command string consists of one or more parts, separated from each other by one or more spaces. All commands begin with the "command operator". Most commands require one or more "arguments". In this case, the argument or arguments make up the second part of the command. Multiple arguments are separated by commas. Commands which cause a process to be initiated, may have a third part, the "program string", which is sent as a message to the process which the command has initiated. For a list of current command formats see figure 4-1.

The command operator is a string of characters sufficient for the command process to recognize the command. From one to three characters are required to

Figure 4-1
List of Command Process Commands
(commands preceded by an asterisk have not been implemeneted)

```
*  CC[ONTINUE]

*  CO[NTINUE]

*  COP[Y]<BLNKS><PGM STRING>

   CR[UN]<BLNKS><FILE SPEC.>[,<MACH.NO>][<BLNKS><PGM STRING>]

   CS[TART][<BLNKS><PROCESS H.N>]

*  D[EPOSIT][<BLNKS><PROCESS H.N>,<LOCATION>,<VALUE>

*  DI[RECTORY][<BLNKS><PGM STRING>]

*  E[XAMINE]<BLNKS><PROCESS H.N>,<LOCATION>

   K[JOB]

   KI[LL][<BLNKS><PROCESS H.N>]

   L[OAD]<BLNKS><FILE SPEC>[,<MACH.NO>][<BLNKS><PGM STRING>]

   LOG[IN]<BLNKS><USER DIR.NAME>

*  Q[UEUE]<BLNKS><TIME>,<ANY COMMAND>

   R[UN]<BLNKS><FILE SPEC>[,<MACH.NO>][<BLNKS><PGM STRING>]

*  S[TART][<BLNKS><PROCESS H.N>]

*  T[YPE]<BLNKS><PGM STRING>]

   <FILE SPEC>[,<MACH.NO>][<BLNKS><PGM STRING>]
```

notes:

1. Items in [] are optional, at least under certain circumstances.
2. <BLNKS> is a string of one or more spaces
3. Numerical input arguments are base 16.

uniquely identify a command. If additional characters are provided, the first six characters of the command must match those expected. Further characters will be ignored. If no command is found which matches the command string, the command is assumed to be 'RUN' and the entire command string is treated as argument and program strings.

Once a command has been recognized, the command process scans off the required arguments and attempts to perform the requested service. The reply, error, or acknowledgement message is returned. Terminals are then initialized with the command process prompt character ('.'), and another command is awaited.

4.1.2 Loading a Process. The commands 'RUN', 'CRUN', and 'LOAD' all cause a process to be loaded on the DCS. The arguments to these commands are: a specification for a load file in the free format required by the I/O handler, and a machine number for the machine on which the process is to be run. The file must be output from the relocating link loader. This is the process to be loaded in relocatable binary form.

Certain fields may be omitted from the arguments. If no extension is specified in a disk file specification, the command process will first attempt to find a file with no extension, and, if that fails, will attempt to find the file with a default extension 'RB' (for 'relocatable binary'). If no directory name is specified, the command process will first look for the file on the directory under which the terminal user or process which sent the command message is logged in, and will then look for the file on the system directory 'DCS'. If no machine number is specified, (the second argument is optional), the command process will attempt to load the process successively on each processor on the DCS, by increasing machine number until a load attempt is successful or all machines have failed to load the process.

When the process has been loaded, the command 'LOAD' is completed and the acknowledgement message is sent. The program string (possibly empty) is sent to the new process, and if the command is from a terminal, the terminal is readied for another command input. The command 'CRUN' loads the process and, after sending the program string message, sends a message to initiate execution of the process. The command 'RUN' (and the implicit run command) loads and starts the process and in addition, if the command to run the process came from a terminal, ownership of the terminal is transferred to the new process so that the user can communicate with the process. The hardware name of the terminal I/O handler and the terminal LFN are stored in the process context block and are passed in the program string message to the process. If the command to run the process came from another process, no transfer occurs, but the initiating process name and the identifier from the command message are passed to the process in the program string message.

