Data General

Software Documentation

# Introduction to the Advanced Operating System (AOS)

# Introduction to the Advanced Operating System (AOS)

069-000016-01

*For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.*

Ordering No. 069-000016

Revision 01, October 1984

Introduction to the
Advanced Operating System
(AOS)
069-000016

Revision History:

093-000121
Original Release - April 1976
First Revision - October 1976
Second Revision - April 1977

069-000016
Original Release - June 1978
First Revision - October 1984

This document has been extensively revised from revision 02 of manual 093-000121; therefore, change indicators have not been used.

CONTENT UNCHANGED

The content in this revision is unchanged from 069-000016-00. This revision changes only printing and binding details.

# Preface

This manual describes the logical structure and operating features of AOS (Advanced Operating System) in a simple, step-by-step fashion. It presents topics in a logical progression, yet each topic is self contained, so you do not need to digest all the material in a single session. Rather, you may read the chapters selectively according to your interests.

We have assumed in this manual that you have some familiarity with minicomputer architecture and operating system concepts. If you need a more elementary introduction to AOS, aimed at a novice user approaching the system at a console, refer to *Learning to Use Your Advanced Operating System* (69-000018).

To understand any AOS feature in depth after finishing this primer, consult one or more of the publications listed in the *AOS Software Documentation Guide* (69-000020). This document lists AOS publications by title and order number and summarizes the contents of each.

End of Preface

# Contents

# Illustrations

Manager decides which operating system files, such as system utilities and shared data, each user may examine, modify or execute. The System Manager also assigns each user privileges which control the degree of multiprocessing, peripheral access, and other features each user may implement. The system's integrated hardware and software protection scheme enforces these privileges and maintains file and process integrity.

As stated above, AOS has device access procedures which are device independent. The hierarchical file structure allows you to treat any device as a file, and in some cases to access multiple files on a device (such as a magnetic tape transport). A typical AOS hardware configuration could include the console TTY's, digital plotters, moving- and fixed-head disks, diskettes, magnetic tape transports, multiple synchronous lines (with or without DCU/50 Data Control Units), paper tape readers and punches, line printers, card readers, and other devices accessed through multiprocessor communications adaptors. (See Figure 1-1.) The operating system keeps track of each device's characteristics for you, so that, for example, it will never accidentally try to write to a card reader.

The operating system is easy to use. To compile and execute higher-level language programs, the system's Command Line Interpreter (CLI) requires only a few command sequences. If you wish to use more advanced operating system facilities, you will want to acquire a more extensive understanding of the system, including process management, file structure, input/output processing and task management concepts. These concepts are described in this manual.

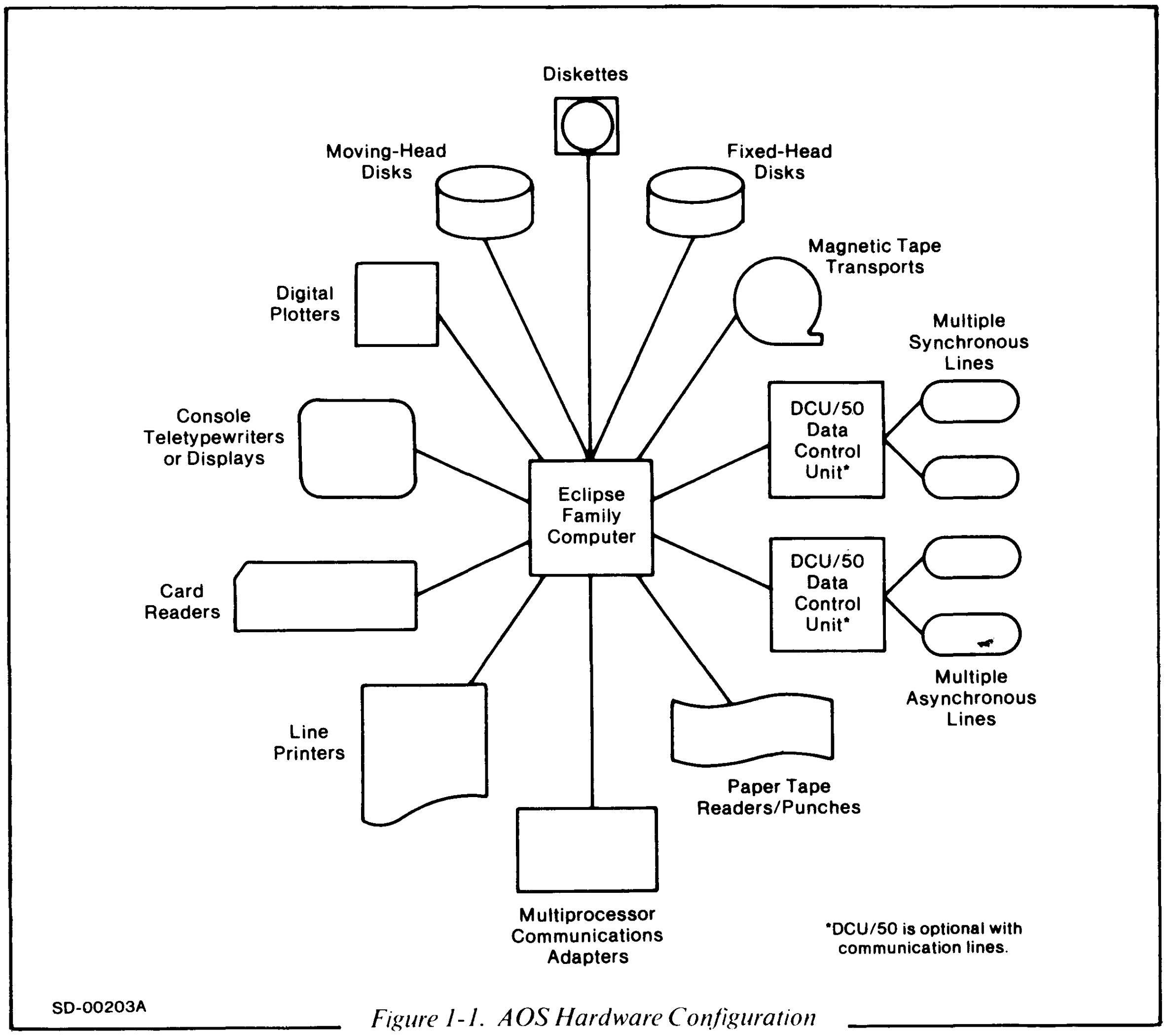

Diskettes
Moving-Head Disks
Fixed-Head Disks
Magnetic Tape Transports
Digital Plotters
Multiple Synchronous Lines
Console Teletypewriters or Displays
DCU/50 Data Control Unit*
Eclipse Family Computer
DCU/50 Data Control Unit*
Card Readers
Multiple Asynchronous Lines
Line Printers
Paper Tape Readers/Punches
Multiprocessor Communications Adapters
*DCU/50 is optional with communication lines.
SD-00203A

*Figure 1-1. AOS Hardware Configuration*

End of Chapter

# Chapter 1 Introduction

AOS (Advanced Operating System) is a general-purpose, disk-based operating system that controls and monitors user program processing on ECLIPSE-line computers. AOS takes responsibility for many program control, input/output and file access functions.

Using AOS, you can manage and control a wide variety of environments. This system is at once a multiuser system suitable for computational processing, and a real-time system suitable for demanding process control applications.

## Features

AOS combines the powerful features of large central computer installations with the low cost and modularity of ECLIPSE minicomputers. Features include:

- Simultaneous support of time sharing, multiple batch streams, and real-time applications.
- Efficient management of system and user resources, based on changes in system environment or performance.
- A complete hardware/software security system, providing file access protection, process integrity and system security.
- A flexible and powerful hierarchic file structure, with a sophisticated and convenient organization for disk files and the ability to reference all peripheral devices as files.
- A multitasking environment, providing a structured method of managing complex processes.
- A full complement of utilities and high-level languages.
- A generalized code- and data-sharing facility, with a software cache feature which increases access speeds.
- Device-independent I/O access procedures that are both powerful and easy to use.
- Accounting procedures at the directory, process and user levels.
- A Command Line Interpreter that gives the console user access to all the sophisticated AOS features, but which is simple enough even for an inexperienced user to run.
- Communications between concurrently-existing processes.
- Support of multidisk files.
- Spooled output to character-oriented peripheral devices.
- Full support of binary synchronous communications between computers running AOS or any other system capable of BISYNC protocol.
- Compatibility with Data General's Hasp II Workstation Emulator (HAMLET) and Remote Job Entry Control Program (RJE80). These allow synchronous communications with other systems which are IBM 360/370 compatible.

Typically, the System Manager assigns execution privileges to individual users. System Managers can assign memory resources, execution priority and file access rights uniquely to each user. As a user, you own all of your own files and may control others' access to them. Likewise, other users will decide whether you may access their files, and in what ways. The System

# Chapter 2 Process Concepts

## Processes, Tasks and Programs

AOS is a multiprogramming system. Each *process* competes with the others for system resources such as memory, I/O devices and disk file space. A variable number of processes can run independently of and concurrently with each other.

AOS protects processes from each other's activities and from interfering with AOS itself.

For example, an ECLIPSE system running AOS might simultaneously support a real time data acquisition program, batch mode operation, and several interactive consoles used for program development. AOS would see the real time program, each batch stream, and each of the developmental programs (e.g., Command Line Interpreter, Text Editor, compilers) as independent processes; in this case, each process operates without knowledge of the others, as if it had exclusive use of the central processor.

A *program* occupies each process's memory space. A program is an executable memory image. Typically it is produced by compiling one or more source files to produce one or more object files, which the binder utility then links to produce the executable program. A program, the static entity, contains code paths executed by the tasks, the dynamic entities.

A *task* is an asynchronous flow of code execution through a process's logical address space. Each task has its own Task Control Block, which is a block of data maintained by the operating system with a memory image of the CPU registers and other context data for each task. Since each task has its own program counter, several tasks can appear to run simultaneously, either through a section of re-entrant code (i.e., code which is not modified during execution), or through their own unique code paths. What in fact occurs is that the system schedules tasks and allows them to run on the basis of their states and priorities.

For example, in the system mentioned above, the real time program might consist of a number of tasks, each monitoring a single input line. Other tasks might take the data and write it to a disk file. Each of the monitoring tasks would be able to share the same re-entrant code and execute when data was available on the line it was monitoring.

## Process Types

A process can be resident, preemptible, or swappable. Process priority and type govern when and for how long a process acquires main memory space; Figure 2-1 illustrates this.

*Resident* processes always remain in main memory. You might want to make resident the line interface process in a communications system, or the alarm process in a real-time process control system.

A *preemptible* process resides in main memory, but is deprived of main memory and swapped out to disk if it becomes blocked (described later) and any other process needs memory, or if a higher priority resident or preemptible process needs memory.

For example, if you had to complete a text editing job in a hurry and there were many other users on your system who were engaged in less important activities, you would want to make your process preemptible because you would need a fast response time at your console but real-time considerations would not be critical.

*Swappable* processes receive main memory only after higher types have received what they need. AOS can write swappable processes out to disk according to the system's heuristic policy, which the system derives from factors such as the numerical priority you assigned the process and its past behavior.

Most processes on a heavily used system will be swappable, to allow the operating system to be maximally efficient in distributing its resources equally between users.

## Process Priority

AOS allocates central processor time and memory according to several parameters, one of which is process priority. Priority gives a fine gradation of control in resource management. Resident and preemptible processes have priorities from 1 (the highest) through 255 (the lowest). They are treated as groups, which means a high priority preemptible process will be scheduled ahead of a low priority resident process. Swappable processes have priorities 1, 2 or 3 (high, normal, low).

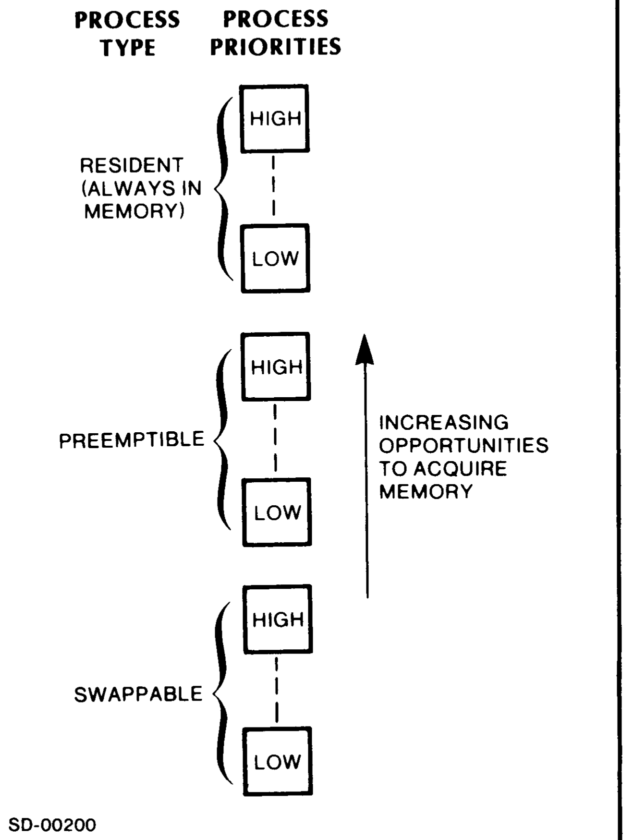


*Figure 2-1. Process Types and Priorities*

## Process States

At any time, every process will be *eligible* to gain CPU control, if it has been allocated main memory and is ready to run, or *ineligible* to gain CPU control, if it is ready to execute but has not been allocated main memory. A *blocked* process is waiting for a specific external event (which might never occur). A resident process that is blocked stays in main memory. If any other type of process becomes blocked, it becomes ineligible, and may be swapped to disk. Each newly-created process is first ineligible. See Figure 2-2.

