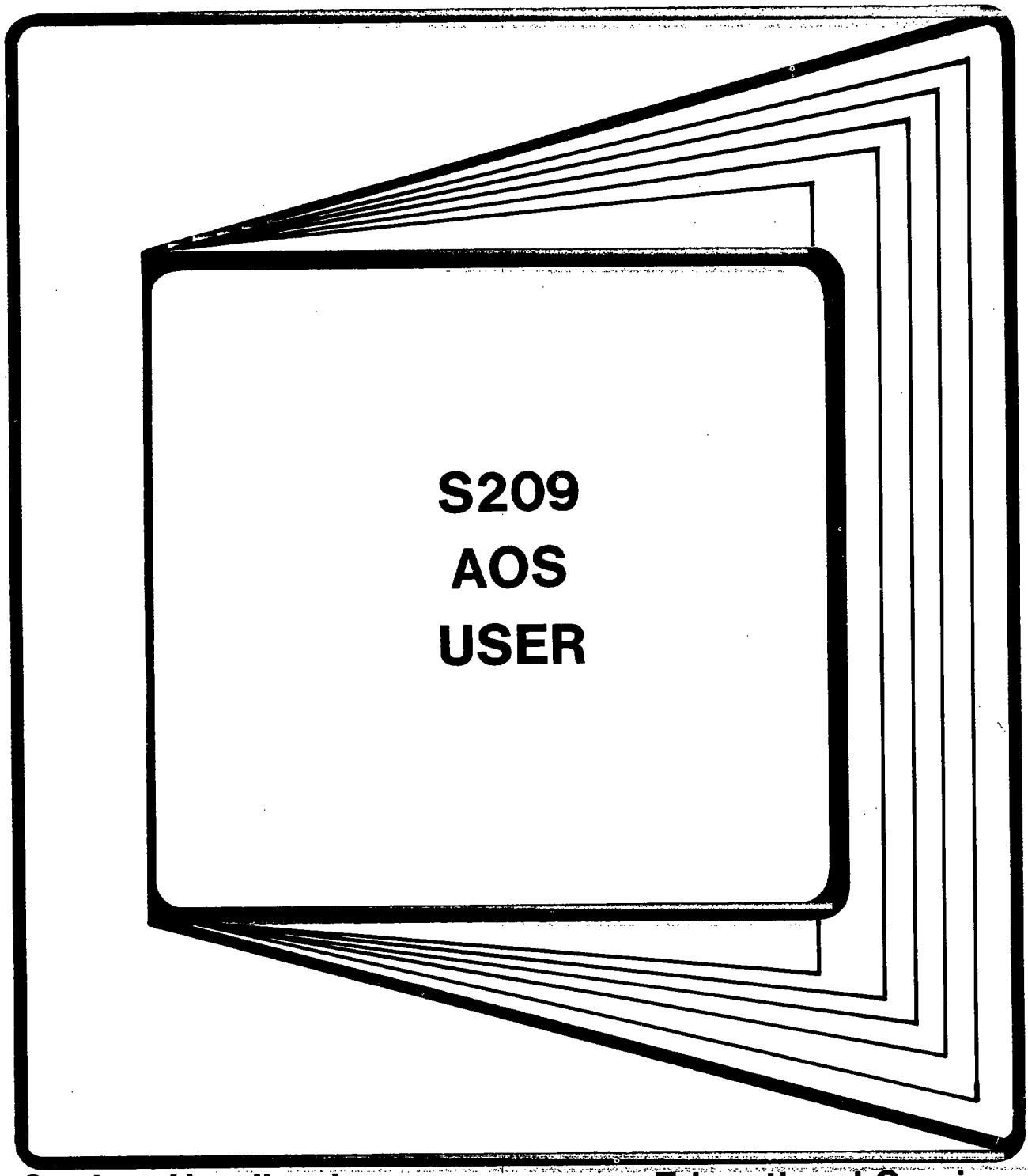
Data General



S209
AOS
USER

Student Handbook

Educational Services

019-000049-02

NOTICE

DATA GENERAL CORPORATION (DGC) HAS PREPARED THIS DOCUMENT FOR USE BY DGC PERSONNEL, LICENSEES, AND CUSTOMERS. THE INFORMATION CONTAINED HEREIN IS THE PROPERTY OF DGC AND SHALL NOT BE REPRODUCED IN WHOLE OR IN PART WITHOUT DGC PRIOR WRITTEN APPROVAL.

DGC reserves the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult DGC to determine whether any such changes have been made.

THE TERMS AND CONDITIONS GOVERNING THE SALE OF DGC HARDWARE PRODUCTS AND THE LICENSING OF DGC SOFTWARE CONSIST SOLELY OF THOSE SET FORTH IN THE WRITTEN CONTRACTS BETWEEN DGC AND ITS CUSTOMERS. NO REPRESENTATION OR OTHER AFFIRMATION OF FACT CONTAINED IN THIS DOCUMENT INCLUDING BUT NOT LIMITED TO STATEMENTS REGARDING CAPACITY, RESPONSE-TIME PERFORMANCE, SUITABILITY FOR USE OR PERFORMANCE OF PRODUCTS DESCRIBED HEREIN SHALL BE DEEMED TO BE A WARRANTY BY DGC FOR ANY PURPOSE, OR GIVE RISE TO ANY LIABILITY OF DGC WHATSOEVER.

Rev. 02, May 1979

S209

AOS USER

019-000049-02

COURSE CONTENTS

SECTION I

INTRODUCTION

SECTION I: INTRODUCTION

GOAL: To introduce Data General and AOS to Computer System Users.

OBJECTIVES: To be able to:

1) Find answers to AOS-related questions in DG manuals.

2) List typical categories of AOS application programs.

3) Identify which computer languages are supported by AOS.

4) Specify key distinctions between AOS and each of Data General's predecessor operating systems.

5) Write CLI commands in correct syntactical format.

6) Utilize parentheses, angle brackets, and templates to replace groups of CLI commands with more concise expressions.

S209 SOFTWARE DOCUMENTATION

• CLI	93-122
• Learning to Use AOS	93-196
• Intro to AOS	93-121
• Software Documentation	93-202
• AOS Binder	93-190
• Speed	93-197
• Linedit	93-197
• Debug/Diskedit	93-195

WHAT IS AOS?

- A general-purpose, disk-based operating system that controls and monitors user program processing on Data General ECLIPSE computers. CLI - COMMAND LINE INTERPRETER
- A manager of many program control, input/output, and file access functions.
- You manage AOS, and AOS manages the computer.

DG OPERATING SYSTEM HISTORY

•	DOS	Disc Operating System	(April, 1970)
•	RTOS	Real Time Operating System	(January, 1971)
•	SOS	Stand Alone Operating System	(June, 1971)
•	RDOS	Real Time Disc Operating System	(February, 1972)
•	MRDOS	Mapped Real Time Disc Operating System	(March, 1973)
•	AOS	Advanced Operating System	(November, 1976)
•	AOS/VS	" " " " " " VIRTUAL STORAGE	
•	MPOS	MNOVA OPERATING SYSTEM	
•	MPAOS	MECLIPSE " " "	

INTELLIGENT RESOURCE MANAGEMENT

A) Control of Resources

1) Keeps track of status of each resource
2) Decides what job is to get a resource
3) Allocation of resources
4) Reclamation of resources

B) Resources to be managed

1) Processor
2) Memory
3) Devices
4) Information

TYPICAL AOS APPLICATIONS

	Characteristics	Typical Applications
REAL-TIME	Strict Time Limits Dedicated Resources Special Devices	Process Control Alarm Reaction Monitoring System
BATCH	Time Independent Uses "Leftover" System Resources Mag Tapes, Cards, Printers	Compilations Payroll Reports
TIME-SHARING	Interactive Entry/Retrieval Variable Resource Demands Consoles	Point of Sale Inventory Program Development

MINIMUM EQUIPMENT CONFIGURATION

- Any ECLIPSE S/130, C/150, S/230, S/250, C/330, C/350 or M600 S140 computer with at least 128K ~~bytes~~ WORDS of read/write memory. MV SERIES
- 10 MB disc storage.
- Real Time Clock (RTC), 10 HZ
- Programmable Interval Timer (PIT)
- 9-track magnetic tape or diskette and all applicable subassemblies
- Any DGC terminal

LANGUAGES

- MACRO ASSEMBLER
- BASIC
- FORTRAN IV
- FORTRAN 5
- FORTRAN 77
- DG/L
- PL/I
- COBOL
- RPG II
- Idea*
- PASCOW

*Interactive Data Entry/Access

PROCESS DEFINITION

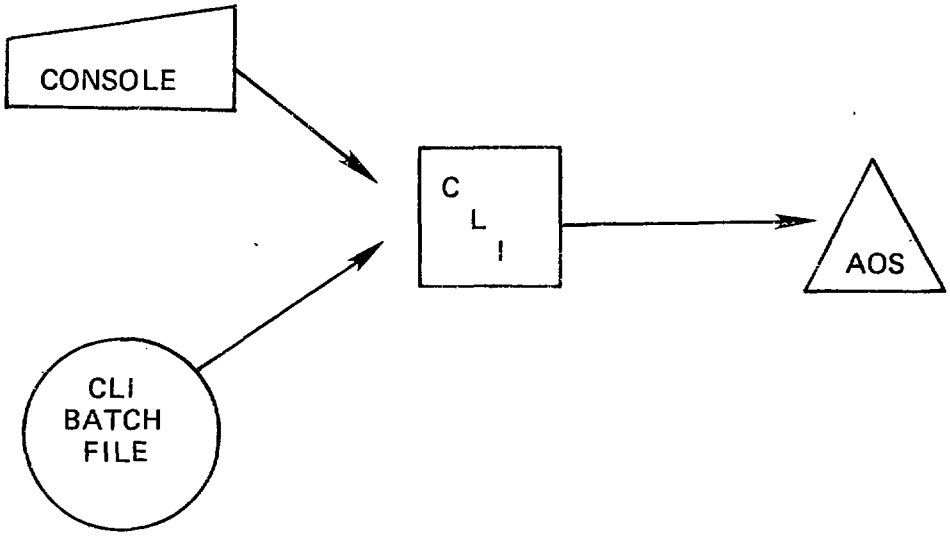
- A user of SYSTEM resources
- Each process consists of:

  UNIQUE ID
  USERNAME
  MEMORY
  PRIORITY
  PROGRAM
  PRIVILEGES
  STATE
  TYPE

COMMAND LINE INTERPRETER – CLI

Definition

- Interface between the operating system and the user. i.e., translates simple to use commands into AOS system calls
- can be used in both interactive and BATCH operations.