The format of the program string message sent to every process initiated is:

LEN	LFN	1	IOH	string

where:

LEN	is the length of the message.
LFN	is the logical file number for the terminal connected to the process (if any).
IOH	is the I/O Handler name for the terminal (if any).
string	is the string of characters after the file name on the load request (not including the carriage return).

As an example, suppose the following command was entered at the terminal:

```
.PIP X=Y.SU↩
```

The command process would initiate the process named "PIP", connect the terminal to it and send "X=Y.SU" as the "string" in the program string message.

4.1.3 Killing a Process. The command 'KILL' terminates a process. It takes one argument, the hardware name of the process to be terminated. If the last command prior to the 'KILL' is the command which initiated the process, the argument is optional. Any terminal user who has logged in with the same user name as the process to be killed, or any process which has the same user name, may kill a process. If the process had a terminal, that terminal is transferred to the process named in the CBNTF field in the process context block (initialized to the command process if the load request was from a terminal). A termination reply message is written and the terminal is initialized to accept a new command.

4.1.4 Logging on to DCS. The command 'LOGIN' allows a user to log onto the DCS. The command requires one argument, the user name, currently the user directory name. This command is temporarily implemented by the command process. Future versions of DCOS will log users in by connecting the user to a login process.

4.1.5 Logging off DCS. The command 'KJOB' allows a user to log off the DCS. The command requires no arguments.

4.2 DIRECT (Directory Process)

<Not available at this time.>

4.3 LAP (Lockheed Assembly Program)

<Not available at this time.>

4.4 MAIL

The mail process provides a message (or mail) sending and receiving service. Messages are addressed to a person by using their DCOS login name. Commands are available for sending, receiving, and saving messages.

Mail is maintained in the file 'MSG.TX' in the users file directory. Mail received from another user is copied into 'MSG.TX' and the receiver is notified the next time he logs on that mail has arrived. Mail is retained in the file until the receiver explicitly requests that the file be deleted. Users are encouraged to use 'MSG.TX' like a mailbox. Mail to be saved should be archived in another file (commands are provided for saving pieces of mail).

The mail process maintains a current mail file and a current mail pointer. When MAIL is entered, 'MSG.TX' is initialized as the current mail file, the subjects of all messages in the file are listed, and the current mail pointer is set to the first message. At this point the top level command loop is entered and the user may issue a request.

This description uses the following conventions:

<mp>	Message pointer: i message # i, and i,j messages i through j inclusive. A null message pointer means the current message.
<file spec>	A standard DCOS file specification: device:filename.ext[dir] with the usual defaults.
<user spec>	User name. Currently logon directory name.

The commands available are:

<mp>COPY <file spec>

Copy the messages specified in the range <mp> to the file <file spec>. This command allows a user to make copies of selected pieces of mail in an archive file.

DELETE

Delete the current mail file. With this command a user can delete a mail file. This command should be used when deleting the default mail file after having read and saved the selected mail.

EXIT

Exit the process. Before exiting another attempt is made to send unsent mail.

FILE <file spec>

This command allows a user to redefine the current mail file. The old mail file is closed and the new one opened.

LIST

This command lists the subjects of all messages in the current mail file.

NEXT

This command moves the current message pointer to the next message. If the current message pointer is at the end of the message file the command is a noop.

PREV

This command moves the current message pointer to the previous message in the current mail file. If the pointer is at the front of the file the command is a noop.

<mp>READ
<mp><blank or null line>

This command prints the message at the current message pointer on the user's terminal and leaves the current message pointer at the beginning of the next message.

SEND <user spec>

This command initiates a send mail request to the user specified. A subject for the mail is requested and then the user may enter the text of the mail. Control characters recognized by the mail process while entering the text are:

↑Z send the message
↑F incorporate message from file
↑V retype message
↑E do not send message, return to command loop

4.5 MOLSUE

<Not available at this time.>

4.6 PIP (Peripheral Interchange Program)

<Not available at this time.>

4.7 QED

<Not available at this time.>

4.8 RLDP (Ring Load Process)

<Not available at this time.>

4.9 RLL (Relocating Link Loader)

The Relocating Link Loader (RLL) links together object modules to produce a relocatable binary module which can be run on the DCOS. To run RLL, one types the following to the command process (constructs in brackets "[" and "]" are optional):

```
.RLL↵
```