Assigning process types and priorities structures the execution of operations, since the system gives central processor control to processes in the following organized way: among processes that are eligible to run, it gives control first to the highest priority resident or preemptible process; if no process of either type is eligible, it gives control to the highest priority swappable process.

## Process Hierarchy and Privileges

By assigning privileges and taking advantage of the hierarchy of process relationships, you gain still another dimension of control over process execution.

When AOS is first brought into execution, it creates a logical node called the root. From the root, AOS creates system processes (for example, PMGR, the peripheral process that manages character device I/O) and the initial user process. From this initial process, the console operator can create other processes of any type, at any priority. Also, the console operator uses the initial process to create other system processes (such as the EXEC) to handle special user needs.

All processes existing at any time are related in a tree-like structure. Figure 2-3 shows one possible process "tree".

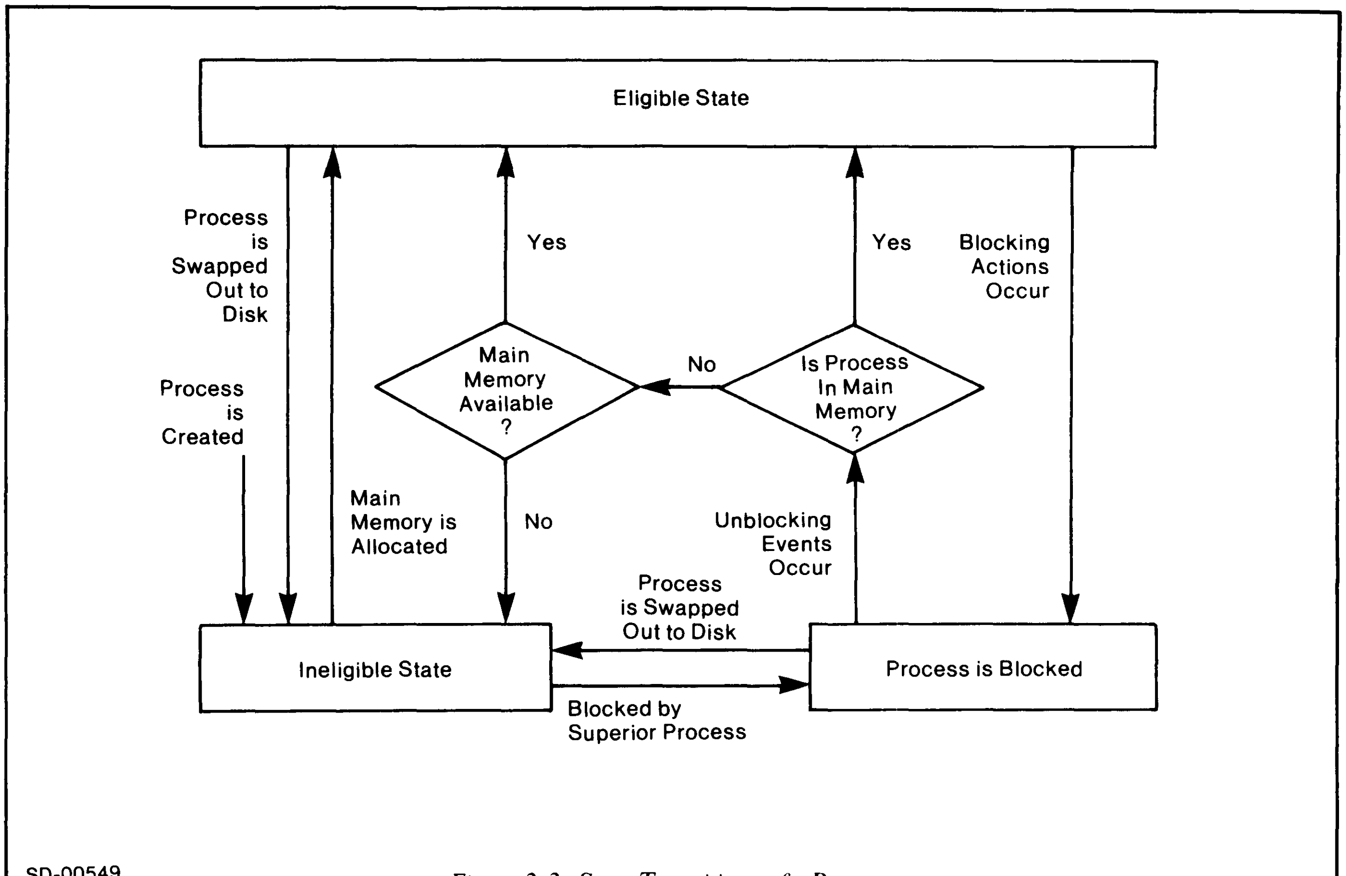


*Figure 2-2. State Transitions of a Process*

A process can be created:

- by your own process directly, when you key a console command to the Command Line Interpreter (CLI is the primary interface between the console and the system); or
- by another process (either a system or user process).

A process under another process is subordinate to that process. In Figure 2-3, processes D, E, F, G, H, I, and J are all subordinate to A, the superior process. Any process can block or unblock a subordinate process, and any properly privileged process can change its own priority or type and the priority or type of any of its subordinates.

The creator (father) of a process can assign privileges to its sons; one of these is the privilege allowing the sons to create their own sons. An offspring process may be privileged to create processes subordinate to itself.

A process with special privileges can enter Superprocess mode, in which it may change the state of any other process, or Superuser mode, in which it may access any user's files.

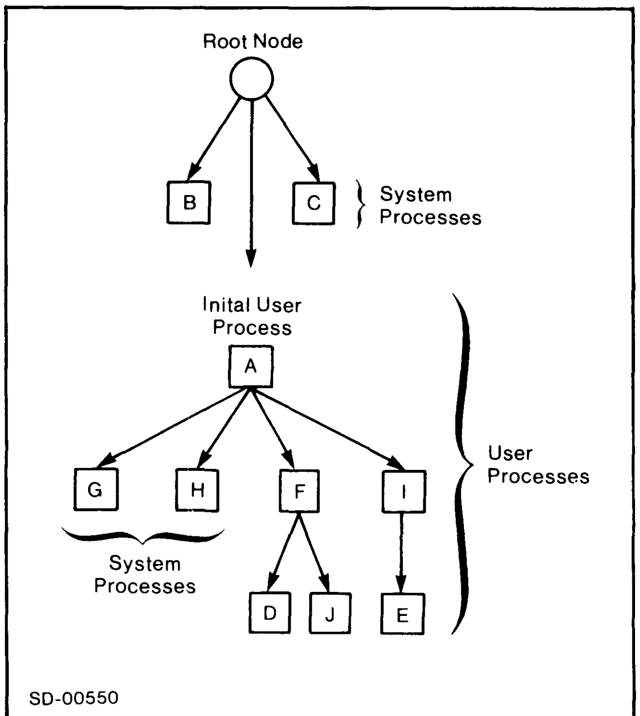
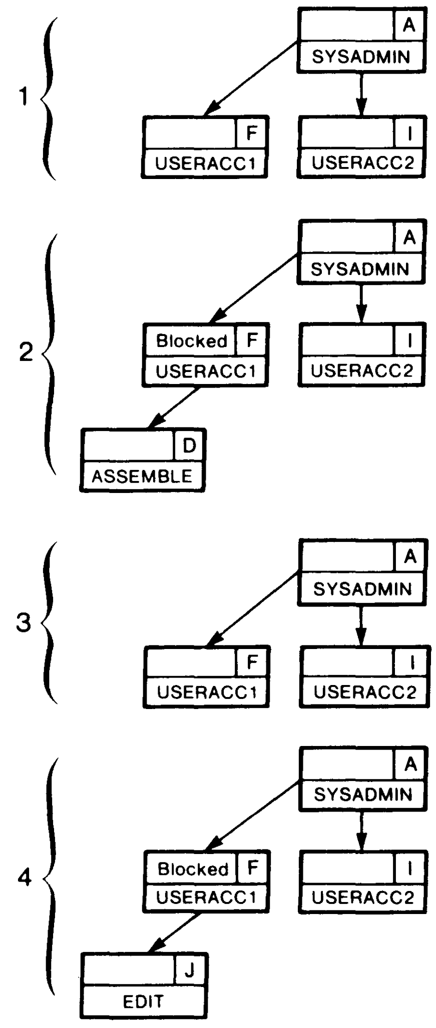


*Figure 2-3. Tree Structure of Process Relationships*

Figure 2-4 illustrates a common practical application of the privilege and hierarchy features shown in Figure 2-3. Process A, the system administrator process, created processes F and I to allow two users to access the system. To assemble a program, process F created a subordinate process, D, and became blocked until the assembly in D completed. Process F then created subordinate process J to correct assembly errors by re-editing the source program file.



SD-00202

*Figure 2-4. Sequence of Process Relationships*

End of Chapter

# Chapter 3
# Memory Management

## Memory Management Overview

AOS provides a flexible and efficient means for utilizing main memory using the Memory Allocation and Protection feature (MAP), the hardware memory-mapping scheme for the ECLIPSE computer. The MAP feature allows you to reference a maximum of 512K bytes of main memory (1 megabyte for the M/600), organized in fixed pages of 2048 8-bit bytes. Your logical address space need not be contiguous in physical memory.

When each process is created, one of the parameters specified is the maximum number of pages of main memory available to it. Each process can be dynamically allocated up to 32 memory pages, or about 65,000 bytes, of user address space. This space includes all user code and data.

In addition, AOS provides certain functions which need not occupy user address space. For example, in the system's address space are modules for processing all system calls, debuggers, and each process's task scheduler. AOS apportions total main memory (see Figure 3-1) dynamically between itself and user processes. System address space varies with the amount of user process activity, such as open channels and active processes, and with the amount of free (unused) memory available.

Process type, blocking actions, shared page operations, completion of process execution, and calls for more memory dynamically interact to change the composition of main memory.

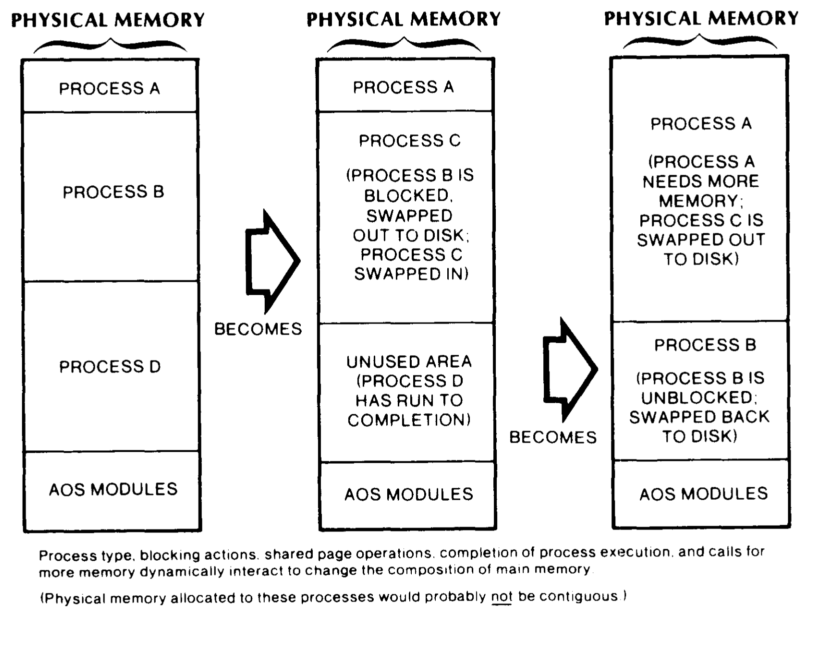

PHYSICAL MEMORY
PROCESS A
PROCESS B
PROCESS D
AOS MODULES
BECOMES
PHYSICAL MEMORY
PROCESS A
PROCESS C
(PROCESS B IS BLOCKED, SWAPPED OUT TO DISK; PROCESS C SWAPPED IN)
UNUSED AREA (PROCESS D HAS RUN TO COMPLETION)
AOS MODULES
BECOMES
PHYSICAL MEMORY
PROCESS A
(PROCESS A NEEDS MORE MEMORY; PROCESS C IS SWAPPED OUT TO DISK)
PROCESS B
(PROCESS B IS UNBLOCKED; SWAPPED BACK TO DISK)
AOS MODULES
Process type, blocking actions, shared page operations, completion of process execution, and calls for more memory dynamically interact to change the composition of main memory
(Physical memory allocated to these processes would probably not be contiguous.)
SD-00205