COMMAND LINE SYNTAX

– COMMAND [/Switch], [argument] [/Switch] . . .

where:

COMMA, TAB, AND SPACE ARE EQUIVALENT SEPERATORS

COMMAND	–	Identifies an operation to be performed. Command can be truncated to a form that will uniquely identify it to CLI.
SWITCH	–	a slash (/) followed by a word or number that modifies the default CLI function.
ARGUMENT	–	the filename, process name, or device name which the command will affect.

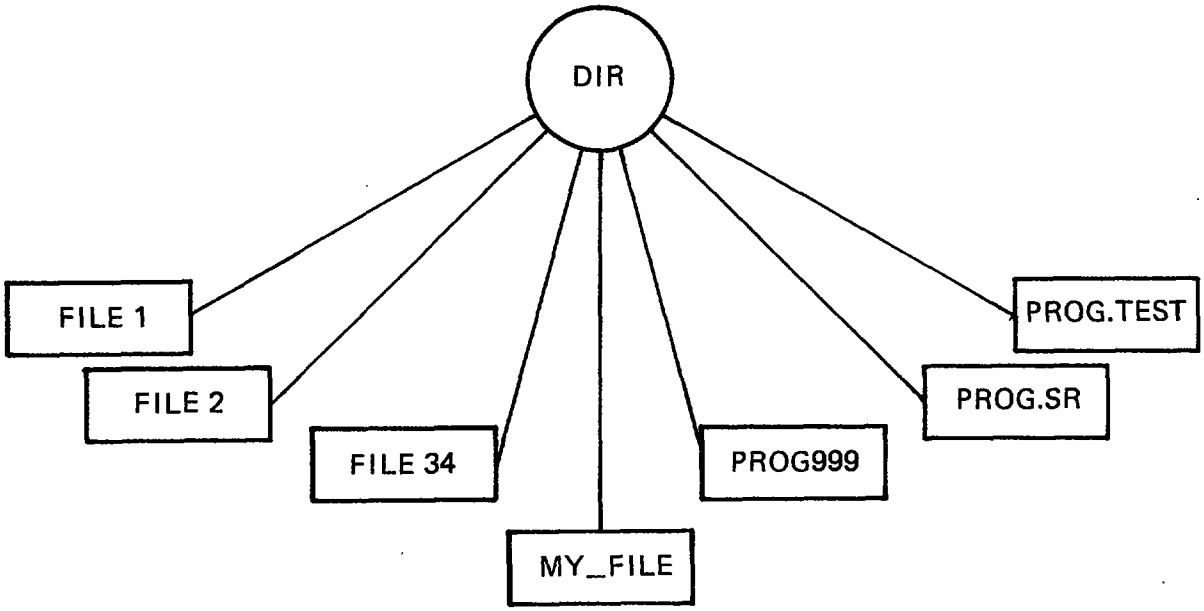
DELIMITERS

SEPARATORS	—	used to separate the COMMAND from arguments on a single command line
LEGAL VALUES:		blanks (1 or more) commas tabs (1 or more)
TERMINATORS	—	used to indicate the end of a command line
LEGAL VALUES:		NEW LINE (carriage return, line feed) carriage return (carriage return) Form feed (CTRL L) End of File (CTRL D)
CONTINUATION	—	used to continue a CLI command which exceeds one line in length
LEGAL VALUE:		& (ampersand)
COMMAND SEPERATOR	—	USED TO SEPERATE COMMANDS ON A LINE
LEGAL VALUE		; (SEMICOLON)

CODING AIDS

PARENTHESIS	–	Allows a CLI command to be executed multiple times
ANGLE BRACKETS	–	Causes in-line expansion of a CLI command
TEMPLATES	–	A symbol which can be used to represent a range of characters in a CLI command.
*	–	Matches any single character except a period
–	–	Matches any series of characters not containing a period
+	–	Matches any series of characters including periods

USE OF TEMPLATES



DIR is the working directory.

TEMPLATE	FILENAMES MATCHED
FILE1	FILE1
FILE*	FILE1 and FILE2
FILE-	FILE1, FILE2, and FILE34
PROG.-	PROG.SR and PROG.TEST
PROG+	PROG.SR, PROG999 and PROG.TEST
+	All filenames

SECTION II

FILE STRUCTURES

SECTION II: FILE STRUCTURES

GOAL: To understand the AOS Directory/File structure, and to become sufficiently familiar with the capabilities and usage of the Command Line Interpreter (CLI) to be able to design and modify the contents of that structure.

OBJECTIVES: To be able to

1) define essential facets of the file structure, such as BLOCK, FILENAME, DIRECTORY.

2) create, traverse, modify, and precisely define access limits for an AOS file.

3) list and define the four types of AOS data records

4) define the AOS file referencing mechanism and procedure

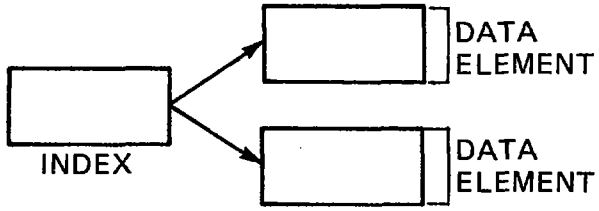
5) define and utilize Logical Disks

6) define INFOS, and state the characteristics of an INFOS file.

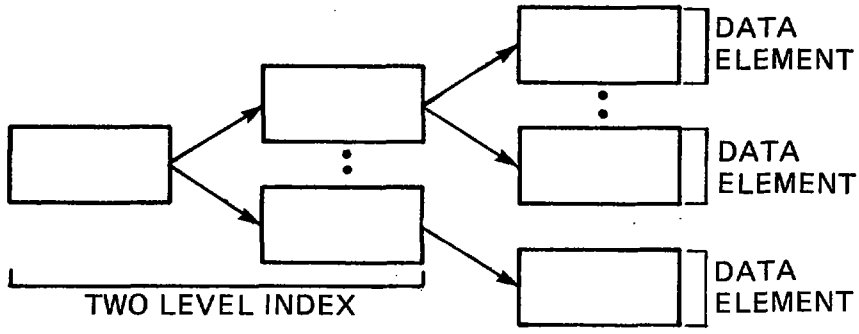
FILE STRUCTURES

BLOCK	–	512 bytes of information (1 disk sector)
ELEMENT	–	1, 2, 3, 4 or a multiple of 4 Disk Blocks
FILE	–	group of elements containing similar information, i.e., inventory, new orders, shipping.
DIRECTORY	–	Groups of logically related files

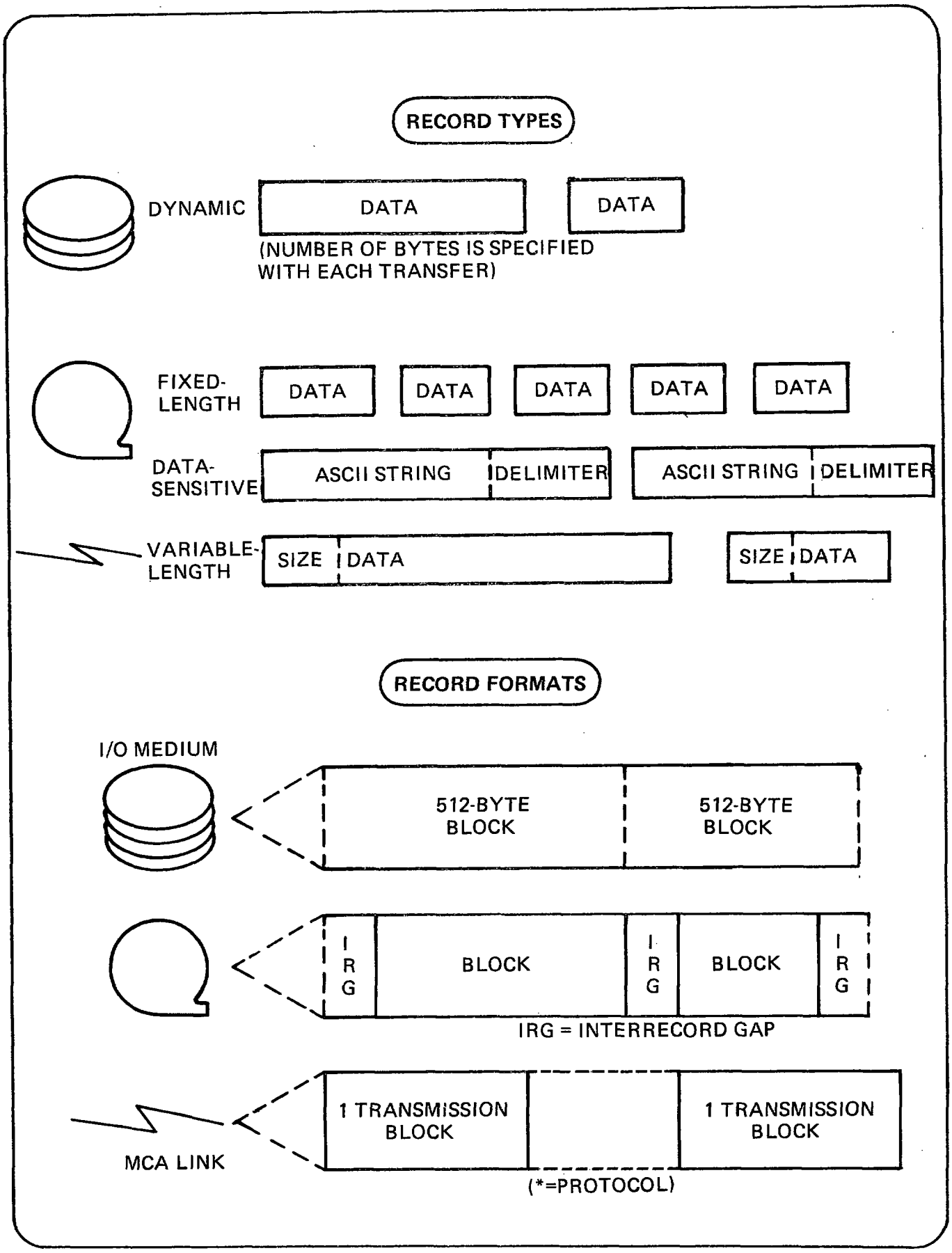
INITIAL FILE ELEMENT



MULTIPLE ELEMENTS NEED SEPARATE INDEX



More indexes are added as file data grows. To a maximum of 3 index levels.