When loaded, RLL will prompt the user terminal with a '*', and the user should type a command string of the form:

```
*<RB-FILE>[,<MP-FILE>]=<OB-FILE>[/<SW>][,<OB-FILE> ....]↵
```

where:

<RB-FILE>	is the relocatable binary output file. If the extension is omitted, .RB will be the default.
<MP-FILE>	is the symbolic memory map output file, where the default extension is .MP.
<OB-FILE>	is an object input file, where the default extension is .OB. If no .OB file is found, a search is made for a file without an extension.
<SW>	is either "LU" or "LC". A toggle is maintained within RLL indicating whether the object file is to be linked in the unconditional or conditional mode. The toggle is initialized to the unconditional (/LU) mode, and changed when a switch indicating the other mode is interpreted in a command string.

For example,

```
*DCS,DCS=DCS1,DCS2,LIB/LC,LIB2,DCS3/LU↵
```

links a process together, where DCS.RB is the relocatable binary output file, DCS.MP is the symbolic memory map output file, DCS1.OB, DCS2.OB and DCS3.OB are the object modules linked in the unconditional mode, and LIB.OB and LIB2.OB are object modules linked in the conditional mode.

Command strings which cannot fit on one line may be continued by placing a semicolon at the end of a line to be continued (instead of a comma). For example,

```
*DCS,DCS=DCS1,DCS2;↵
*LIB/LC,LIB2;↵
```

```
*DCS3/LU↵
```

is equivalent to the previous example's command string. The linking operations are perfromed as each line is entered, not after the entire command string is entered.

Figure 4-2 summarizes the filename conventions for RLL.

Figure 4-2
RLL Filename Conventions

Output Files (RB, MP files)

command string	filename meaning
filename	filename.RB or filename.MP
filename.	filename (no extension)
filename.xx	filename.xx

Input Files (OB files)

command string	filename meaning
filename	filename.OB or if no such file exists then filename (no extension)
filename.	filename (no extension)
filename.xx	filename.xx

4.10 RUNOFF

<Not available at this time.>

4.11 SPM (Software Protection Module)

SPM provides protection for the system while a user is debugging a process. Each instruction is interpreted so that memory references out of the process address space and certain instructions can be trapped. SPM is included in a special version of the procedures file, SPM.OB[DCS].

To run a program under DCOS and SPM control, the programmer must:

1. Link his user program with SPM.OB[DCS] instead of PROC.OB[DCS].
2. Define as a program entry point (internal) the label 'START', which must be the execution start address of the user program.
3. Run the program under DCOS as usual.

SPM interprets each user instructuion and verifies that:

1. The instruction is 'legal'. all instructions are legal except control (class 0) instructions. Exception: control instructions 'REGM' and 'MREG' are considered legal. This checks for instructions which would mess up the system, such as HALT, RETN, and other instructions which affect the processor status word.
2. The memory location referenced lies within the core block allocated to the process by the system.

In the event that an illegal instruction or an illegal memory reference is detected by the SPM, a message will be displayed on the user's teletype of the form:

```
?<error-code><program-counter><instruction><operand-address>
```

where the possible error-codes are:

IN - illegal instruction detected
MR - illegal (out of bounds) memory reference detected
ID - indirect-address error. In computing the effective operand address, more than 15 levels of indirection were counted.

The <program-counter> will be the core address of the instruction after the one that triggered the error. Exception: if the program counter itself was not within the core block allocated (mr error) then the <program-counter> will be the out of bounds address. In this case the <instruction> and <operand-address> fields will be meaningless. The <instruction> will be the instruction that triggered the error (see exception above). The <operand-address> will be the computed operand address of the instruction that triggered the error (see exception above). Another exception: if the error was an 'in' (illegal instruction) the <operand-address> will be meaningless. After the error message is displayed, the process is terminated.

5. USING DCS

This section describes how to start up the system (operating instructions), how to write a program to be executed on DCS (including an example), and the system and user libraries of useful definitions and procedures.