*Figure 3-1. Dynamic Main Memory Changes*

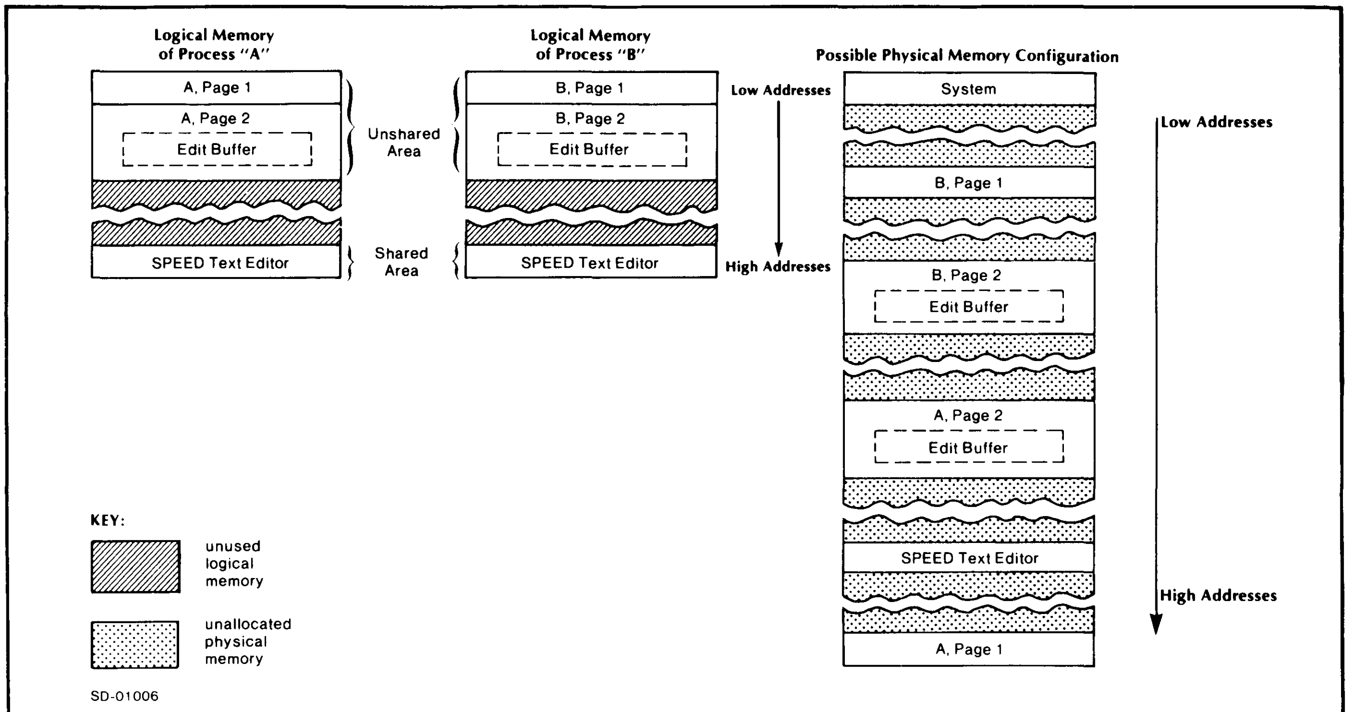


*Figure 3-2. Process Sharing a Commonly-Used Utility, the SPEED Text Editor*

## Shared and Unshared User Areas

A process's logical address space (user address space) may contain both *shared* and *unshared* pages. Unshared pages contain information usable by only one process; shared pages contain information that all processes can share according to their access privileges. In this way you can substantially reduce memory requirements for individual processes. For example, all processes running the text editor program concurrently are said to contain this program's code in their *shared* areas, but only one *physical* copy of the shared part of the editor exists in main memory, no matter how many processes are sharing it. (Each process contains a unique edit buffer in its *unshared* area.) See Figure 3-2. Note that the code shared by the processes in this figure subtracts from the user address space limit for each process (the maximum is 64K bytes).

At any instant, a process's address space can contain up to 32 pages of main memory. (However, a process can call in many *more* than 32 pages of code or data over a period of time by taking advantage of AOS's overlay facility and shared library facility, as explained later.) A process's address space must contain at least one unshared page. AOS uses part of each unshared user area for certain system tables, the shared library list, and an overlay directory.

Each process can define its shared and unshared areas dynamically.

## Shared and Unshared Access Protection

A process has exclusive rights to the information in its unshared area (except for the part used by the system, as mentioned above). Access to information in shared areas is assigned in the form of privileges (to read, write, etc.; these privileges are explained in detail later). Using these privileges, the system enforces security, making it impossible for one user process to violate another's protected files.

## Shared Page Operation

The system applies an algorithm to ensure optimum use of each shared memory page. When a shared page is not currently in demand, it is written to disk only if it was modified and if the system needs memory for other purposes; the system maintains a list of least-recently-used pages for this purpose.

The page-sharing feature makes random access of a disk file easier. Since a frequently-used shared page tends to remain in main memory, the system need not perform disk I/O each time the page is called. Thus, this feature reduces disk I/O traffic.

The system coordinates all of these complex memory allocation activities invisibly. You need not be concerned whether the data you request is being read by another user's process, whether the data is in the least-recently-used list and can be mapped quickly into your address space, or whether the data must be brought into main memory from secondary storage.

## Shared Routines and Overlays

When user address space is not sufficient to contain all code and data necessary for your process, you need some form of program read-in scheme to bring programs in on demand from disk storage. Along with the shared-page mechanism, AOS provides three ways to modify user address space dynamically: with shared routines, with relocatable overlays and with nonrelocatable overlays.

Shared routines are dynamically relocatable modules stored in a disk library of overlays called a shared library. AOS can read shared routines into main memory, then map them into the shared areas of different processes (see Figure 3-3). Shared libraries are built with the shared library builder utility.

When reading *overlays* into shared space, you can take advantage of the system's shared page mechanism to reduce disk swapping. AOS will attempt to retain in main memory those overlays you recently referenced (and may need again soon) but are not currently using. Shared overlays are shared among users of a single program. Shared routines can be shared among users of different programs.

AOS can load a nonrelocatable overlay from disk into a single specified area; and can load a relocatable overlay into any one of a set of specified areas.

## Calling Overlays and Shared Routines

AOS lets a process dynamically bring programs into main memory from disk in two ways.

First, if you want to bring in additional programs or parts of a program as shared routines or overlays, you can do so without coding these resources in any special way. AOS's binder utility links the object files that make up your program to build the executable program file and optional overlay file. In your command to call the binder, you can control whether a resource is to come into main memory as a shared routine, as a relocatable overlay, or as a nonrelocatable overlay. By changing this command, you can change the way any program will be brought into the user address space.

Second, to bring an overlay or shared routine into main memory, you need to code only a simple call. The AOS *load-on-call* mechanism determines whether the resource called is an overlay or a routine in a shared library, locates the resource on disk, and loads it into the user address space. This load-on-call mechanism gives you an easy way to bring more code and data into main memory than the user address space can contain at one time.

## Shared Routines

Shared routines can be used by different programs. For example, if you build a shared routine to calculate cube roots, programs that perform this function can share the routine and save main memory.

Shared routines must be reentrant to be shareable, and must be relocatable, since the system will try to load them into any available shared routine area (see Figure 3-3). To call a shared routine, a program need only provide its name; AOS locates the shared library that contains the routine with no further specification necessary. A routine in a shared library can be executed independently by all eligible processes.

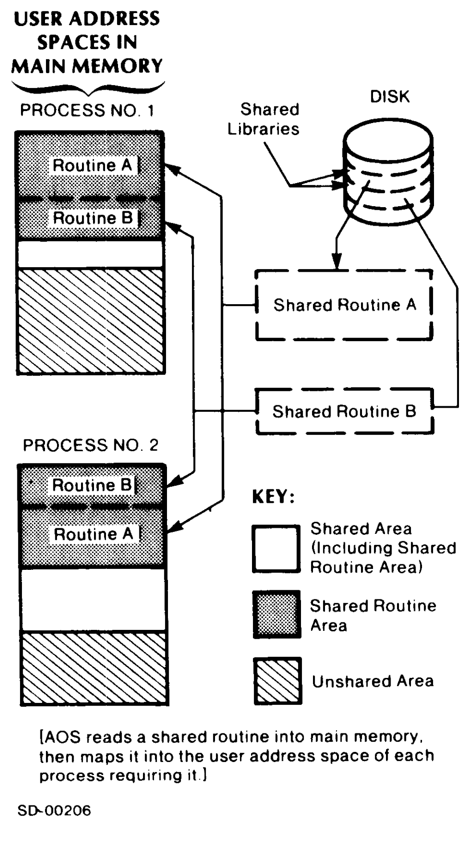

USER ADDRESS SPACES IN MAIN MEMORY
PROCESS NO. 1
Routine A
Routine B
Shared Libraries
DISK
Shared Routine A
Shared Routine B
PROCESS NO. 2
Routine B
Routine A
KEY:
Shared Area (Including Shared Routine Area)
Shared Routine Area
Unshared Area
[AOS reads a shared routine into main memory, then maps it into the user address space of each process requiring it.]
SD-00206

*Figure 3-3. Shared Routine Operation*

## Overlay Concepts

AOS can reserve parts of shared and unshared user address storage space, and divide these parts into fixed-length storage areas. This lets you segment a large program into portions that will fit into these fixed-size areas at execution time. These program segments are called *overlays* in AOS. The fixed-length storage areas in main memory are called *overlay areas.* Several overlays can occupy the same overlay area at different times. This enhances efficient use of main memory by allowing you to execute more code than will fit in your user address space at one time.

Overlays reside in a disk overlay file, which has the same name as the program file. A system call reads an overlay into its reserved area in memory. If an overlay is shared, only one physical copy of it is read into main memory, and all processes sharing the overlay use that copy.

User address space can contain up to 64 overlay areas. Up to 511 overlays can "belong" to an overlay area; they are read into the area one by one.

Overlay areas can be shared or unshared. Unshared overlay areas and their corresponding overlays on disk are allocated and built in multiples of 512 bytes. Shared overlay areas and their associated disk overlays are built in multiples of 2048 bytes (i.e., in page units).

When your program calls an overlay, the system first determines whether or not the desired overlay already resides in an overlay area; only if it does not, or if the call specifies an unconditional load, will AOS load the overlay from disk. This conditional loading eliminates unnecessary disk traffic.

The overlay operation does not interfere with other system processing. A program loads overlays one by one into their fixed memory area. After the program uses an overlay, any new overlay (for that area) overwrites a previous one. The program itself remains active and memory-resident while it loads them. You can execute program segments in one or more currently-resident overlays while AOS loads another overlay. AOS itself uses a relocatable overlay operation to enhance its own processing.

### Relocatable Overlays

Relocatable overlays are overlays that can be executed anywhere in main memory. The overlay code must be written so that it does not depend on being executed at a particular location in memory. You can associate several relocatable overlays with an area large enough to hold more than one of them at the same time. For example, if you need a total of ten overlays, but a maximum of only three at one time, you can associate all ten with an area in main memory three times as large as the largest overlay.

Overlay areas associated with relocatable overlays can be loaded into shared or unshared portions of the user address space.

### Nonrelocatable Overlays

You can also read nonrelocatable overlays into shared or unshared memory space. The system ensures that each overlay area in a program will fit the largest overlay loaded into it. Consequently, overlays intended for one overlay area can be of different sizes. Nonrelocatable overlays *can* be written to execute at a particular location in main memory. If a nonrelocatable overlay is *shared,* AOS loads it into the same overlay area for each process sharing it (see Figure 3-4).

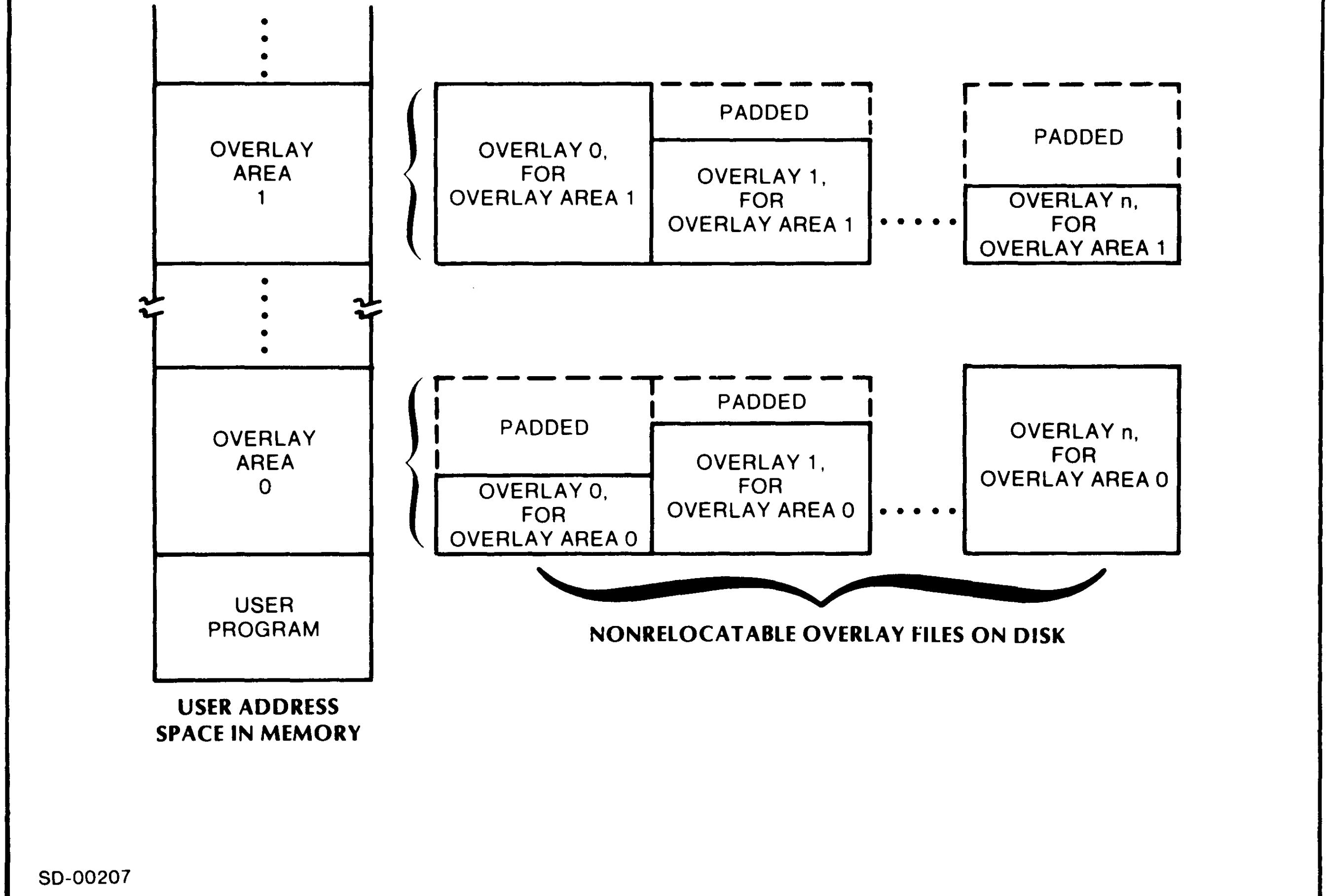

OVERLAY AREA 1
OVERLAY 0, FOR OVERLAY AREA 1
PADDED
OVERLAY 1, FOR OVERLAY AREA 1
PADDED
OVERLAY n, FOR OVERLAY AREA 1
OVERLAY AREA 0
PADDED
OVERLAY 0, FOR OVERLAY AREA 0
PADDED
OVERLAY 1, FOR OVERLAY AREA 0
OVERLAY n, FOR OVERLAY AREA 0
USER PROGRAM
NONRELOCATABLE OVERLAY FILES ON DISK
USER ADDRESS SPACE IN MEMORY
SD-00207

*Figure 3-4. Nonrelocatable Overlay Structure*

End of Chapter

# Chapter 4
# File Structures

A file is a collection of related data treated as a unit; files are stored on devices (i.e., hardware peripherals). Each file has a name (called a filename) which allows both you and the system to address it. Names assigned to files are independent of the device used.