RECORD TYPES
DYNAMIC
DATA
DATA
(NUMBER OF BYTES IS SPECIFIED WITH EACH TRANSFER)
FIXED-LENGTH
DATA
DATA
DATA
DATA
DATA
DATA-SENSITIVE
ASCII STRING
DELIMITER
ASCII STRING
DELIMITER
VARIABLE-LENGTH
SIZE
DATA
SIZE
DATA
RECORD FORMATS
I/O MEDIUM
512-BYTE BLOCK
512-BYTE BLOCK
IRG
BLOCK
IRG
BLOCK
IRG
IRG = INTERRECORD GAP
MCA LINK
1 TRANSMISSION BLOCK
1 TRANSMISSION BLOCK
(*=PROTOCOL)

FILE NAMING CONVENTIONS

- Each filename can be a maximum of 31 characters long.
- Legal characters:

  A thru Z
  0 thru 9
  ?
  .
  _ (underscore)
  $

DIRECTORIES

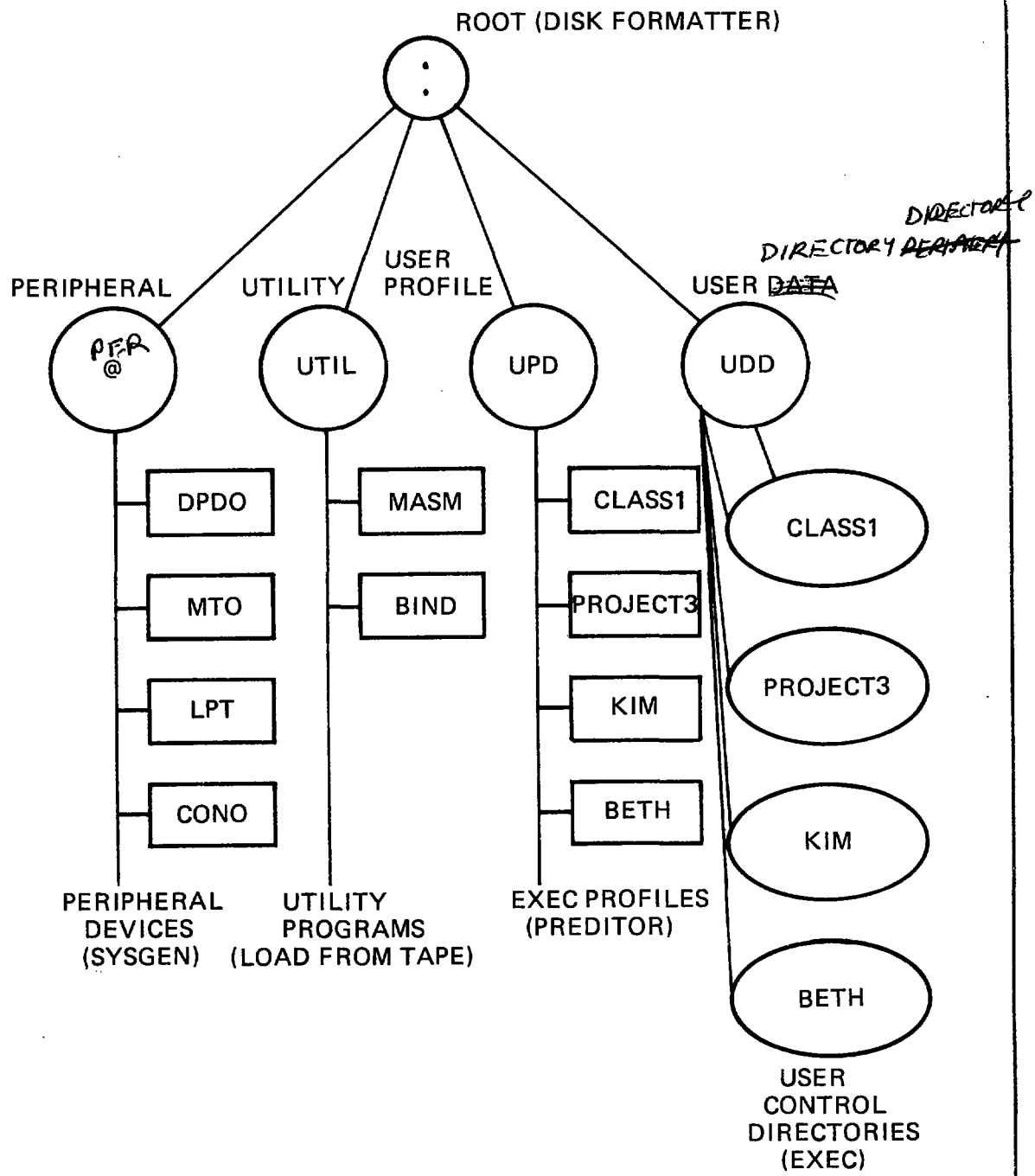
DIRECTORY — a catalog of bookkeeping information and pointers to files and subordinate directories within the directory tree

TYPES:

Control Point Directory — a directory with a fixed maximum size
i.e., create/directory/maxsize = 2000 CPD creates a control point directory named CPD having a maximum size of 2000 disk blocks

Directory — a directory which can expand dynamically
i.e., create/directory project A creates a directory named project A

DIRECTORY STRUCTURE



ACCESS CONTROL LISTS

ACL – provides protection against unauthorized use or alteration of files.

Access	Abbreviation	Non-directory File	Directory File
EXECUTE	E	User can execute file	User can use directory's name in a pathname.
READ	R	User can read (examine) data in the file.	User can examine list of files in the directory.
APPEND	A	n/a	User can insert names of new files into the directory.
WRITE	W	User can modify files' contents.	User can insert and delete files from the directory; also change ACLs.
OWNER	O	User can change file's ACL and delete file.	User can change directory's ACL, or delete directory.

REFERENCING FILES

WORKING DIRECTORY	–	Directory that the user is currently located within.
INITIAL WORKING DIRECTORY	–	Directory that user is in following log-on
PATHNAME	–	Describes the route thru the directory tree structure that the system must take to find a file.
SEARCHLIST	–	List of directories that are automatically searched when a file cannot be found in the working directory. Order of search is: 1. Look in working directory 2. Look in all directories in searchlist. 3. File not found
LINKS	–	A file that contains a pathname or pathname segment.

INFOS

INFOS files let you access records randomly, using a key you create to identify the records, or sequentially.

LANGUAGE SUPPORTED:

COBOL

RPG

Idea

MACRO ASSEMBLER

DG/L

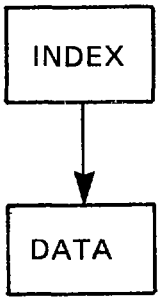
FORTRAN IV

FORTRAN V

PL/1

INDEXED SEQUENTIAL ACCESS METHOD (ISAM)

Two logical structures, an index and a database, have to be defined according to the designer's characteristics.



An INFOS file can be made up of 1 or more disks.

Up to 256 separate users can simultaneously access a single ISAM file.

Record locking is available.

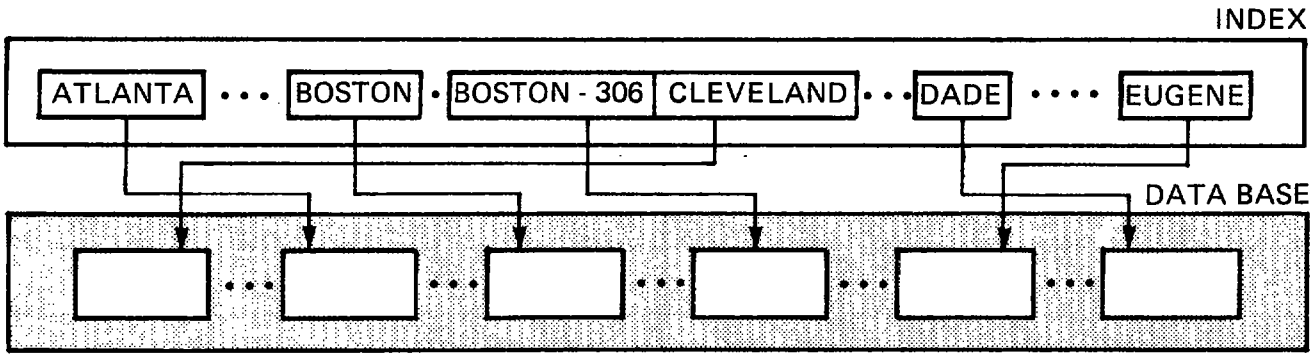
Any user can gain exclusive use of an index, if no one else is already using the file.

Each record in a database can be any length up to some predefined maximum. It is possible to combine two or more record formats in the same file.

There is automatic space management. The system recovers and makes available for reuse any space freed when you physically deleted or relocate a record.

Data records can be up to 4088 databytes long.

File inversion makes it possible to create one or more additional indexes for an existing ISAM data base, without ever having to duplicate a data record.



The keys of the index will be stored in sequence by binary value.

Key can be variable in length from one to 255 characters long.

Duplicate keys are kept unique by the INFOS assignment of occurrence numbers.