5.1 DCS Operating Instructions

To run the DCOS first load the QOS-3 system [7]. When interp has been loaded enter

```
.>DCS[DCS]↲
```

This loads the current version of DCOS (it takes about 30 seconds).

To halt DCOS and reload QOS-3:

1. Wait until there is no i/o activity between the Varian 620/i and Lockheed SUE.
2. Halt the SUE.
3. Step the 620/i.
4. Set P=2, U=0 in 620/i.
5. Hit system reset and then run on the 620/i.
6. Store H'80' into location 6 in the SUE.
7. Hit reset and then attn on the SUE.

After about 5 seconds QOS will have been reloaded.

5.2 Preparing a Process to Execute on DCS

5.2.1 Filename Extension Conventions. Standard filename extensions are:

Extension	Type of file
SU	MOLSUE
LA	LAP
OB	Object
TM	TREEMETA
PL	Plotter
LS	Listing
RN	RUNOFF
LP	Line Printer
RB	Relocatable Binary
DE	Structure definition files (MOLSUE)
TX	Mail file format
MP	Link loader memory map

5.2.2 SUE Programming Conventions. On the SUE the programmable flags (F1, F2 and F3) in the status word are used by the system so processes must not change them. This restriction extends to almost all of the SUE control class instructions. The only exceptions are MREG and REGM. The instructions which must not be used are: HALT, RSTS, SETS, ENBL, ENBW, DSBL, DSBW, STSM, RETN and MSTS.

MOLSUE procedures are limited to three arguments which are passed by value in registers 1, 2 and 3. Register 4 is the return address. A sample LAP call of a MOLSUE procedure is:

```
MOVW value arg 1,R1
MOVW value arg 2,R2
MOVW value arg 3,R3
JSBR procedure name,R4
```

Values returned by functions are returned in register 1. MOLSUE procedures do not, in general, save registers on entry, so registers containing values used by the program must be saved before the call and restored after the return.

5.2.3 Program Preparation Example. To run a program on DCOS it must be linked by the relocating link loader (RLL), a modified version of the Lockheed Link Loader [8]. At present RLL is only available on QOS-3.

To link a set of object modules into a relocatable binary module establishes binary input, binary output and listing output (optional) devices. The following example links together two files, MAI.OB and MUT.OB, into the file MAIL.RB. A load map is output into file MAIL.MP (system responses are underlined):

```
.>PIP↵
*TMP=MAI.OB,MUT.OB↵
*↑C
.>RLL,BI=TMP,BO=MAIL.OB,LO=MAIL.MP↵

OK
>ST↵
CURRENT CORE SIZE IN THOUSANDS>16↵
TARGET CORE SIZE IN THOUSANDS>16↵

OK
>LU↵

OK
>LU↵

OK
>FL↵

OK
>PM↵

OK
>EX↵
```

5.3 Libraries

5.3.1 System Library. No description available.

5.3.2 User Library. No description available.

6. REFERENCES

[1] Hopwood, G. L. "Notes on MOLSUE." U. C. Irvine Distributed Computer Project, Project Memo (October 1973).

[2] LAP-2 Assembler Manual. Lockheed Electronics, Los Angeles, Ca., (December 1971).

[3] SUE Computer Handbook. Lockheed Electronics, Los Angeles, Ca.

[4] Farber, D. J., J. Feldman, F. R. Heinrich, M. D. Hopwood, K. C. Larson, D. C. Loomis, and L. A. Rowe. "The Distributed Computing System." Proc. Seventh Annual IEEE Computer Society International Conference, (February 1973), 31-34.

[5] "Input Output Control System (IOCS) Specifications." Lockheed Electronics, Los Angeles, Ca. (June 1972).

[6] Rowe, L. A. "Notes on System Programming in DCOS." U. C. Irvine Distributed Computer Project, Project Memo (April 1974).

[7] "QOS-3 Version 3." U. C. Irvine Distributed Computer Project, Project Memo (August 1973).

[8] "Link Loader Specifications." Lockheed Electronics, Los Angeles, Ca. (Decemeber 1971).