Certain peripheral devices, like the line printer and consoles, can be treated as files; for instance, in a program you can access the line printer like a file on disk. This device independence permits you to refer to files in programs without needing to specify the device until the program is run.

Some devices, such as magnetic tape and disk units, permit both storage and retrieval of multiple files. AOS supports both 7- and 9-track magnetic tape units and Data General disk units, including diskettes.

Magnetic tape units and disk units read or write files in any order, although the serial nature of magnetic tape reels makes random access slower and file writing less flexible than on disk. Multifile devices, and disk units in particular, have the flexibility which permits AOS to construct its hierarchical file organization.

## Disk File Structures

Every disk file is built from one or more 512-byte disk blocks. The system organizes disk blocks into files with a hierarchical indexing method. The basic unit of storage in this file organization is a file element, consisting of one or more contiguous disk blocks (i.e., blocks with physical disk addresses in an unbroken series). This file element size is specified when a file is first created.

Data file elements can be any number of disk blocks up to 8 megablocks, or the size of the largest disk, whichever is smaller. Since file space is allocated in units of the file element size, if a file with a file element size of 8 grows, it grows in units of eight blocks. A disk file can be up to $2^{32}$ (4 billion) bytes in length.

Figure 4-1 shows a data file created with an element size of two blocks. To add another element, AOS allocates a disk block as an index for locating the old and new data element. An index block contains the addresses of each of the data elements.

As data storage requirements continue to grow and exhaust all space in the index, AOS adds another level of indexing. This sophisticated file growth scheme allows files to expand without requiring physically contiguous storage.

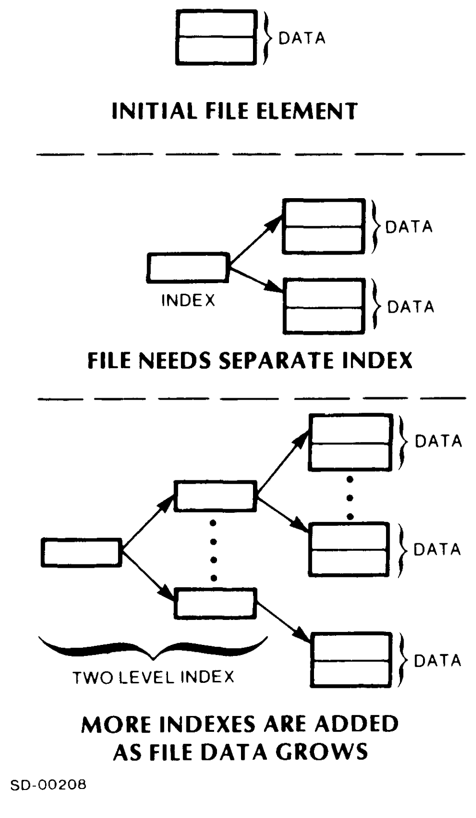


*Figure 4-1. Stages in Disk File Growth*

## Directory Structures

A directory is a file that catalogs information about other files; through it, qualified users can easily access files.

All directories are connected in a network resembling an inverted tree. The top of this network is the system root directory, and all other directories are inferior to the system root. Each directory except the system root is pointed to by an entry in some higher directory. Higher directories are called superior; directories lower in the "tree" hierarchy than others are called inferior. The system root contains entries for directories and for certain files used by the operating system; it also contains a directory called the User Directory Directory (UDD), which holds an initial directory for each user (see Figure 4-2).

## Directory Entries

Directory file data is organized into entries of many different types. You can define up to 128 entry types, and Data General defines up to 128 additional types. The major distinction among the types is that some entries point to files, and the rest do not.

Each directory contains entries pointing to the files associated with it. These files may be other directories immediately lower on the directory "tree"; or they may be user data files. Examples of user data files are source files (files produced by a text editor program or the CLI CREATE command), object files (files which you have assembled or compiled) and executable program files (files output by the binder utility).

A directory entry that points to a file contains the file's name, its access control information, and the data necessary to access it.

Directory entries that do not point to files include peripheral device entries, interprocess communications (IPC) entries, link entries, and unit entries (explained later). Each of these entries contains a name and other information unique to that type of entry.

## Disk Space Control

Either you or your System Manager can regulate use of disk space by designating certain directories as control points. This mechanism limits the total size of files below the control point directory to a maximum number of disk blocks. This maximum is specified when the control point directory is designated.

## Access Control

A strong access protection scheme prevents unauthorized use or alteration of data through the directory structure. Privileges extended to users or groups of users are attached to each directory entry when created, and you may change these during the life of the entry. You can assign up to five different types of access to each directory entry; their use differs slightly according to the type of entry to which they apply. The five types are: execute access, read access, append access, write access, and owner access.

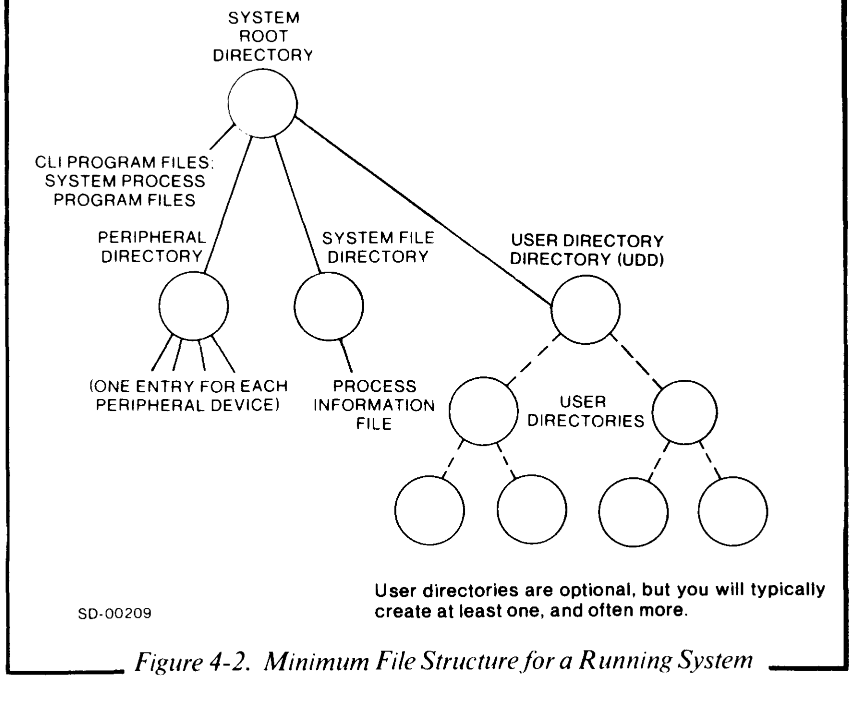

SYSTEM ROOT DIRECTORY
CLI PROGRAM FILES; SYSTEM PROCESS PROGRAM FILES
PERIPHERAL DIRECTORY
SYSTEM FILE DIRECTORY
USER DIRECTORY DIRECTORY (UDD)
(ONE ENTRY FOR EACH PERIPHERAL DEVICE)
PROCESS INFORMATION FILE
USER DIRECTORIES
SD-00209
User directories are optional, but you will typically create at least one, and often more.

*Figure 4-2. Minimum File Structure for a Running System*

*For directory entries:* Execute access permits using a directory's name in a pathname; read access permits listing a directory's entries and the status of files pointed to by the entries; append access permits adding entries to a directory; write access permits adding and deleting entries and altering access control information (for subordinate directories and files); and owner access permits changing the directory's access control list and deleting the directory. Note that without execute access, you cannot utilize any other type of access to a directory, since you must use a directory's name in a pathname in order to do anything with it.

*For entries other than directories:* Execute access allows execution of a program file; read access permits examining data to which the entry points; write access permits changing existing data. Append access has no meaning; owner access permits changing the file's access control list or deleting the file.

The system maintains an access control list for each entry, which lists names of users or groups of users who may access a file or unit, as well as the types of access to which they are entitled.

The following examples illustrate how the file access protection scheme can help you:

- A class of students uses a computer system as part of their study. The professor can assign each student an independent work space, and limit access privileges to that area. By assuming all access privileges, the professor can execute, examine, or modify any student's program/data.
- A company stores employee information in a large database. Nearly unlimited access is allowed to the names of employees, but access to descriptions of their jobs is much more restricted, and only a few users are privileged to access salary information.

## Access Through Directories

At any time, a process has a *working directory,* which is the directory to which the process currently directs its attention. You can name either the system root directory, the peripheral directory, or the working directory as a reference point when you access an entry. Alternatively, you can specify a list, called a *search list,* of other directories to search if the system cannot find the named entry in the working directory.

## Pathnames and Search Lists

A pathname points to a desired entry in the directory hierarchy. You access any entry with a pathname, using directories as reference points.

One method of file access uses the working directory and the search list. If an entry is within the working directory, you need specify no directory when you access the entry. If the entry is not in the working directory, the system looks for it in the search list. The system scans entries in the search list in the order you specify.

Alternatively, you can give an absolute pathname to access the entry. This pathname has an optional prefix and/or one or more directory names linked by colons; these prefixes and names identify the directories in the path the system traces to find the entry. If specified, the prefix denotes the directory at which the system begins its search for the entry. Using a prefix prevents the system from scanning a search list.

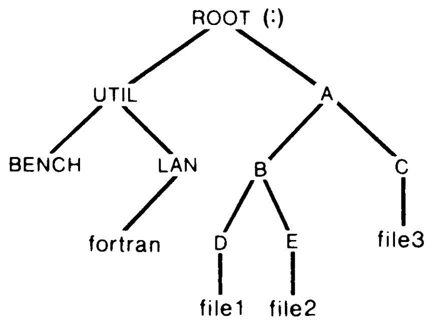
There are four pathname prefixes:

| **Prefix** | **Meaning** |
|---|---|
| : | Start the search at the system root directory. |
| @ | Start the search at the peripheral directory. |
| = | Start the search at the working directory. |
| ↑ | Start the search at the immediately superior directory. (You can use more than one ↑ in a prefix if the pathname ascends through more than one superior directory.) |

As a pathname that points to another directory, a prefix alone can be a valid pathname ( ↑ ). Otherwise, you must follow a prefix with at least one filename (↑FILEA). To access an entry in another directory, follow a prefix with the name of the entry. If you give the system a pathname with *no* prefix, and the entry is not in the working directory, the system will search for the entry in the directories listed in the *search list.*

Figure 4-3 depicts a sample directory "tree" structure and search list, and gives a series of sample pathnames.

**DIRECTORY STRUCTURE**



**SEARCH LIST**

:A:B:E
:UTIL:LAN (LANGUAGE DIRECTORY)
:UTIL (UTILITY DIRECTORY)

**KEY:**
UPPERCASE = directories
lowercase = files

| WORKING DIRECTORY | PATHNAME | SYSTEM ACTION |
|---|---|---|
| A | =B:D:file1 | Moves from directory A to directory B to directory D, then finds "file1." If it did not find file1 in D, it would not have scanned the search list (= inhibits this) |
| A | file2 | Searches for "file2" in A, not found, so scans directories in search list and finds "file2" in directory E |
| E | ↑D:file1 | Moves from E to B, then to D and finds "file1" |
| E | fortran | Searches for "fortran" in E; not found, so scans directories in search list and finds "fortran" in LAN |
| UTIL | :A:B:D:file2 | Starts at ROOT, then moves to A, B, and D. Does not find file2 and returns an error condition code |
| D | ↑↑C:file3 | Moves up to B, then to A, then down to C, to find "file3" |

SD-00258

*Figure 4-3. Searching a Directory Structure by Pathnames*

## Link Entries

Link entries provide a special flexibility in directory access. The link has a name which you can use in a pathname, or treat as a pathname itself. When you use the link name in a pathname, the contents of the link are substituted for it. Thus, a link name can be used as an abbreviation for a long pathname with its prefixes.