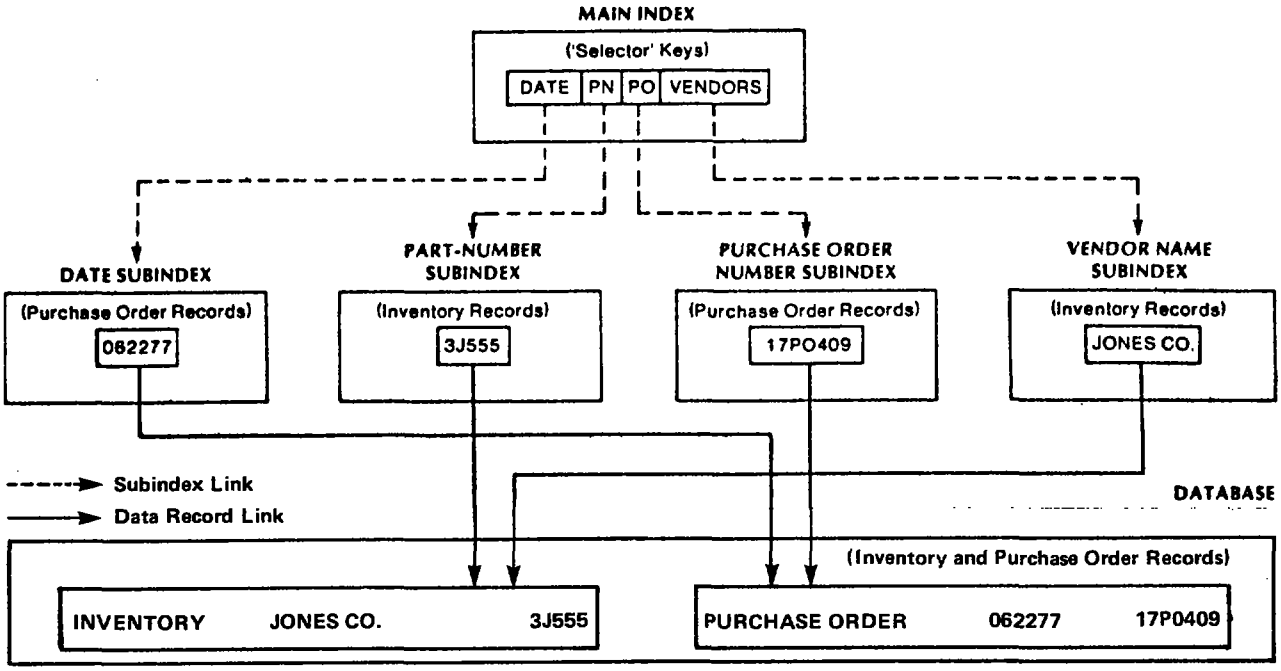
Data records are not kept in any particular order.

DATA BASE ACCESS METHOD (DBAM)

DBAM is a super set of ISAM with two additional major features.

1) You can create subsets of keys with the subindexing feature.

2) More than one key in the same index pointing to a single data record by using the inversion feature.



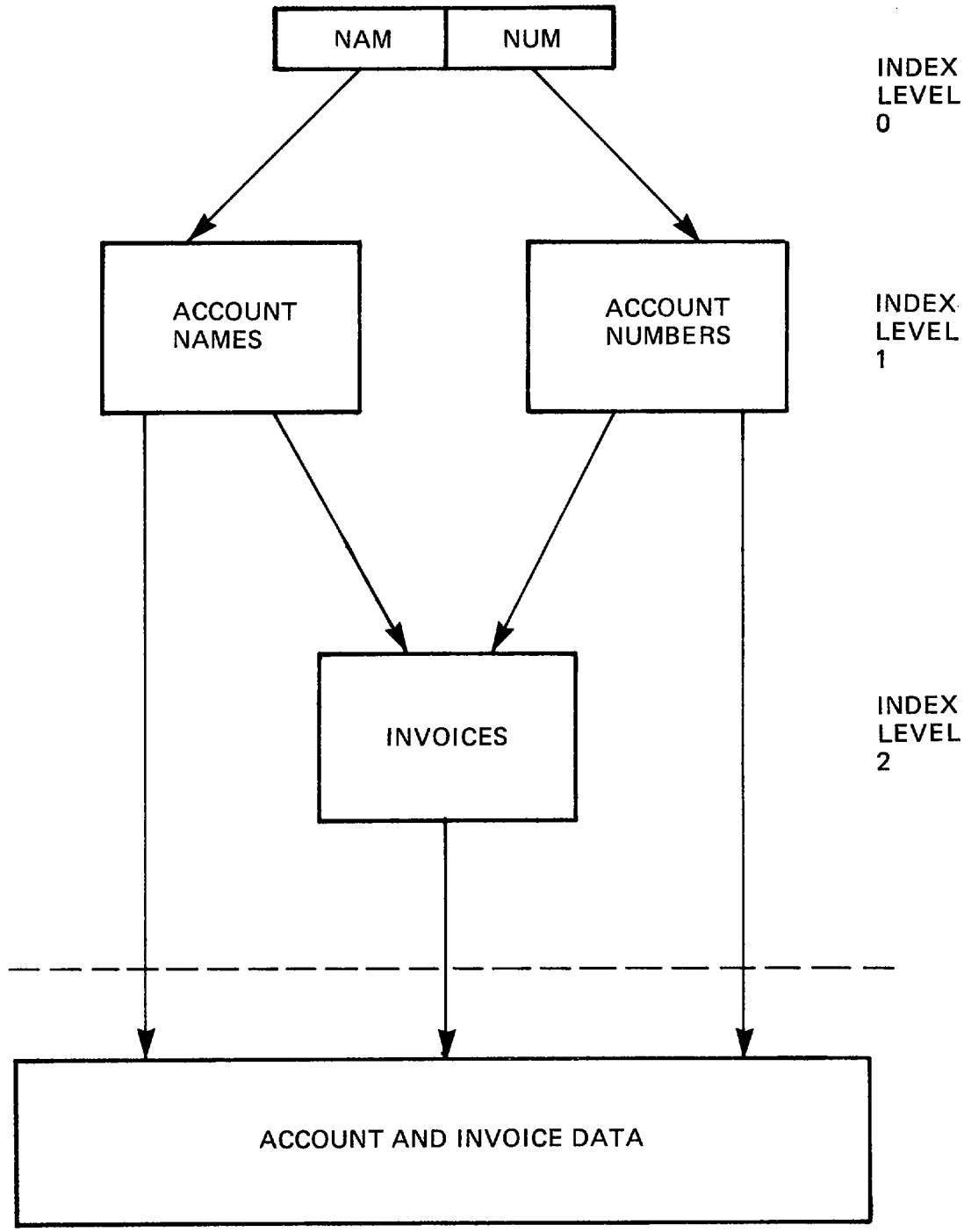
To retrieve the data base record JONES CO. the following key would be used.

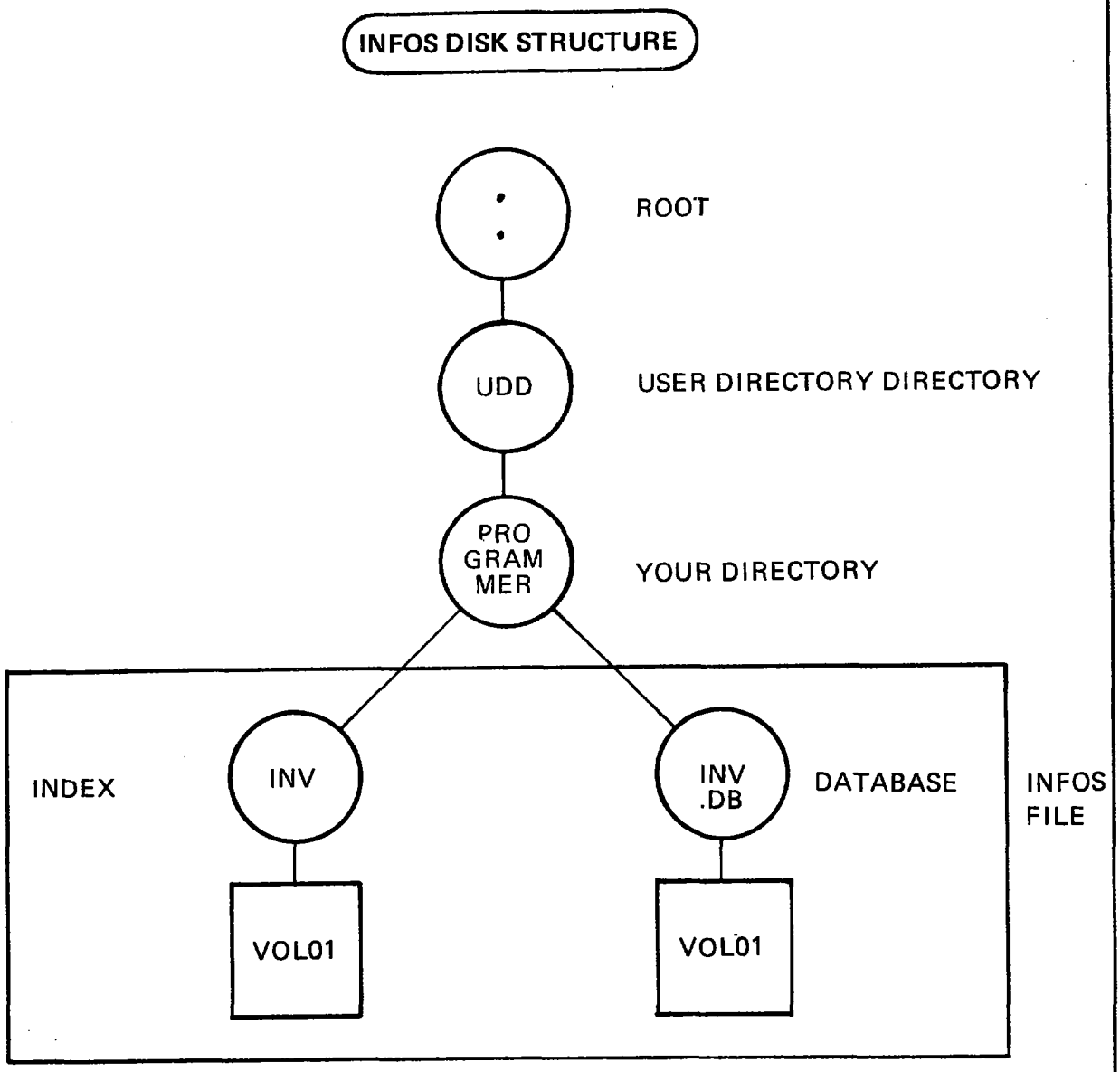
KEY: VENDORS, JONES CO.

The data base record need not contain a copy of the key information.

MULTI-LEVEL INDEXING

It is possible to have 32 levels of indexes

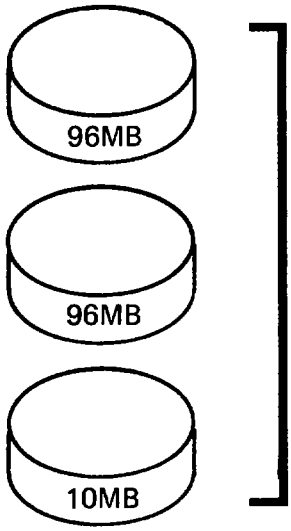




The two INFOS directory are control point directories. The Files within these directories are the actual index and data files.

LOGICAL DISKS

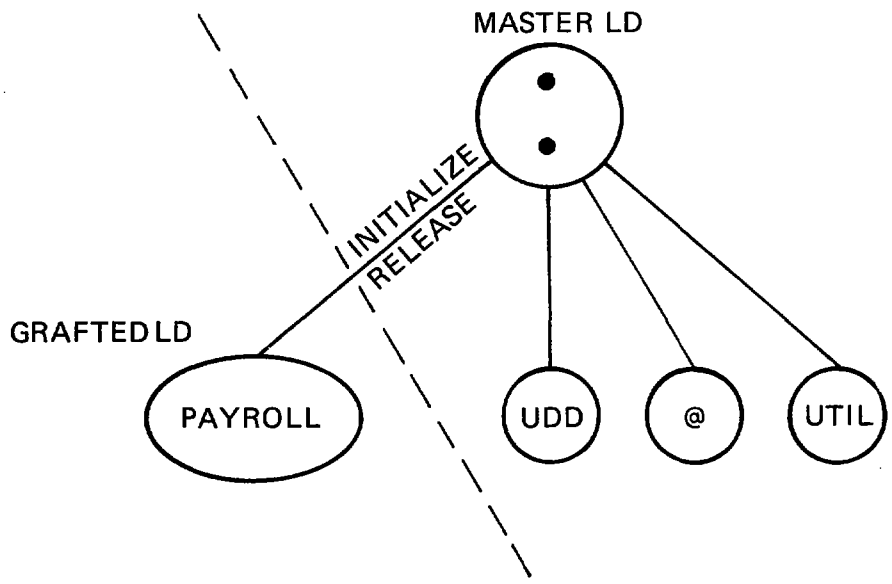
LOGICAL DISKS — A set of one or more physical disks treated by AOS as one logical unit.



The above LD would have been created by the Disk Formatter (DFMTR) utility

GRAFTING LOGICAL DISKS

Grafting an LD — the ability to add an LD to the existing directory tree structure



SECTION III

CLI ENVIRONMENT AND MACROS

SECTION III: CLI - ENVIRONMENT AND MACROS

OBJECTIVES: To be able to

1. incorporate a precise set of commands into a CLI macro and to successfully execute the macro
2. define dummy macro arguments
3. write a recursive macro which utilizes dummy arguments
4. define CLI Environment
5. define and modify AOS environmental parameters

CLI ENVIRONMENT

At log-on the environment is initially set to system default values.

The following is the CLI environment:

LEVEL
SUPERUSER
SQUEEZE
CLASS 1
CLASS 2
LISTFILE
DATAFILE
DIRECTORY
SEARCHLIST
STRING
PROMPT

All of the above are valid CLI commands which can be used to examine or alter the default settings.

MANAGING CLI ENVIRONMENT

PUSH	—	saves the current environment settings
POP	—	restores the previous environment settings
CURRENT	—	Displays the present environment
PREVIOUS	—	Displays the settings for the last environment "Push" ed

NOTES

CLI MACROS

- A file which consists of 1 or more valid CLI commands uses:

  saves key strokes
  eliminates key entry errors

- Example:

  To create the CLI macro called TRANSFER.CLI

```
)CREATE/I, TRANSFER.CLI
))MOVE/NACL,:UDD:JONES,FILE1
))DIR,:UDD:JONES
))RENAME,FILE1,MYFILE
)))
)
```

  To ~~execute~~ INVOKE the TRANSFER.CLI MACRO
  Enter:

```
)TRANSFER
```

MACRO DUMMY ARGUMENTS

- CLI allows you to write generalized MACROS for which arguments can be substituted at execution time

- Dummy arguments are enclosed between percent signs (%n%)

- Example
  To create a MACRO called TRANSFER.CLI using dummy arguments:

```
)CREATE/I, TRANSFER.CLI
))MOVE/NACL, %1%,%2%
))DIR %1%
))RENAME %2% %3%
)))
)
```

TO EXECUTE THE TRANSFER MACRO:

```
)TRANSFER,:UDD:FILE1, MYFILE
```

:UDD:JONES IS SUBSTITUTED FOR %1%
FILE1 IS SUBSTITUTED FOR %2%
MYFILE IS SUBSTITUTED FOR %3%

SEE HANDOUT

PSEUDO MACROS

A tool which allows the user greater flexibility when interacting with CLI

Pseudo Macros:

1. Return environmental settings:
   i.e., [!SEARCHLIST]

2. Return system variables:
   i.e., [!DATE]

3. Allow for conditional execution of commands
   i.e., [!EQUAL, ARG1, ARG2]

4. Allow for converting of values
   i.e., [!OCTAL, DECIMALNUMBER]

EXAMPLE

The name of the following Pseudo Macro is analysis.CLI

```
[!EQUAL, SALESMGR, [!USERNAME]]
PUSH
DIR:SALES
SEND, @CON0,*SALES ANALYSIS PROGRAM*
SEND,@CON0,*[!DATE],[!TIME]*
XEQ SALES ANALYSIS
SEND @CON0,*SALES ANALYSIS COMPLETED*
SEND @CON0, *[!DATE],[!TIME]*
POP
[!ELSE]
WRITE YOU ARE NOT AUTHORIZED TO USE THIS PROGRAM
BYE
[!END]
```

To execute this Macro:

```
)ANALYSIS
```

SEE HANDOUT

SECTION IV

PROCESS CONCEPTS

SECTION IV: PROCESS CONCEPTS

GOAL: To understand the AOS Process Heirarchy, and the factors influencing process scheduling.

OBJECTIVES: To be able to

1. state the differences between PROGRAM and PROCESS
2. create processes of different types at different priorities
3. state the impact of process type on memory utilization
4. state the impact of process priority on CPU-time allocation
5. define and list differences between interactive and compute-bourd processes

PROCESS

A Process is a resource allocation unit of AOS

PROCESS vs PROGRAMS

A PROGRAM is a series of instructions, produced by linking together a set of compiled or assembled source files.

A PROGRAM is a static entity which contains potentially executable code paths while

---

A PROCESS is a dynamic entity, existing only when it is executing program instructions.

Therefore,

A PROCESS is seen by AOS as the primary user of system resources.

For this reason,

A PROCESS competes with other processes for system resources such as memory, I/O devices, disk file space, and CPU time.

PROCESS TYPES

RESIDENT: Always remains in main memory

PREEMPTIBLE: Usually resides in main memory, but is swapped to disk if:

1. It becomes blocked and any other process needs memory.
2. A resident or higher priority preemptible process needs memory.

SWAPPABLE: Receives main memory only if any remains available after all higher type processes have received what they need.

PROCESS STATES

ELIGIBLE	INELIGIBLE

• Ready to Run
• In Main Memory
	• Ready to Run, BUT
	• Main Memory Not allocated
	  • swapped to disk
	  • Initial creation

BLOCKED

- Waiting for Some External Event
- In Main Memory, or Swapped to Disk

PROCESS PRIORITY CONCEPTS

SHOWS RELATIVE IMPORTANCE
USER – ASSIGNED NUMBER
LOW NUMBER = HIGH PRIORITY
(1 = HIGHEST)

USED TO AWARD MEMORY TO PREEMPTIBLE AND SWAPPABLE

USED FOR SCHEDULING OF ALL TYPES

RESOURCES PASSED AROUND ON A ROUND ROBIN BASIS IF MANY PROCESSES AT SAME PRIORITY

PROCESS PRIORITY

RESIDENT PREEMPTIBLE	numerical priorities ranging from 1 (highest) to 255 (lowest) are given. Both process types are treated as a single group and are allocated CPU time according to relative priority numbers.
SWAPPABLE:	user defined relative priorities are assigned from 1 (highest) to 3 (lowest). The derived priority is a function of relative priority and Behavior Rating.

CHANGING RATINGS

- AOS continually updates behavior ratings to keep up with any changes in user-program interactions.
- Behavior rating values range from 1 thru 6 with 1 being "most interactive" and 6 being "compute bound"
- Behavior ratings are calculated in the following manner:

1. AOS gives a process some time to run its program.
2. If a process becomes blocked before the given time slice is over, move behavior rating toward interactive.
   NEW BR = OLD BR −1
3. If Process used its time slice without becoming blocked, move behavior rating toward compute bound
   NEW BR = OLD BR +1

BIAS FACTOR

SWAPPABLE ONLY

AOS USE OF
BEHAVIOR RATING IS
BIASED TOWARD HIGHLY INTERACTIVE
FOR MEMORY AND SCHEDULING

BIAS FACTOR IS
NUMBER OF COMPUTE BOUND
(RATING = 6)
PROCESSES TO KEEP IN MEMORY
IF POSSIBLE –
GETS TO RUN WHILE OTHERS SWAPPING

ONLY USE IF
PLENTY OF MEMORY

NOTES

PROCESS HIERARCHY

The relationship of processes permits an additional dimension of control over process execution.

At start up, AOS creates two subordinate processes:

PMGR	—	The Peripheral Manager, which handles all byte I/O devices for AOS
OP	—	The Operator Process, which executes the initial CLI and permits the operator to create other subordinate processes of any type, at any priority

The creator (parent) of a process can assign any privileges it possesses to its offspring.