## Forming File Space on Physical Disks

AOS supports several types of disk units, which can be intermixed in any system. Before performing file I/O on any physical disk in the system, you must first process the disk with the Disk Formatter utility. This utility builds a logical disk (LD), a grouping of one or more physical disks the system treats as a single logical entity (see Figure 4-4). Each logical disk is a file hierarchy or free-standing tree structure. Each LD has a local root; the LD containing the system root directory is the master LD. You can graft one or more LDs to the master LD, and graft other LDs to those. The file hierarchy of the new LD becomes part of the existing file structure, and its root node becomes subordinate to the file system to which it is grafted.

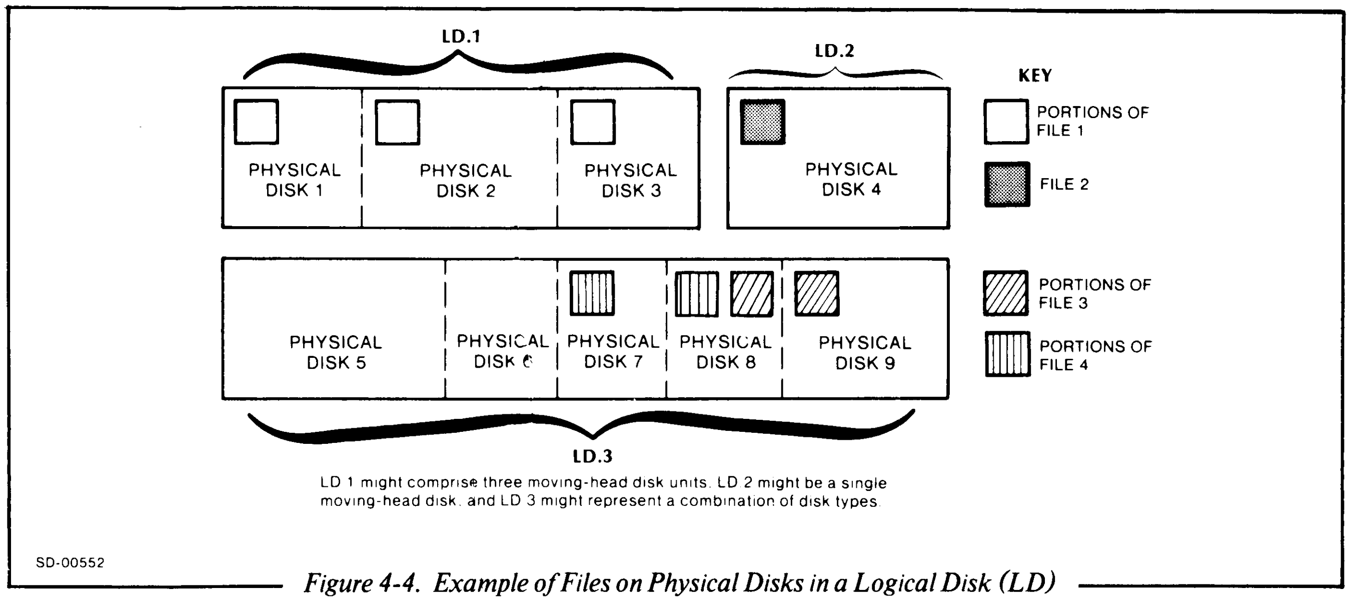


*Figure 4-4. Example of Files on Physical Disks in a Logical Disk (LD)*

## Selecting the System File Hierarchy

For the system to treat the logical disks as a directory hierarchy, you must first initialize each LD. Once the LD is initialized, AOS "sees" all physical disks in an LD as a collection of files related in a directory structure. If you open a disk without initializing it, the system treats it as a single free-standing file consisting of a single contiguous storage space.

## Magnetic Tape File Access

Each reel of tape (called a volume) contains records organized into files, one after the other, from the start of the tape to its end. Tape files are specified simply by their relative position within a volume (e.g., file number 0, 1, 2, etc.). The CLI command MOUNT not only tells the operator to mount a tape volume for you, but also creates a link entry for you so that you may refer to the tape unit by the link name. You can access tape files using standard I/O procedures. Access control applies to the entire tape unit.

## Labeled Magnetic Tape Access

AOS has implemented ANSI standards 1, 2 and 3 and IBM levels 1 and 2 for labeled magnetic tape. With the LABEL utility you can prepare an ordinary volume of tape for this use. The advantages of labeled tape are numerous:

- The labels on each file keep track of the date a file is created, its retention time and other information.
- When you use the CLI MOUNT command, you must provide the unique volume identifier for the tape volume you want mounted. This prevents other users from erasing your files accidentally. Also, as in unlabeled tape, the system creates a link for you at this time.
- You can refer to files on a labeled tape even if the tape is not on a tape drive, and the system will issue a MOUNT command for you. Likewise, when you close a file on a labeled tape that was implicitly mounted, the system will issue the DISMOUNT command for you.
- The SCRATCH switch on the LABEL utility allows you to effectively delete all the files on a labeled tape volume.

## Multiprocessor Communications Adapter Access

The multiprocessor communications adapter (MCA) permits communication between two Data General central processors using the processors' data channels. Transferring MCA data is very similar to transferring a block of information on a magnetic tape file, except the MCA "file" number uniquely identifies a specific central processor link. The MCA protocol is compatible with other Data General operating systems.

End of Chapter

# Chapter 5
# File I/O

AOS provides you with a flexible system for accessing information from disk and magnetic tape, and from character-oriented peripheral devices. Previous chapters have described the ways in which disk and tape files are organized. This chapter describes how to store and retrieve information in the files.

## Peripheral Device Support

AOS supports a wide variety of devices:

- moving-head disks (including diskettes)
- fixed head disks
- magnetic tape units
- line printers
- synchronous and asynchronous communications lines
- multiprocessor communications adapter (MCA)
- paper tape readers and punches
- card readers
- digital plotters

When you initialize the system, AOS writes the names of available devices (including each magnetic tape and disk unit) into a system directory called the peripheral directory (:PER). These standard device names are not reserved exclusively for these devices; you can create your own files with these names.

## I/O Methods

To execute an I/O transfer, you generally:

1. Open the file
2. Read or write the file data
3. Close the file.

You can read data from, and write data to the standard devices and magnetic tape or disk files in two ways: by blocks, or by logical groupings called records.

### Record I/O

Record I/O uses any of four record types on any device or file. These types are: dynamic, fixed-length, data-sensitive, and variable-length. (See Figure 5-1.)

You specify the length of a *dynamic record* (in 8-bit bytes) when it is read or written. A *fixed-length record* has a constant length. A *data-sensitive record* is a series of characters terminated by a character defined to be a delimiter. Several different characters can be delimiters, and you can dynamically change a delimiter definition. A *variable-length record* has a fixed-length header describing the number of bytes of data that it contains.

You can specify the record type when creating a file; and you can specify or change the record type and I/O buffer when opening it, and performing I/O operations.

## Block I/O

Block I/O permits you to transfer information in multiples of physical blocks. This feature is available with disk units, disk files, magnetic tape units, tape files, MCA links, and all other noncharacter devices and, where meaningful, individual files on those devices. A disk unit block is 512 bytes of physical storage space; magnetic tape blocks are data groupings separated by physical interrecord gaps; an MCA block is the data group sent or received in a single transmission, and can be any length up to 8192 bytes. (See Figure 5-1.)

As mentioned in Chapter 4, you can perform I/O on entire magnetic tape or disk units as freestanding files by opening them without first initializing them.

To access magnetic tape blocks, specify the numerical position of a file on tape, the number of the starting block, and the number of blocks to be transferred. When you open a disk without initializing it, you can access all blocks on it, including those not accessible to you on an initialized unit. You can open MCA links in the same way as tape files. You can open a specific MCA link, or specify a zero link to allow the system to read from any central processor.

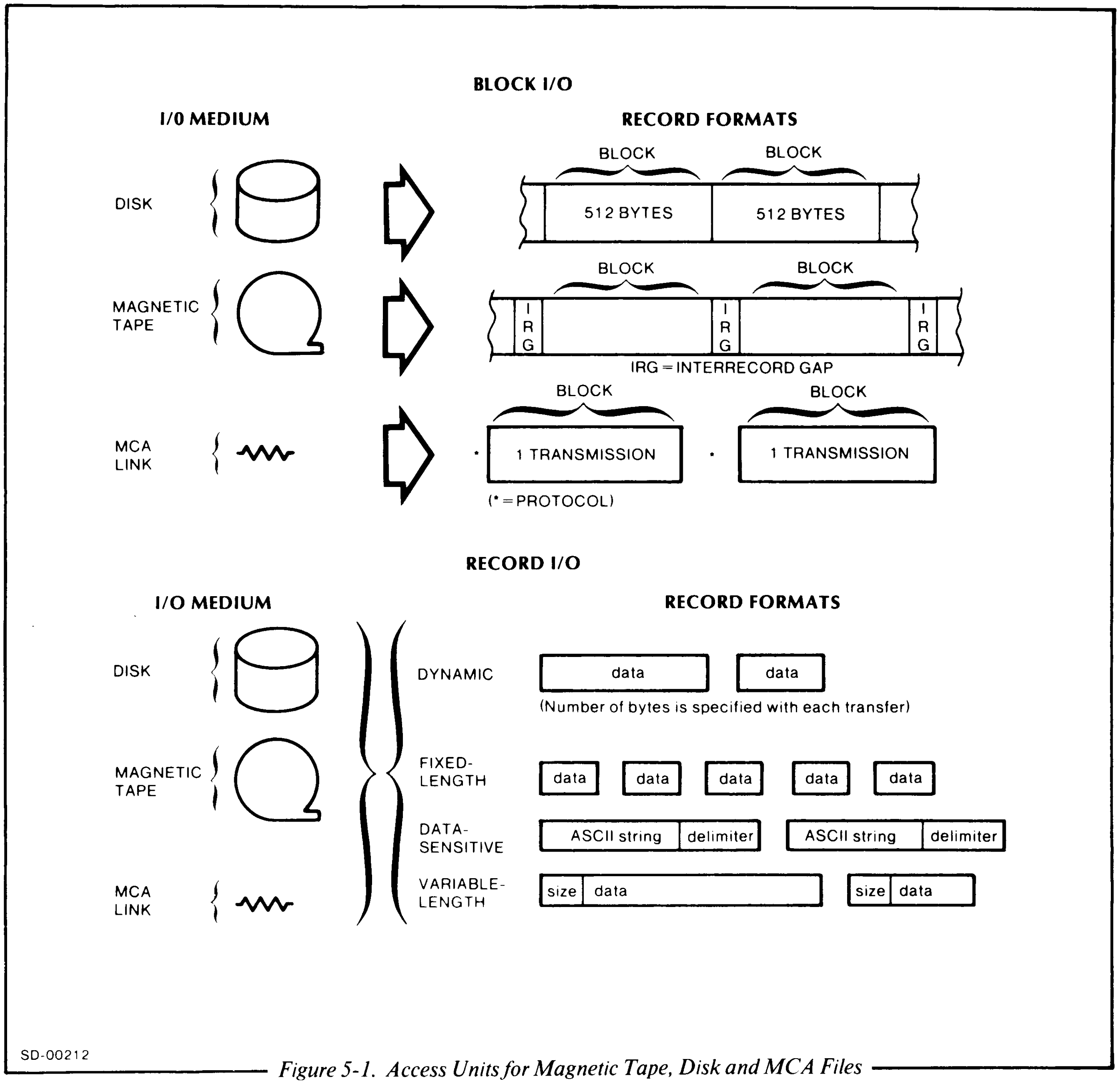


*Figure 5-1. Access Units for Magnetic Tape, Disk and MCA Files*

## Spooling Files for Output

To use central processor time effectively, efficient I/O handling is essential. In a multiple-user environment, many processes typically compete for a relatively slow output device. With a mechanism called spooling, AOS ensures that processes sending output to line printers, paper tape punch devices and asynchronous communications lines receive the service they need in a orderly way.

The spooling mechanism diverts output intended for a slow-output device to a temporary disk file. Each device has a spool queue containing the names of files intended for the device.

You can approach spooling in two ways:

- You can issue a CLI command (e.g., ENQUEUE, QPRINT, etc.) which enqueues an existing file to an output device. The system then places the filename in the spool queue for the specified device, and the file proceeds normally through the queue.
- Alternatively, you can open a spoolable device directly. This creates a temporary file on disk, to which AOS directs output data. When you close the file, AOS places it in the appropriate device's spool queue. When output processing is complete, the system deletes the file from disk.

## Screen Management

When you are at a console using the LINEDIT Text Editor, AOS interprets certain control characters in special ways. You may also use the same control characters for screen management while reading data from a terminal by selecting this option on the appropriate I/O system call.

## Communications Support

AOS provides easy access to synchronous and asynchronous communications lines for intercomputer communications. The software currently supports the following:

1. General-purpose protocol for asynchronous lines, and automatic spooling.
2. BISYNC protocol supported by primitive system calls in AOS.
3. Compatibility with Data General's Remote Job Entry Control Program (RJE80), which permits point-to-point communication with any IBM 360/370 compatible system, including an RJE80- or HAMLET-equipped Data General system.
4. Compatibility with Data General's HASP II Workstation Emulator (HAMLET), which permits point-to-point communications with any IBM 360/370 compatible system, including an RJE80- or HAMLET-equipped Data General system.

End of Chapter

# Chapter 6
# Creating and Managing a Multitask Process

## Task Concepts

A *task* is the basic program unit that uses system resources (including CPU control); it executes asynchronously from and independently of other tasks. The *program,* the current executable contents of a process's address space, contains the code paths executed by tasks.