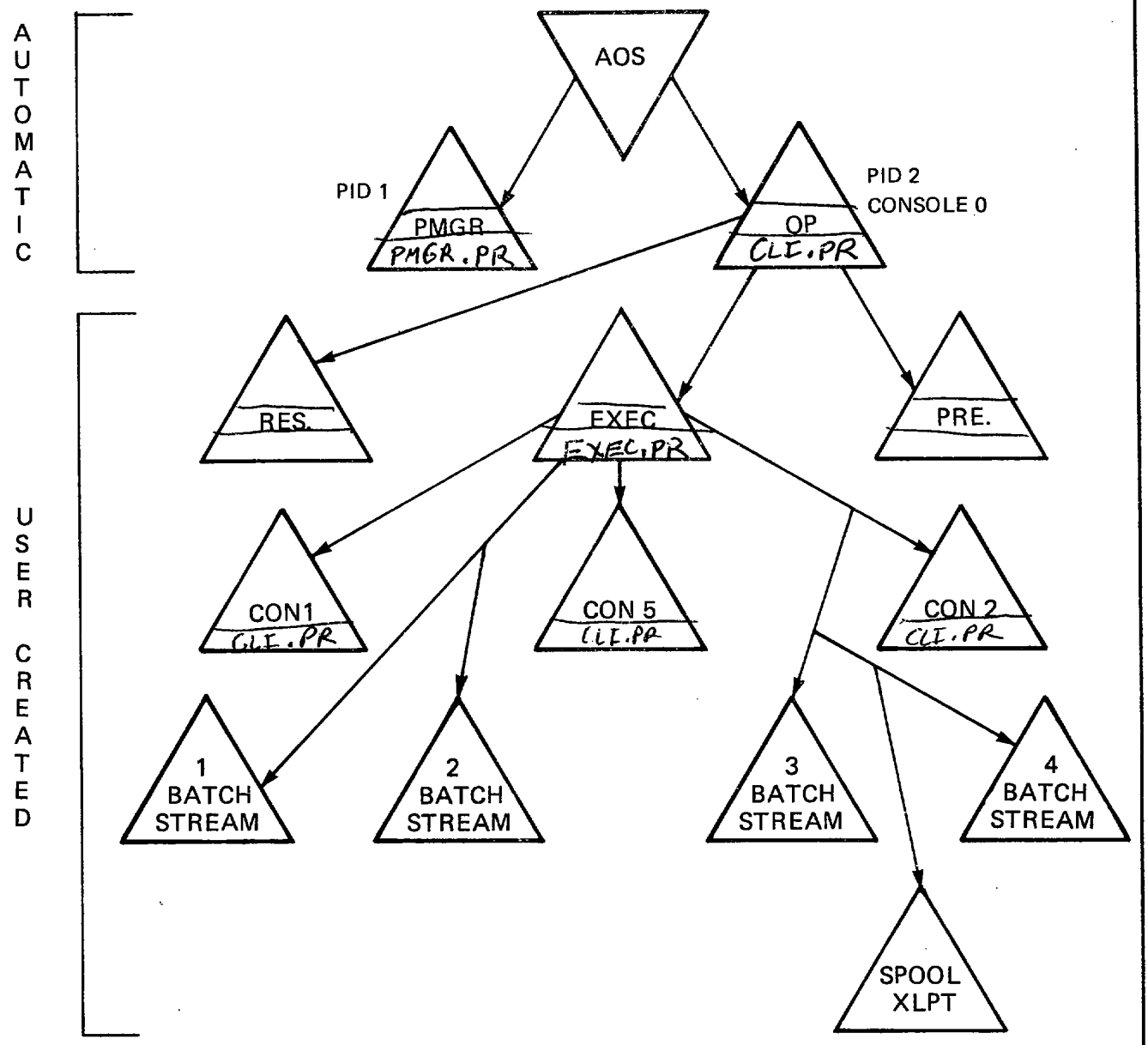
When one process creates another, the new one is subordinate to the creator.

The parent (creating) process can terminate or block any of its descendents.

When a process terminates, any process subordinate to it is also terminated.

Subordinate processes cannot terminate or block superior processes.

AOS PROCESS TREE (HIERARCHY)



THE EXEC PROCESS

- Handles AOS-User interaction at all consoles.
- Creates swappable process for time-shared terminals, batch streams, and spooling (virtual devices).
- Generally created at start-up time by "OP", as part of the "UP" macro.

PROCESS NAMES

EACH PROCESS HAS A
UNIQUE FULL NAME

USER NAME: SIMPLE PROCESS NAME
(MAX = 15 CHARACTERS EACH)

USER NAME
SAME FOR ALL PROCESSES
RUNNING UNDER ONE USER

KEY TO FILE ACCESS –
EACH USER OWNS PART OF THE DISK

SIMPLE PROCESS NAME
IDENTIFIES DIFFERENT PROCESSES
OF A SINGLE USER

NOTES

PROCESS ID (PID)

AOS CHOOSES
A UNIQUE ID NUMBER
FOR EACH PROCESS CREATED

EITHER PID
OR
FULL PROCESS NAME
CAN BE USED TO
ACCESS A PROCESS

NAMES EASY FOR YOU
PIDS EASY FOR AOS

PRIVILEGES

POWER TO CONTROL
RESOURCE ALLOCATION

- SEEN BY THE USER AS THE ABILITY TO ACQUIRE MORE RESOURCES FROM AOS, AND EXTEND CONTROL OVER RESOURCE DISTRIBUTION.
- SYSTEM MANAGER LIMITS USERS'. ABILITY TO OBTAIN RESOURCES BY APPROPRIATE RESPONSE (YES, NO) WHEN BUILDING THE USER'S PROFILE, OR BY SWITCHES DURING PROCESS CREATION.
- SYSTEM MANAGER, THEREFORE, SHOULD HAVE ULTIMATE CONTROL OVER RESOURCE ALLOCATION BY GRANTING OR DENYING PRIVILEGES

GRANTING PRIVILEGES

PARENT PROCESS MUST HAVE ALL PRIVILEGES IT WANTS TO PASS TO DESCENDENTS
—OR IN OTHER WORDS—
ONCE A PRIVILEGE IS DENIED FOR A PROCESS, IT IS ALSO UNAVAILABLE TO THE OFFSPRING OF THAT PROCESS

PRIVILEGES ALWAYS GIVEN BY PARENT EVEN IF THE NEWLY CREATED PROCESS HAS A DIFFERENT USER NAME.

PRIVILEGES SHOULD BE GIVEN ON A NEED–TO–HAVE BASIS.

SYSTEM PROCESS CONCERNS

- AOS supports 125 processes.
- PMGR, OP, EXEC use up 1 process each.
- Each general batch stream uses 2 processes.
- Each device being spooled use 1 process.
- Each general timesharing terminal uses 2 processes.
- Each additional process stretches resources thinner while increasing AOS bookkeeping demands.
- PID 2 has all possible privileges assigned to it by AOS

GENERIC FILENAMES

- Predefined entries in the peripheral directory.
- Each parent process links these names to actual files or devices.
- By using a generic filename, an offspring process need not name specific files.
- Different processes running the same program may use different devices for the same purpose (e.g., output to magnetic tape or the printer) simply by having different generic filename associations.
- There are six generic files:

@ CONSOLE	Your processes interactive console.
@ DATA	A long input file.
@ INPUT	A short input file.
@ LIST	A long output file.
@ OUTPUT	A short output file
@ NULL	A special purpose I/O file.

SECTION V

EXEC

SECTION V: EXEC

GOAL: To understand the usage and features of the AOS Executive utility program.

OBJECTIVES: To be able to

1. list the four principal functions of EXEC
2. submit program requests to BATCH_INPUT
3. create and direct process output to a spool queue
4. control data flow to and from spool queues
5. control data flow to and from I/O devices

THE EXEC PROCESS

CONTROLS USER ACCESS TO THE AOS SYSTEM

4 KINDS OF FUNCTIONS

1. LOGON FUNCTION MONITORS CONSOLES TO ALLOW USERS TO GAIN ACCESS TO THE SYSTEM

2. BATCH FUNCTION CAN SUPERVISE AND RUN USER PROGRAMS IN BATCH STREAMS.

3. SPOOLING FUNCTION ENSURES THAT USERS SENDING OUTPUT TO CERTAIN CHARACTER ORIENTED DEVICES CAN USE THEM IN AN ORDERLY WAY.

4. MOUNT DISMOUNT FUNCTION ALLOWS USERS IN BATCH OR INTERACTIVE MODE TO REQUEST THE OPERATOR TO MOUNT AND DISMOUNT TAPES AND DISKS.

EXEC CREATES ONLY SWAPPABLE PROCESSES

PROVIDES USER LEVEL ACCOUNTING INFORMATION

EXEC–AOS INTERACTION

AOS

* ENFORCES PRIVILEGE LIMITS
* ENFORCES MEMORY & DISK LIMITS
* PROTECTS FILES BY USER NAME
* HEURISTIC SCHEDULING OF SWAPPABLE PROCESSES
* RECORDS RESOURCE USAGE

EXEC

* MONITORS SPECIFIED CONSOLES
* LINKS PASSWORD WITH USER NAME
* CREATES SWAPPABLE PROCESSES
* ONLY RECOGNIZES USERS WITH PROFILES
* LISTENS FOR CONTROL COMMANDS
* LOGS EXTRA STATISTICS
* RUNS BATCH STREAMS
* PERFORMS DEVICE SPOOLING

EXEC CREATION

- The EXECUTIVE PROCESS is created as a swappable process with all privileges granted to it.

- Only one process running the EXECUTIVE program is permitted.

- To create a process running the EXECUTIVE Program:

)PROC/DIR=@/DEF,:UTIL:EXEC.PR

THE EXEC BATCH FUNCTION

The BATCH function of EXEC is the non-interactive processing of program requests, submitted by users through:

CLI — Permitting other on-line processing while the batch job is serviced

- QBATCH — One time submission
- QSUBMIT — User File containing several commands used for repeated submissions.

Card Readers — In "STACKED" format, including a user-name/password pair

EXEC can process up to four batch jobs simultaneously, each one executing in a separate BATCH STREAM.

EXEC also creates four specialized queues to support batch processing:

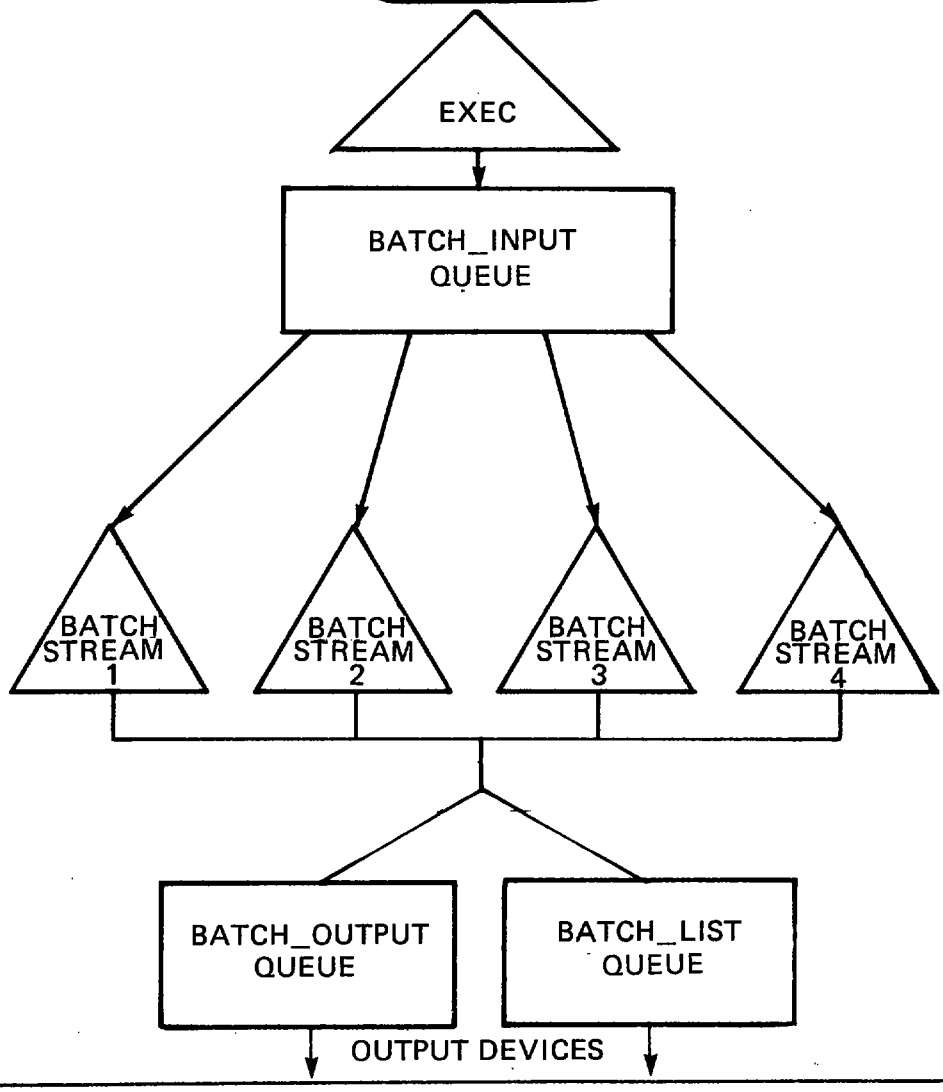
- BATCH_INPUT
- BATCH_OUTPUT
- BATCH_LIST
- MOUNT

MAKING BATCH REQUESTS

EXEC NEEDS USER PROFILE
TO CREATE SWAPPABLE
PROCESS AND RUN PROPER INITIAL
PROGRAM

QBATCH & QSUBMIT
INTERACTIVE REQUESTS
USE SAME PROFILE AS REQUESTING
PROGRAM'S PROCESS

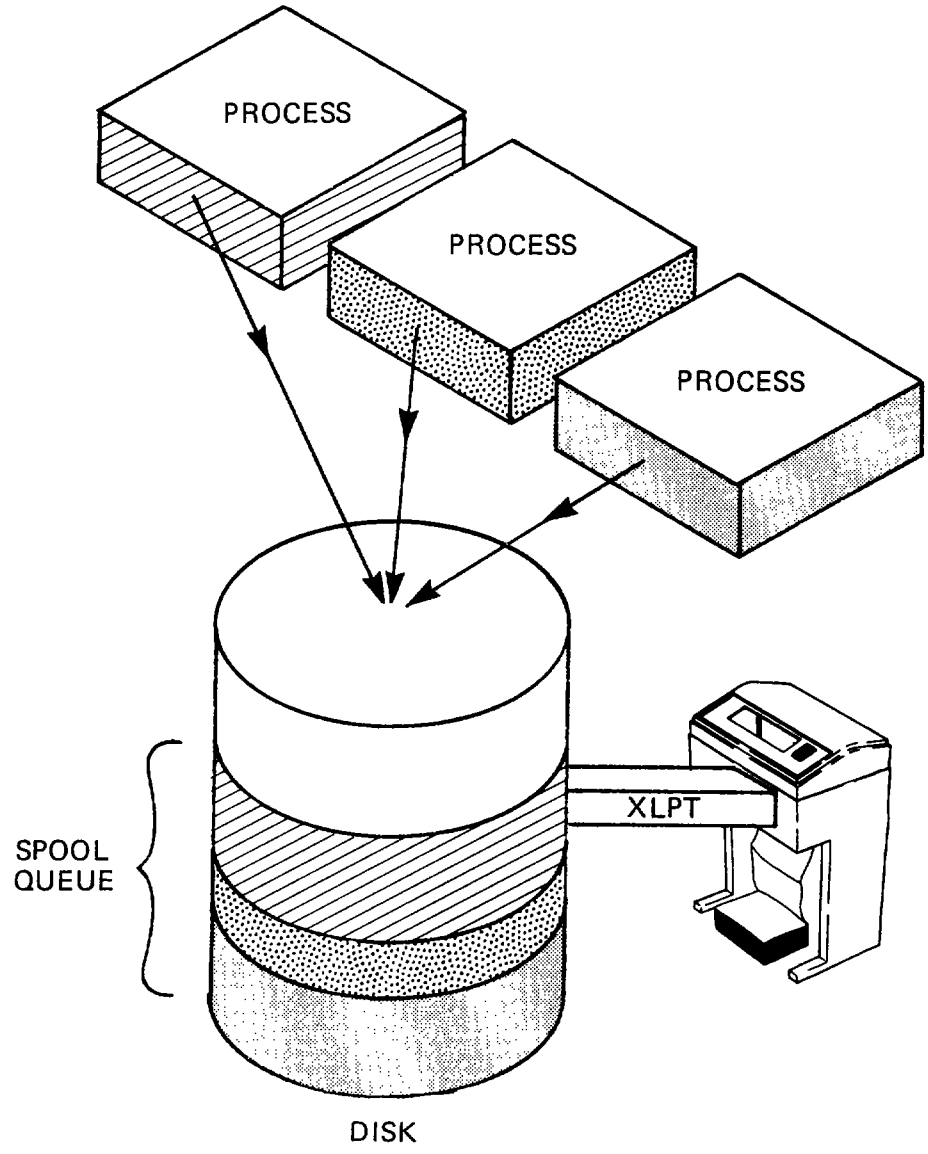
BATCH FLOW



SPOOLING

- Slow output devices need not slow down processing.
- A spool queue is itself a virtual output device. Actual output is deferred.
- Multiple processes may appear to use the same devices simultaneously.
- When you open a spool queue, any process accessing the queue has its output temporarily diverted to and queued by a disk file. This permits output speeds to be at fast disk rates.
- CLI commands QPRINT, QPUNCH, and QPLOT assume queue names LPT, PTP, and PLT.
- Operator can control
  - Characters per line
  - Lines per page
  - Pages per request
  - Header and trailer sheets

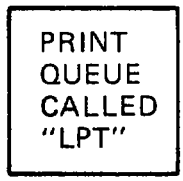
SPOOLING



SPOOLING EXAMPLE

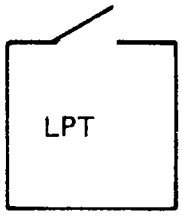
CONTROL @EXEC CREATE PRINT LPT

CREATES:



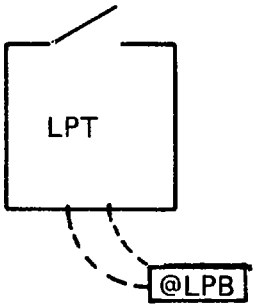
CONTROL @EXEC OPEN LPT

opens an "input door" to the queue, thereby allowing output data from processes to be collected.



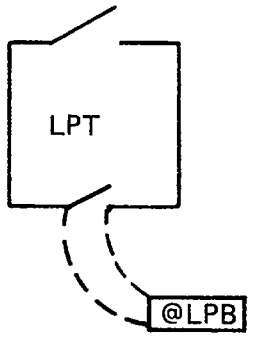
CONTROL @EXEC START LPT @LPB

creates a logical "output door" connecting the print queue LPT to a printer called @LPB



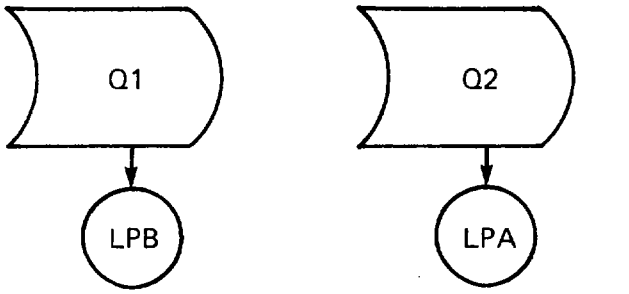
CONTROL @EXEC CONTINUE @LPB

opens the "output door" and permits printing to begin on @LPB of the spooled contents of print queue LPT

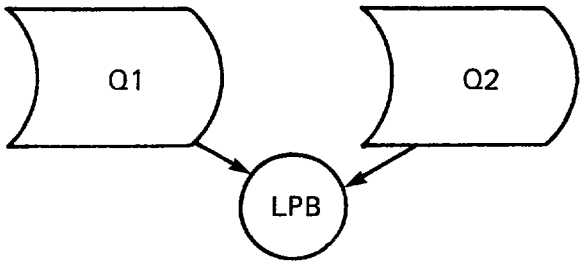


SPOOLING VARIATIONS

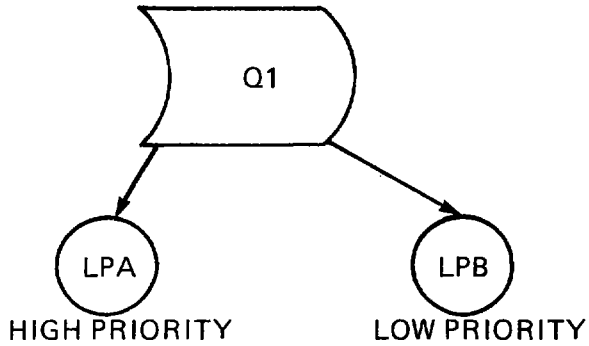
ONE QUEUE PER DEVICE:



TWO QUEUES PER DEVICE:



ONE QUEUE FOR MULTIPLE DEVICES:



UNIT MOUNT REQUEST

IF USER NEEDS MAG TAPE OR DISK BUT DOESN'T KNOW WHAT PHYSICAL UNITS ARE AVAILABLE, HE CAN REQUEST OPERATOR TO MOUNT STORAGE MEDIA ON ANY FREE UNIT VIA THE MOUNT COMMAND

i.e., "MOUNT LOGICALNAME MESSAGE"
EXEC PASSES MESSAGE TO OPERATOR AND HOLDS USER UNTIL READY.