A single-task program has one unified path connecting all its program logic, no matter how complex the branches, and one task, called the *initial task,* which follows this code path. A process need not be limited to performing one task, however. AOS allows up to 32 tasks in each process. Many tasks can be directed to run in a single code path, provided that path is re-entrant, i.e., it is not modified during execution. Alternatively, each task may follow a unique code path. AOS can preserve and restore the state of each task as required (see Figure 6-1). The state information is called the task's *context,* and the operating system stores it in a unique *Task Control Block* for each task.

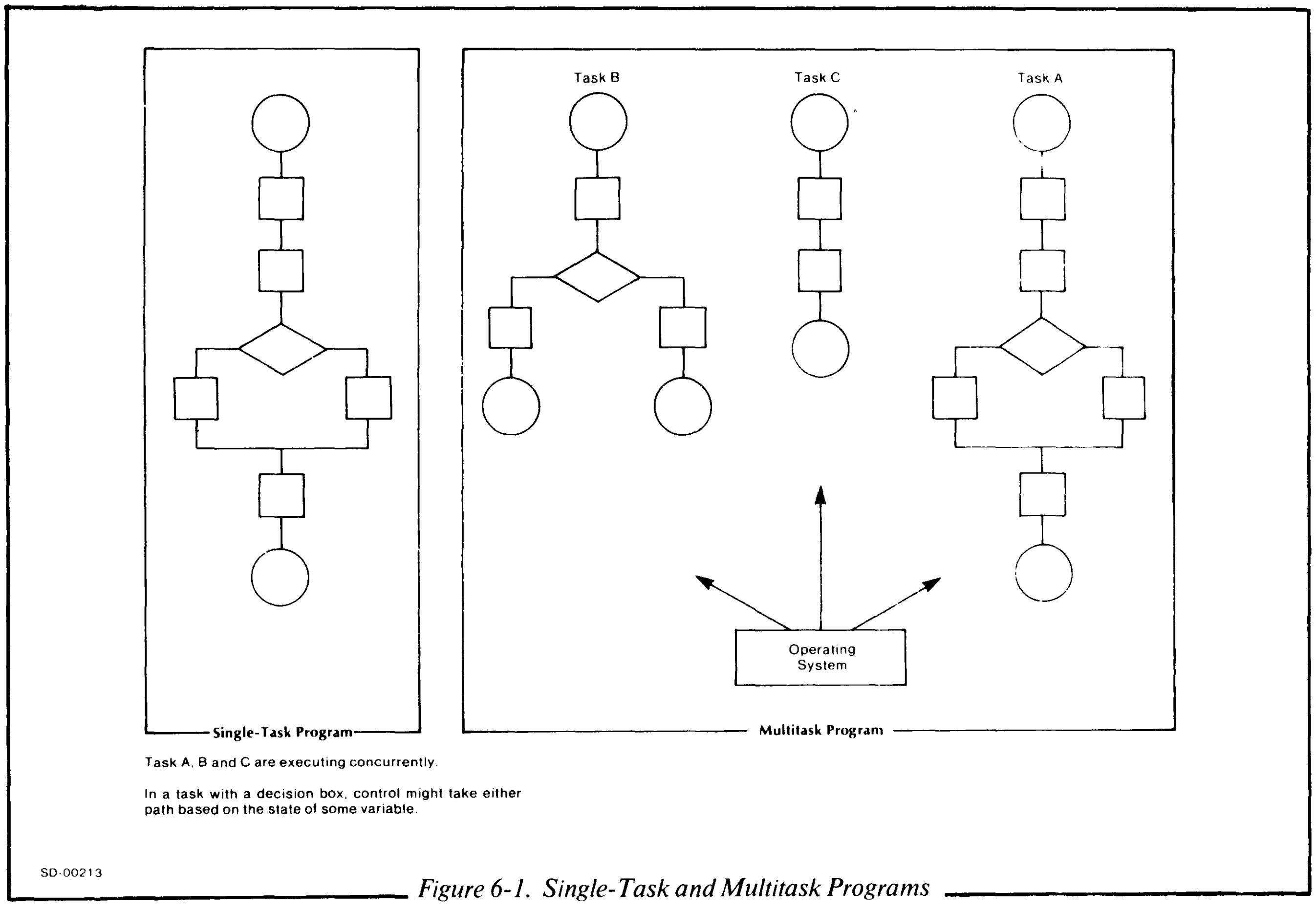

Task B
Task C
Task A
Operating System
Single-Task Program
Multitask Program
Task A, B and C are executing concurrently.
In a task with a decision box, control might take either path based on the state of some variable.
SD-00213

*Figure 6-1. Single-Task and Multitask Programs*

AOS supports *structured multitasking:* priority assignments made to single tasks or groups of tasks ensure that only the highest priority task ready for execution will execute. A unique identification number permits each task to be specified without ambiguity.

Some potential uses of multitask processes are airline reservation systems, inventory control systems, process control operations, and communications networks with message queuing and switching.

## Task Priorities

The system creates an initial task for each user program. An appropriate task call from the initial task will initiate more tasks; each of these can in turn initiate others.

Like processes, each task is assigned a priority when first initiated; priorities range from 0 (the highest) through 255 (the lowest). Priorities can be changed during the execution of the program. If more than one task exists at the same priority level, AOS gives each task on that level control of the central processor in round-robin fashion when tasks at that priority are executing.

## Task States

Tasks can exist in any of four states (see Figure 6-2):

- When you create a program you specify the number of tasks in it. One of these tasks is the program's initial task; the remaining tasks, as yet uninitiated, are in the *dormant* state.

- When the initial task in the program initiates the second or subsequent tasks, the task scheduler executes the highest priority task within the process. Other tasks that have been initiated but are of lower priority than the executing task are *ready* to run.

- At any moment, only one task in one process is *executing* on the central processing unit (CPU). The system's task scheduler gives CPU control to the highest priority ready task in the process to be given CPU control. (See "Process States" in Chapter 2.)

- The executing task becomes *suspended* when it makes a call to the operating system. The task remains suspended until the call is completed, at which time it again becomes ready. Certain task calls can also suspend an executing task.

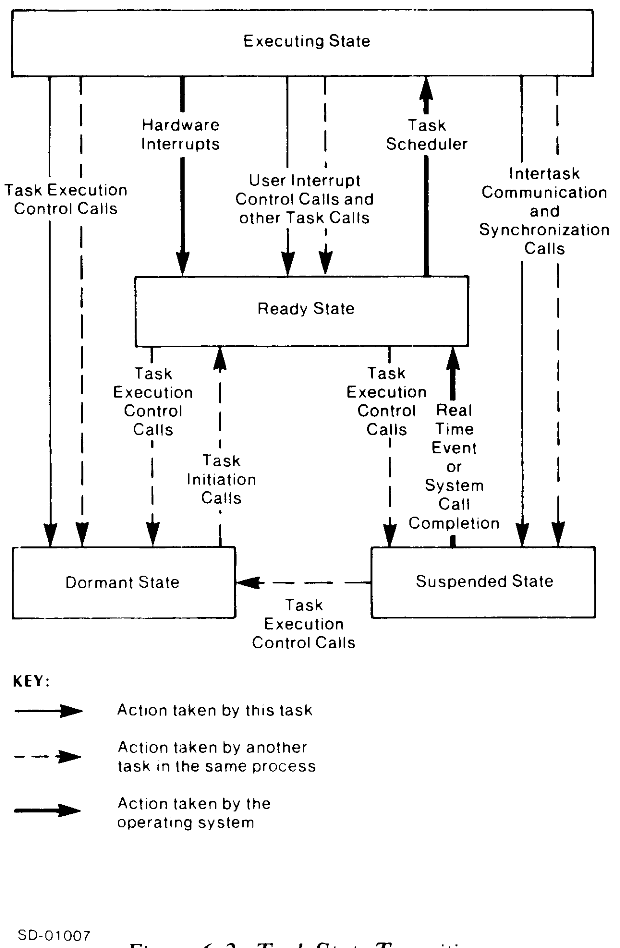

Executing State
Hardware Interrupts
Task Scheduler
Task Execution Control Calls
User Interrupt Control Calls and other Task Calls
Intertask Communication and Synchronization Calls
Ready State
Task Execution Control Calls
Task Initiation Calls
Task Execution Control Calls
Real Time Event or System Call Completion
Dormant State
Suspended State
Task Execution Control Calls
KEY:
Action taken by this task
Action taken by another task in the same process
Action taken by the operating system
SD-01007

*Figure 6-2. Task State Transitions*

## Task Identification and Control

To enable your program to differentiate further among tasks, the system permits you to assign each task a unique identification number. This gives an added dimension of clarity to task management, since many tasks can have the same priority and can even change priorities, but each task's I.D. remains both unique and constant.

The task monitor maintains certain status information about each active task (i.e., executing, ready, or suspended) to make sure the executing task is always the highest priority ready task. This status information is kept in each task's Task Control Block.

## Task Management Calls

AOS offers a complete set of task management calls to control a dynamically varying environment. Task calls originate in the source program, and are of four kinds:

- task initiation
- task execution control
- intertask communication and synchronization
- task timing control

### Task Initiation Calls

AOS automatically initiates the first task in a program, i.e., raises it from the dormant to the ready state. To create a multitask environment this task must issue at least the first task initiation call. After more than one task is running, any task can initiate other tasks.

### Task Execution Control Calls

Task execution control calls allow you to exercise precise control over individual tasks or groups of tasks. You can suspend, ready, or terminate tasks in a process by specifying their unique identification numbers, or by specifying a priority.

### Intertask Communication and Synchronization Calls

Tasks must communicate with each other to control a multitask environment effectively. Tasks can send messages to other tasks and halt their own activity, if desired, until the receiving task accepts the transmitted message. User interrupt service routines can also send messages, even though AOS suspends multitask activity while servicing user devices.

### Task Timing Control Calls

A task can suspend itself for a period of time which it specifies. AOS also maintains a system clock and calendar for tasks that must be scheduled on a time-of-day basis. Tasks can examine the frequency of the system clock and can obtain and set the date and the correct time.

End of Chapter

# Chapter 7
# Interprocess Communications

AOS lets processes intercommunicate and synchronize their activities. With this feature, called the interprocess communications facility or IPC, processes can send free-format messages of any length.

You can use IPC on two levels. If you wish to perform sophisticated process synchronization and intercommunication, you can use the IPC "primitive" calls. Otherwise, you may access IPC "ports" as files with normal file I/O system calls.

## Ports

Processes send and receive IPC messages between ports. A *port* is a full-duplex communications path to a process, and each process can have 127 of these. A unique 32-bit port number identifies each port; you can also give each port a name which other processes use to obtain the port number.

When one process transmits a message to another via the IPC facility, it issues a send request to the system, specifying the location of an IPC header. The header is a parameter block which gives information pertinent to the message transfer, such as the origin and destination port numbers and the length of the message. The sender specifies the destination port -- the port to which the message is going -- and the origin port -- the port from which the message is being sent.

Potential message receivers issue a complementary receive request which also specifies the location of an IPC header. If the origin and destination port numbers in the two headers match, then AOS passes a message between the two processes. The IPC facility also permits a process to send a message to itself for testing or other purposes.

## Port Matching Rules

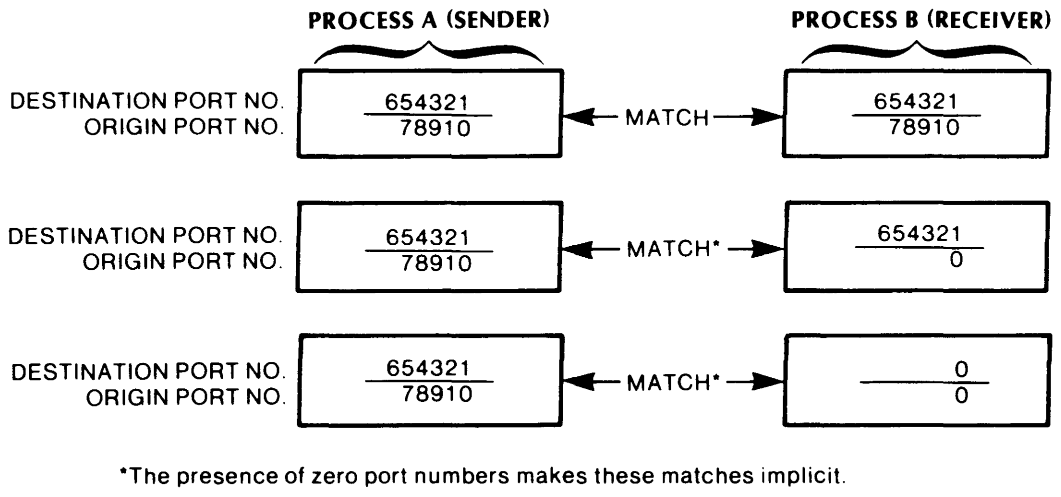
Port number matching can be either explicit or implicit (see Figure 7-1). An explicit match occurs when the origin and destination ports specified by both sender and receiver are the same. An implicit match of port numbers occurs when one or both port numbers specified in the receive header are zero.

If the receiver's origin port number is zero, the process can receive a message from any sender. If the receiver's destination port number is zero, then the process can receive the message on any port number assigned to it.

After each transmission, AOS copies the port numbers actually used to send and receive the message into the receiver's header. If desired, messages can consist merely of header information passed from sender to receiver.

## Message Spooling

Tasks sending a message regain control of the CPU immediately, enabling them to continue activity without delay. If no receiver is waiting for the message, the system spools the message, writing it out to disk until an appropriate receive call is issued. However, the system suspends a task that issues a receive request until some task issues a matching send call; then it moves the data and readies the receiver task.



SD-00551

*Figure 7-1. Port Number Matching*

End of Chapter

# Chapter 8
# Utilities

You can develop your programs under AOS concurrently with other users. AOS supports a full complement of utilities to let you create, edit, assemble, execute, debug, compile, and manipulate files.

## Language Support

AOS provides the following high-level languages to fill the needs of many different user applications:

- Extended BASIC
- FORTRAN IV and FORTRAN 5
- COBOL
- RPG II (Report Program Generator)
- DG/L
- PL/I
- INFOS

Extended BASIC is suitable for educational and scientific applications.

FORTRAN IV and FORTRAN 5 come with a series of real-time and file control extensions for monitoring and controlling real-world environments.

COBOL implements ANSI '74 specifications at the highest levels, and includes additional features such as a sort/merge facility, extra data types, and the ability to call assembly language routines in a COBOL program. Data General COBOL is also compatible with the INFOS file system, and greatly simplifies access to and management of INFOS files.

Data General's RPG II is an easy-to-use high-level language for report generation, file maintenance and data preparation. It is a full implementation of IBM's Report Program Generator. Data General's RPG II includes an editor and program analyzer, and is fully compatible with INFOS software.

DG/L is an ALGOL-derived structured programming language designed for a wide range of sophisticated applications. It features a comprehensive operating system interface, a multitasking runtime environment, memory management, arithmetic and mathematical library functions, string handling, and string arithmetic. The DG/L compiler globally optimizes user source code, and produces object code that is re-entrant, recursive and position independent, and may run on any Data General ECLIPSE, NOVA or microNOVA under AOS, RDOS, RTOS, or DOS.

PL/I is a structured high-level language which contains the features of earlier languages such as FORTRAN, COBOL, and ALGOL. It is, therefore, well-suited for scientific, commercial, and systems programming applications.

The INFOS® file system lets users create, maintain and use large databases. The system supports the conventional sequential and random access methods as well as an advanced indexed sequential access method. It also provides a sophisticated Data Base Access Method (DBAM) which minimizes data redundancy and database reorganization. INFOS advanced data handling capabilities make it suitable for numerous applications in inventory, production, sales, credit, insurance, purchasing, and distribution management.

## System Utilities

A set of system utilities completes the collection of program development tools. These are:

- the Command-Line Interpreter (CLI), the primary interface between the console or batch programmer and the system
- two text editors, which let you conveniently edit text files
- the Macroassembler, which provides macro capabilities and produces object files from source programs written in assembly language
- the binder, which links object files to produce an executable program file
- a symbolic program debugger
- library builder programs which let you easily create, edit and maintain relocatable binary program and shared routine libraries.

In addition to these, you can use a log-on system utility to control user access to the system. This utility creates a process for each user logging onto the system, according to the privileges assigned by the System Manager, and monitors users' activities. You can also use the accounting facility to maintain user-level accounting information, and a special program for examining this information in an intelligible format.

End of Chapter

# Chapter 9
# Command Line Interpreter

The Command Line Interpreter (CLI) lets you perform file and process maintenance at your console terminal, or in batch mode. The CLI can also invoke other system utilities or user programs.

Specifically, some of the operations CLI performs are: program execution; file creation, maintenance and backup; directory creation and maintenance; and spooling control.

One of the CLI's great advantages is its flexibility. Some CLI functions do little more than execute one system call (such as DELETE "filename"), while others perform very complex operations. Depending on your needs, you can choose CLI commands ranging from a few simple ones to the large number available for more complex processing.

## Command Format

Each command you give CLI from the console has a simple format: a name with optional switches, which you can then follow with an argument list with optional switches.

A number of other CLI features (summarized below) enhance its versatility.

## Template Expansion

One of CLI's most useful features is its ability to perform functions on all files matching a certain template. A template is one or more filename characters (the ones to be matched) and one or more expansion operators; these are expanded into a list of all file names matching the template. For example, the CLI expands

```
QPRINT +.SR
```

to include all files in the current directory which end in ".SR" (the expansion operator here is "+"). This might result in the expanded command:

```
QPRINT AA.SR,FILEB.SR,JOHNSFILE.SR
```

## Command and Argument Iteration

The CLI can execute commands and arguments repeatedly until it exhausts a list of arguments. This lets you subject a list of elements (files, for example) to an operation in sequence. Thus,

```
XEQ MASM (TEST1,TEST2,TEST3)
```

executes the macroassembler utility (MASM) three times, once for each argument. If you used a template in this iteration list, you could execute the same operation with the command

```
XEQ MASM ([!FILENAMES TEST*])
```

("*" is a single-character expansion operator, and !FILENAMES is a CLI pseudomacro, as explained below). Combining templates and iteration lists is often very useful.

## Macros

You can group CLI input text on disk in the form of *macros.* A macro is accessible using a single filename. When you issue a command line containing a macro call, the contents of the file on disk replace the macro call. CLI macros have a very powerful "dummy argument" format that allows you to modify a macro's function by calling the macro with values that replace the "dummy" arguments. Replacement can be based on either argument position or keyword.

## Pseudomacros

CLI pseudomacros are similar to the standard macro routines described above, but they are written permanently into the CLI. One kind of pseudomacro provides conditional expansion control. The other pseudomacros expand to certain information commonly used in AOS programming, such as the name of the process's current working directory or a certain group of filenames (!FILENAMES in the previous example is a pseudomacro).

## Help Facility

The CLI has a general-purpose HELP facility; you can use it to find out what CLI commands are available, how any given command executes, how certain general CLI features operate (such as templates and pseudomacros), the location of any file for which you have proper access, and what other HELP commands are available.

End of Chapter

# Chapter 10
# System Generation

Your System Manager can tailor AOS to the specific hardware configuration and software requirements of your installation, and can modify the system as your installation's requirements change.

This tailoring function is system generation. The System Manager calls a system generation program (AOSGEN) which asks questions about the hardware and software configuration of the installation. If the manager requests it, AOSGEN builds an AOS system with devices and options tailored to your installation.

AOS offers the System Manager considerable flexibility in specifying a system. For example, only the necessary device driver modules need be included in the system. Also, the modular design of AOS and numerous system features make it possible to configure multiple systems, with each system tailored for a specific hardware configuration.

Data General provides a standard AOS system (the "starter system") which will run on any hardware configuration satisfying minimum licensing requirements. You can use this system to generate a system tailored for your installation.

End of Chapter

# Glossary

| Term | Meaning |
|---|---|
| access control list | A system-maintained list for each file and directory (e.g., logical disk), containing the names of users who may access that file or directory and the types of access to which they are entitled. |
| address space | See logical address space. |
| block | A logical grouping of contiguous words of code in main memory or on a peripheral storage device. Except for disk blocks, the size may vary. |
| blocked process | One of three process states, in which a process is waiting for a specific external event to occur so that it can gain control of the central processor. A process can block itself or become blocked involuntarily. |
| block I/O | One of two input/output modes in which you can access a file. Information is transferred in 512-byte disk blocks, magnetic tape blocks, or MCA blocks.<br><br>The system always performs I/O in block units, whether you employ block or record I/O. |
| CLI (command line interpreter) | An AOS utility which is the main interface between the system and you at the console or in batch mode. CLI accepts your command lines and (among other functions) translates this input into commands for other utilities, or commands which directly perform functions such as file maintenance. |
| device | A hardware peripheral component; each type of device has unique operating characteristics. Devices are either character-oriented (send or receive single bytes of data) or block-oriented (send or receive data in multibyte blocks). |
| device independence | The ability of a process to communicate with a device without regard to the unique nature of the device. |
| directory | A file that catalogs files, and allows qualified users to access them. Directories are connected in a network resembling an inverted tree, with lower directories inferior to higher directories. Each directory contains an entry for any directory immediately inferior to it. |
| directory entry | A unit of information contained in a directory; a directory can contain multiple entries. A common type of entry is that which lists certain information about a file in the directory (see file status). Examples of other types of entries are IPC entries and links. |
| disk | A magnetic recording medium (e.g., disk pack, disk cartridge, diskette, fixed-head disk). |
| disk address | The location of a block (sector) on a disk. |
| disk block | The smallest allocatable unit of disk memory, standardized as 512 bytes. |

| Term | Meaning |
|---|---|
| disk controller | A mechanism that directs the operator of one or more disk units. A program can direct the operation of a disk controller. |
| disk controller name | The name of a disk controller, consisting of three letters and possibly one decimal digit; for example: DPE and DPE1. |
| disk drive | Same as disk unit. |
| disk unit | A mechanism which physically reads from and writes to a disk. |
| disk unit name | The name of a disk unit, consisting of the name of a disk controller followed by a decimal digit; for example: DPE0 and DPE10. |
| dormant state | One of four task states, in which a task exists that has not yet been initiated (made known to the oeprating system) or that has terminated execution. |
| eligible process | One of three process states, in which a process has been allocated main memory, and competes for control of the central processor with other such processes, based on its process type and priority. |
| executable file | A binary memory-image file which you can read into main memory from a peripheral storage device for execution; a program capable of being run. |
| executing state | One of four task states, in which a task has control of the CPU. Only one task at a time can be executing. |
| file | A collection of related data treated as a unit. A file can contain up to 2**32 bytes of data. Disk and magnetic tape can contain one or more files. |
| file access privilege | The basis of the AOS file access protection feature. You can be assigned up to four types of file access: execute, read, write and owner access. Access to *directories* includes these types, plus an append access privilege. |
| file element | The basic unit of storage in the AOS disk file organization. Each file element consists of one or more contiguous blocks. You specify file element size when a file is first created. If a file grows, it grows in units of this file element size. |
| filename | An alphanumeric file identifier. All filenames in a single directory must be unique, and each can contain no more than 31 characters. |
| file status | The collection of information about each file which AOS maintains in that file's entry in its immediately superior directory. This information includes the file size, time of creation, and other details. |
| hierarchical file system | AOS organizes files in, and accesses them through directories connected in an inverted tree structure. The highest directory in the hierarchy is the root, which points to inferior directories; these in turn point to directories inferior to them. Any process with proper privileges can access any file within any directory. |
| ineligible process | One of three process states, in which a process has not been allocated main memory but in all other ways is ready to run. |
| initial task | When there is only one task executing in a process, that task is called the initial task. |
| interprocess communication facility (IPC) | A generalized AOS facility which sends free-format messages of any length between any two processes. IPC messages are sent between ports (see port). |
| K | An abbreviation for the number 1024 decimal. Thus 32K bytes of memory are 32,768 bytes. |
| link | A type of directory entry which points to another entry in the directory hierarchy. When you use a link in a pathname, the contents of the link are substituted for its name. |

| Term | Meaning |
|---|---|
| logical disk address | The location of a logical block on a logical disk. The address must include a disk pointer and a disk address to access the block. |
| logical address space | The set of memory locations (up to 64K bytes) that one process can access at any time. The physical memory locations constituting the logical address space are not necessarily contiguous, and the process can change the logical address space during its execution. |
| logical disk (LD) | A collection of one or more physical disks with a single block allocation map. An LD is a single logical entity, a freestanding "tree" structure of files. In a running system, you can "graft" logical disks onto the master logical disk (on which the system runs) by issuing a CLI command. |
| logical disk name | The filename of a logical disk's root directory. |
| main memory (physical) | Core or semiconductor storage which contains computer instructions or data. |
| MCA | Multiprocessor communications adaptor; i.e., a device which permits communication between two Data General central processors using the processors' data channels. |
| multi-programming | The ability to run an arbitrary number of independent processes. The system allocates its resources among these processes based on their priorities, types, or certain software events. |
| multitasking process | A process in which more than one task is currently active. |
| object code | Code, consisting of 16-bit instruction words and data words, which has been assembled or compiled from a source code file but not yet bound with other modules by the Binder utility to make an executable program. |

| Term | Meaning |
|---|---|
| object code file | A file containing object code, usually created by the Macroassembler or one of several high-level language compilers and having a filename ending in ".OB". |
| overlay | A segment of a larger program which may be brought into main memory when needed. |
| overlay area | A fixed-length storage area in a program into which different overlays may be read at different times during a program's execution. |
| overlaying | A technique for bringing blocks of code or data (called overlays) into a program's main memory area from disk storage during program execution. This technique permits several routines to occupy the same main memory locations at different times. |
| page | Memory storage area of 2K (2048) bytes, starting on a 2K-byte boundary. |
| pathname | A name that identifies the location of a file within the system's file hierarchy. A pathname may be a filename, or an optional list of directories followed by the file's name. |
| physical disk | Same as disk. |
| port | A data path to or from a process. Interprocess messages are sent between ports, which are full-duplex and can therefore send and receive data simulataneously. Each port is assigned a unique number (see IPC). |
| port numbers | The identification mechanism that allows two processes to send and receive messages via the IPC facility. The system maintains a directory of process numbers and associated port numbers. |
| preemptible process | One of three process types. AOS writes a preemptible process to disk if it becomes blocked, or if a higher priority preemptible process or any resident process requires more memory. |

| Term | Meaning |
|---|---|
| priority | A number assigned to each task and each process, ranging from 1, the highest, through 255 (except for swappable processes, which can have priorities from 0 to 3). If more than one process or task has the same priority, the system gives each process or task at that level CPU control in round robin fashion when that priority becomes the executing one. Priorities can be changed during program execution.<br><br>*process priority* - one of the factors that govern how the system allocates CPU time to a process. More than one process can have the same priority.<br><br>*task priority* - governs which is the executing task within a process. The executing task is always the highest priority task ready to run in the process with control of the central processor. |
| process | A collection of one or more tasks sharing the same logical address space and competing as a unit for system resources, e.g., memory, I/O devices, file space, and processor time. |
| process states | Every process in a system will be in one of three states at any moment; eligible, ineligible, or blocked. |
| process type | Process type governs when and for how long a process acquires main memory space. The three process types are: resident, preemptible, and swappable. |
| program | The current executable contents of a process's address space. A program contains the code paths executed by tasks. A process contains only one program at any given time; but during the execution of a process the current program may change many times. |
| program file | A file containing an executable program, ready to be loaded into a process's logical address space. Usually it is created by the Binder utility and has a filename ending in ".PR". |
| ready state | One of four task states, in which a task is available for execution while a higher priority task has control of the CPU, or while another process is running. More than one task can be ready at any time (see priority). |
| resident process | A process which always remains in main memory, i.e., is never swapped out to disk; the highest of the three process types. |
| root directory | The highest directory in a logical directory unit. The system's root directory is superior to all other directories in the system, and contains entries for system and user directories and files. |
| search list | The list of directories to be searched when you give the system the name of a file that is not in the working (current) directory. Each process has its own search list. |
| shared library | See shared routine facility. |
| shared pages | The facility whereby AOS can read all or portions of disk files into main memory pages for sharing among processes. |
| shared routine facility | The facility whereby AOS implicitly calls one or more library routines on disk into main memory areas in page increments; processes share these if they are privileged to access routines from that library. |
| source code | Code, consisting of byte-packed words of ASCII characters, which can be converted by an assembler or compiler into object code. Such code is usually composed by a user. |
| source code file | A file containing source code, usually created by a user with the CLI or a text editor. |

| Term | Meaning |
|---|---|
| swappable process | One of three process types. A swappable process receives main memory only when both higher types (resident and preemptible) have received what they need. The system scheduler writes the swappable process out to disk at its discretion. Each swappable process receives main memory and processing time according to factors such as user-assigned priority and process behavior in the past. |
| swapping | AOS swaps a process when it writes it out to disk, then reassigns the main memory occupied by that process to another process waiting to be run. This reading from and writing to disk (swapping in and out) is invisible to the process. Swapping optimizes use of system resources. |
| system generation | When you generate AOS, you tailor it to the particular hardware configuration and application environment at your installation. |
| system call | A request to the operating system to act on your behalf, subject to privilege checks. |
| task | An asynchronous flow of code execution through a process's logical address space. |
| task call | Same as system call. |
| task control block | A block of data maintained by the operating system for each task, containing a memory image of the CPU registers and other context data for each task. |
| task states | A task in a process exists in one of four states: dormant, ready, suspended or executing. |

| Term | Meaning |
|---|---|
| template | Certain characters to be matched, plus one or more expansion operator characters which allow specified parts of the template to accept any character as legal.<br><br>1. A feature of the Command Line Interpreter utility, which performs a function on one or more files when you specify the function and a filename template. (See the *CLI User's Manual.)*<br><br>2. A feature of the access control list mechanism. Access privileges can be assigned to user groups whose individual names satisfy a template. (See the *AOS Operator's Guide.)* |
| timesharing | A multiprogramming system in which processes share the central processor on a timed basis; i.e., a process takes control of the CPU for a unit of time called a time slice. At the end of this time, other processes waiting for control are given it, so no process monopolizes the CPU. |
| unshared pages | Main memory areas, built in increments of 2K bytes, used by only one process. |
| word | A 16-bit (two-byte) location of memory. All Data General instruction words (except extended instructions) and memory locations are this standard unit size. |
| working directory | A reference point in the system hierarchy to which a process is currently directing its attention; it can be used for pathname resolution when the process is accessing files in a directory structure. |

End of Glossary

# Index

Within this index, the legend "f" following a page number means "and the following page"; "ff" means "and the following pages". "GL" means "Glossary".

CUT ALONG DOTTED LINE

Data General

# users group Installation Membership Form

Name ______ Position ______ Date ______

Company, Organization or School ______

Address ______ City ______ State ______ Zip ______

Telephone: Area Code ______ No. ______ Ext. ______

**1. Account Category**

☐ OEM
☐ End User
☐ System House
☐ Government

**2. Hardware**

| | Qty. Installed | Qty. On Order |
|---|---|---|
| M/600 | | |
| MV/Series ECLIPSE* | | |
| Commercial ECLIPSE | | |
| Scientific ECLIPSE | | |
| Array Processors | | |
| CS Series | | |
| NOVA*4 Family | | |
| Other NOVAs | | |
| microNOVA* Family | | |
| MPT Family | | |

Other ______
(Specify) ______

**3. Software**

| | |
|---|---|
| ☐ AOS | ☐ RDOS |
| ☐ AOS/VS | ☐ DOS |
| ☐ AOS/RT32 | ☐ RTOS |
| ☐ MP/OS | ☐ Other |
| ☐ MP/AOS | |

Specify ______

**4. Languages**

| | |
|---|---|
| ☐ ALGOL | ☐ BASIC |
| ☐ DG/L | ☐ Assembler |
| ☐ COBOL | ☐ FORTRAN 77 |
| ☐ Interactive COBOL | ☐ FORTRAN 5 |
| | ☐ RPG II |
| ☐ PASCAL | ☐ PL/1 |
| ☐ Business BASIC | ☐ APL |
| | ☐ Other |

Specify ______

**5. Mode of Operation**

☐ Batch (Central)
☐ Batch (Via RJE)
☐ On-Line Interactive

**6. Communication**

| | |
|---|---|
| ☐ HASP | ☐ X.25 |
| ☐ HASP II | ☐ SAM |
| ☐ RJE80 | ☐ CAM |
| ☐ RCX 70 | ☐ XODIAC™ |
| ☐ RSTCP | ☐ DG/SNA |
| ☐ 4025 | ☐ 3270 |
| | ☐ Other |

Specify ______

**7. Application Description**

O ______

**8. Purchase**

From whom was your machine(s) purchased?

☐ **Data General Corp.**
☐ Other
Specify ______

**9. Users Group**

Are you interested in joining a special interest or regional Data General Users Group?

O ______

Data General

Data General Corporation, Westboro, Massachusetts 01580, (617) 366-8911

*FOLD* *FOLD*

*TAPE* *TAPE*

*FOLD* *FOLD*

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

DataGeneral

ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581

CUT ALONG DOTTED LINE

Data General

TP__________

# TIPS ORDER FORM
## Technical Information & Publications Service

BILL TO:
COMPANY NAME____________________
ADDRESS____________________
CITY____________________
STATE__________ ZIP__________
ATTN:____________________

SHIP TO: (if different)
COMPANY NAME____________________
ADDRESS____________________
CITY____________________
STATE__________ ZIP__________
ATTN:____________________

| QTY | MODEL # | DESCRIPTION | UNIT PRICE | LINE DISC | TOTAL PRICE |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

(Additional items can be included on second order form) [Minimum order is $50.00]

Tax Exempt #__________
or Sales Tax (if applicable)

| | |
|---|---|
| TOTAL | |
| Sales Tax | |
| Shipping | |
| TOTAL | |

**METHOD OF PAYMENT**

☐ Check or money order enclosed
For orders less than $100.00

☐ Charge my ☐ Visa ☐ MasterCard
Acc't No.__________ Expiration Date__________

☐ Purchase Order Number:__________

**SHIP VIA**

☐ DGC will select best way (U.P.S or Postal)

☐ Other:
☐ U.P.S. Blue Label
☐ Air Freight
☐ Other __________

NOTE: ORDERS LESS THAN $100, INCLUDE $5.00 FOR SHIPPING AND HANDLING.

Person to contact about this order __________ Phone __________ Extension ______

Mail Orders to:

Data General Corporation
Attn: Educational Services/TIPS F019
4400 Computer Drive
Westboro, MA 01580
Tel. (617) 366-8911 ext. 4032

**Buyer's Authorized Signature** Date
(agrees to terms & conditions on reverse side)

Title

DGC Sales Representative (If Known) Badge #

**DISCOUNTS APPLY TO MAIL ORDERS ONLY**

012-1780

educational services

# DATA GENERAL CORPORATION TECHNICAL INFORMATION AND PUBLICATIONS SERVICE TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form shown on the reverse hereof which is accepted by DGC.

1. **PRICES**
   Prices for DGC publications will be as stated in the Educational Services Literature Catalog in effect at the time DGC accepts Buyer's order or as specified on an authorized DGC quotation in force at the time of receipt by DGC of the Order Form shown on the reverse hereof. Prices are exclusive of all excise, sales, use or similar taxes and, therefore are subject to an increase equal in amount to any tax DGC may be required to collect or pay on the sale, license or delivery of the materials provided hereunder.

2. **PAYMENT**
   Terms are net cash on or prior to delivery except where satisfactory open account credit is established, in which case terms are net thirty (30) days from date of invoice.

3. **SHIPMENT**
   Shipment will be made F.O.B. Point of Origin. DGC normally ships either by UPS or U.S. Mail or other appropriate method depending upon weight, unless Customer designates a specific method and/or carrier on the Order Form. In any case, DGC assumes no liability with regard to loss, damage or delay during shipment.

4. **TERM**
   Upon execution by Buyer and acceptance by DGC, this agreement shall continue to remain in effect until terminated by either party upon thirty (30) days prior written notice. It is the intent of the parties to leave this Agreement in effect so that all subsequent orders for DGC publications will be governed by the terms and conditions of this Agreement.

5. **CUSTOMER CERTIFICATION**
   Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub-licensee of the software which is the subject matter of the publication(s) ordered hereunder.

6. **DATA AND PROPRIETARY RIGHTS**
   Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

7. **DISCLAIMER OF WARRANTY**
   DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS SUPPLIED HEREUNDER.

8. **LIMITATIONS OF LIABILITY**
   IN NO EVENT SHALL DGC BE LIABLE FOR (I) ANY COSTS, DAMAGES OR EXPENSES ARISING OUT OF OR IN CONNECTION WITH ANY CLAIM BY ANY PERSON THAT USE OF THE PUBLICATION OF INFORMATION CONTAINED THEREIN INFRINGES ANY COPYRIGHT OR TRADE SECRET RIGHT OR (II) ANY INCIDENTIAL, SPECIAL, DIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOSS OF DATA, PROGRAMS OR LOST PROFITS.

9. **GENERAL**
   A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer.

## DISCOUNT SCHEDULES

## DISCOUNTS APPLY TO MAIL ORDERS ONLY.

## LINE ITEM DISCOUNT

5-14 manuals of the same part number - 20%
15 or more manuals of the same part number - 30%

**DISCOUNTS APPLY TO PRICES SHOWN IN THE CURRENT TIPS CATALOG ONLY.**

Data General

## TIPS ORDERING PROCEDURE:

Technical literature may be ordered through the Customer Education Service's Technical Information and Publications Service (TIPS).

1. Turn to the TIPS Order Form.
2. Fill in the requested information. If you need more space to list the items you are ordering, use an additional form. Transfer the subtotal from any additional sheet to the space marked "subtotal" on the form.
3. Do not forget to include your MAIL ORDER ONLY discount. (See discount schedules on the back of the TIPS Order Form.)
4. Total your order. (MINIMUM ORDER/CHARGE after discounts of $50.00.)

   If your order totals less than 100.00, enclose a certified check or money order for the total (include sales tax, or your tax exempt number, if applicable) plus $5.00 for shipping and handling.
5. Please indicate on the Order Form if you have any special shipping requirements. Unless specified, orders are normally shipped U.P.S.
6. Read carefully the terms and conditions of the TIPS program on the reverse side of the Order Form.
7. Sign on the line provided on the form and enclose with payment. Mail to:

   TIPS
   Educational Services – M.S. F019
   Data General Corporation
   4400 Computer Drive
   Westboro, MA 01580
8. We'll take care of the rest!

educational services

# User Documentation Remarks Form

Your Name ______________________ Your Title ______________________

Company ______________________

Street ______________________

City ______________________ State __________ Zip __________

We wrote this book for you, and we made certain assumptions about who you are and how you would use it. Your comments will help us correct our assumptions and improve the manual. Please take a few minutes to respond. Thank you.

Manual Title ______________________ Manual No. __________

Who are you? ☐EDP Manager ☐Analyst/Programmer ☐Other __________
☐Senior Systems Analyst ☐Operator __________

What programming language(s) do you use? ______________________

How do you use this manual? *(List in order: 1 = Primary Use)* ______________________

___ Introduction to the product ___ Tutorial Text ___ Other
___ Reference ___ Operating Guide __________

About the manual:

| | Yes | Somewhat | No |
|---|---|---|---|
| Is it easy to read? | ☐ | ☐ | ☐ |
| Is it easy to understand? | ☐ | ☐ | ☐ |
| Are the topics logically organized? | ☐ | ☐ | ☐ |
| Is the technical information accurate? | ☐ | ☐ | ☐ |
| Can you easily find what you want? | ☐ | ☐ | ☐ |
| Does it tell you everything you need to know | ☐ | ☐ | ☐ |
| Do the illustrations help you? | ☐ | ☐ | ☐ |

If you have any comments on the software itself, please contact Data General Systems Engineering.
If you wish to order manuals, use the enclosed TIPS Order Form (USA only).

Remarks:

Date

134-664

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

POSTAGE WILL BE PAID BY ADDRESSEE

DataGeneral

**User Documentation, M.S. E-111**
**4400 Computer Drive**
**Westborough, Massachusetts 01581**

Data General Corporation, Westboro, MA 01580

069-000016-01