POSSIBLE REPLIES:

* MOUNTED – MEDIA READY TO USE
* REFUSED – CAN'T FIND MEDIA NO UNIT FREE
* NO OPERATOR – BY SETTING OPERATOR SWITCH IN EXEC OFF ALL OPERATOR ACTION REQUESTS CAN BE AUTOMATICALLY REFUSED

USER PROFILES

- a list of privileges and characteristics of system users
- access control vehicle
  - user name/password pairs
- resource allocation aid
  - privileges
  - limits
- Each user profile is a one-block file in the profile directory (:UPD)

PREDITOR

Creates, edits, lists and deletes user profiles

)X PREDITOR

LOGON PROCEDURE

WHEN COMMANDED TO MONITOR
A CONSOLE EXEC WAITS
FOR THE "NEW LINE"
KEY TO BE PRESSED
ON THAT CONSOLE

AFTER "NEW LINE", EXEC PROMPTS
USER FOR:
USERNAME:
PASSWORD:

USER INPUTS NAME & PASSWORD

EXEC ACKNOWLEDGES LOGON,
REPORTS LAST LOGON
& TYPES THE
CONTENTS OF "LOGON MESSAGE" FILE
FROM ":UTIL" DIRECTORY
THEN, AOS CLI IS RUN

EXEC SUMMARY

CREATES SWAPPABLE PROCESSES
ACCORDING TO USER PROFILES.

ONCE CREATED,
PROCESS AND AOS COMMUNICATE DIRECTLY

USES QUEUES
TO HOLD WORK ORDERS ON DISK
UNTIL RESOURCES ASSIGNED.

EXEC RUNS ON ITS OWN
BUT LISTENS FOR CHANGE COMMANDS
FROM USERNAME OP.

EXEC AUTOMATES
COMMON PROCESS AND
ENVIRONMENT
CONTROL FUNCTIONS

EXEC IS A UTILITY PROGRAM WHICH
MAY OR MAY NOT BE USED ACCORDING
TO INSTALLATION NEEDS.

SECTION VI

PROGRAM DEVELOPMENT UTILITIES

SECTION VI: PROGRAM DEVELOPMENT UTILITIES

GOAL: To understand the AOS program development cycle, and the relevant utility programs available to System users.

OBJECTIVES: To be able to

1. state the phases of the standard program development cycle

2. correctly invoke and utilize all System utilities.

3. define the function of each system support utility.

4. state the conventional file extensions followed by AOS, and assumed by the utility programs.

PROGRAM DEVELOPMENT CYCLE

EDIT — The text editors (speed or linedit) which are used to create source files.

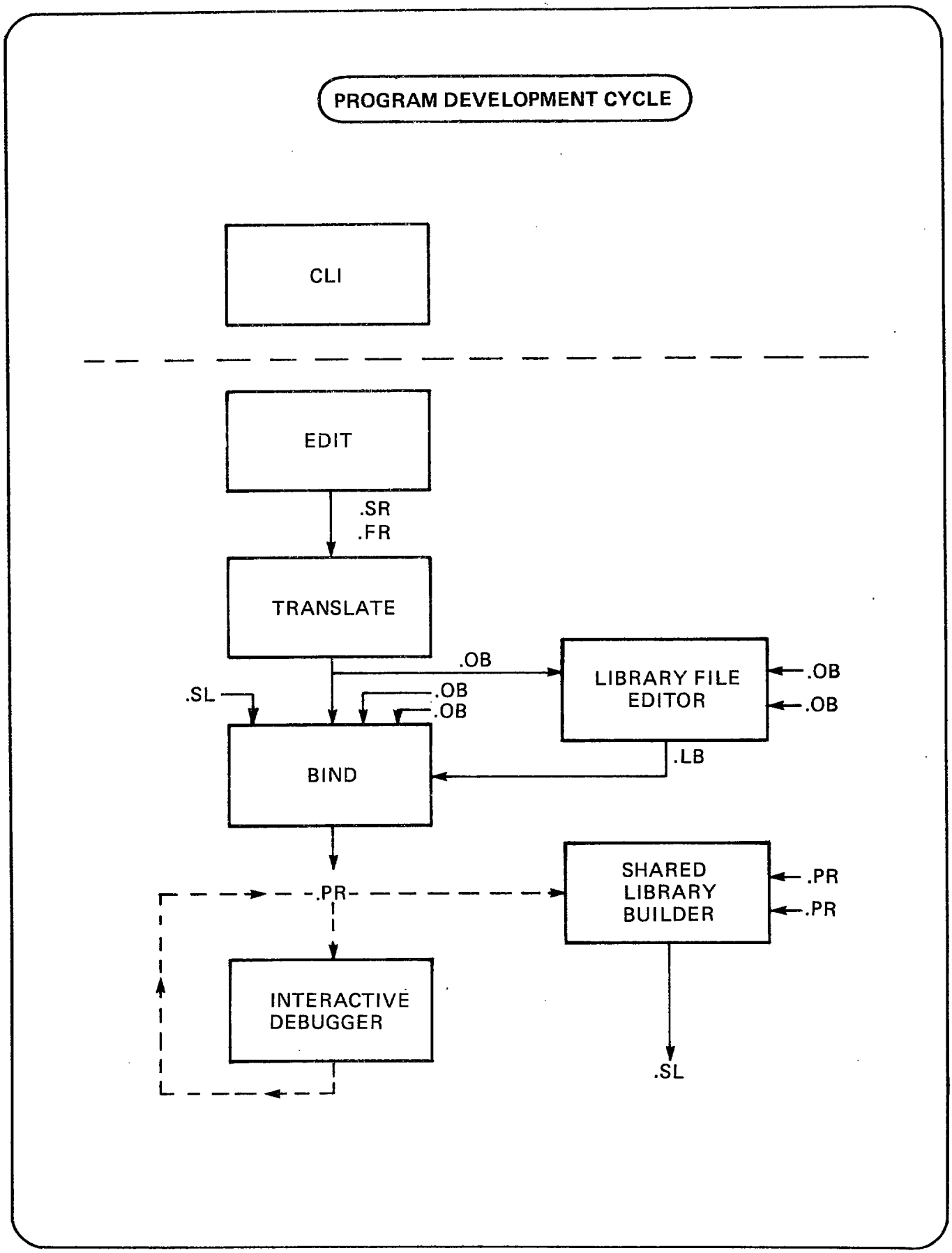
TRANSLATE — The output from the text editors is input to a compiler (FORTRAN, COBOL) or assembler (MASM) producing object modules.

LIBRARY FILE EDITOR — A utility which allows the user to combine various object modules into a library.

BIND — Object modules created in translate phase of CYCLE are linked together to form a program file.

INTERACTIVE DEBUGGER — If errors in program file are detected during execution, the Debugger utility allows the user to step through and change his program file.

SHARED LIBRARY BUILDER — Various program files are combined by this utility to create a shared library.


PROGRAM DEVELOPMENT CYCLE
CLI
EDIT
.SR
.FR
TRANSLATE
.OB
LIBRARY FILE EDITOR
.OB
.OB
.SL
.OB
.OB
BIND
.LB
SHARED LIBRARY BUILDER
.PR
.PR
.PR
INTERACTIVE DEBUGGER
.SL

AOS FILE EXTENSION

Certain conventions are followed when naming files. These conventions uniquely identify the file to a compiler or assembler as well as other utilities.

.SR	Assembly language source file
.FR	FORTRAN IV source files
.OB	object module files
.PR	program (executable) files
.ST	symbol table files
.OL	overlay files
.LB	unshared library files
.SL	shared library files

AOS FILE TYPES

AOS provides 255 types of files. Types 0-127 are defined by AOS. The user may define types 128-255

Table 2-2. File types

Number	Mnemonic	Type
0	LNK	Link
1	SDF	System data file
2	MTF	Magnetic tape file
3	GFN	Generic filename
4-9	------	(reserved)
10	DIR	Disk directory
11	LDU	Logical disk
12	CPD	Control point directory
13	MTV	Magnetic tape volume
14-19	------	(reserved)
20	DKU	Disk Unit
21	MCU	Multiprocessor Communications Unit
22	MTU	Magnetic tape unit
23-29	------	(reserved)
30	IPC	IPC port entry
31	PER	Peripheral entry
32	SPR	Spoolable peripheral entry
33	QUE	Queue entry
34-63	------	(reserved)
64	UDF	User data file
65	PRG	Program file
66	UPF	User profile file
67	STF	Symbol table file
68	TXT	Text file
69-127	------	(reserved)
128-255	------	User-defined file types

SUPER EDITOR (SPEED)

- A character oriented text editor
- Up to 36 input/output buffers allowed
- Allows you to create, modify or merge text files
- Page Mode:
  Only one page of information can be edited at a given time. A page consists of a sequence of characters ending in a form feed.
- Window Mode:
  Disregards a form feed as a page delimiter and allows you to select the number of lines to be available for editing.
- Character Pointer (CP):
  Marks the current position in the editing buffer for editing commands.
- New Line (NL), form feed, tab, and carriage return are treated as single characters by Speed.

USING SPEED

- To invoke speed:

)XEQ, SPEED, filename

- Command delimiter is the escape (ESC) key which is echoed as a single dollar sign ($)

- Command terminator is a CTRL D which is echoed as double dollar sign ($$)

- Commonly used Commands:

T	used to display one or more lines of text
L	used to move CP to the beginning of a line
J	moves CP to 1st line in EDIT BUFFER
K	deletes lines of text
M	move CP a number of characters to left or right
S	searches buffer for a character string
C	changes a character string
P	outputs the edit buffer to a file
Y	Yanks in next page or window to be edited
FU	Outputs the remaining page(s) and closes all Input and Output files
H	Terminates editing session

LINE EDIT (LINEDIT)

- A line oriented text editor
- Allows you to create and modify text files
- Only 1 page of information can be edited at a given time. A page can be a maximum of 1024 Lines long.
- Cursor Controls can be used to help modify a selected line
- A "Help" file is available for assistance during an editing session
- A command can be abbreviated to its shortest unique form
- Only the current contents of the edit buffer can be modified

USING LINEDIT

- To invoke linedit:
)XEQ linedit filename
- Command Terminator is the new line key (NL)
- Commonly used commands:

DISPLAY	Lists current position - page & line number
SET LINE	Assigns each line of text a line number
LIST	Displays lines of text
POSITION	Used to establish cursor location on a line or page
FIND	Used to search buffer for a character string
SUBSTITUTE	Changes a character string
MODIFY	Used to alter a line of text
JOIN	Combines 2 pages of text
HELP	Provides information about Linedit.
BYE	Terminates editing session

BINDER

- The Binder is a utility
- Used to link object modules (.OB files) into executable program files (.PR files)
- Creates overlay files (.OL file) if directed
- Produces a symbol table (.ST file) which contains the names and values of all the global symbols defined within an object module

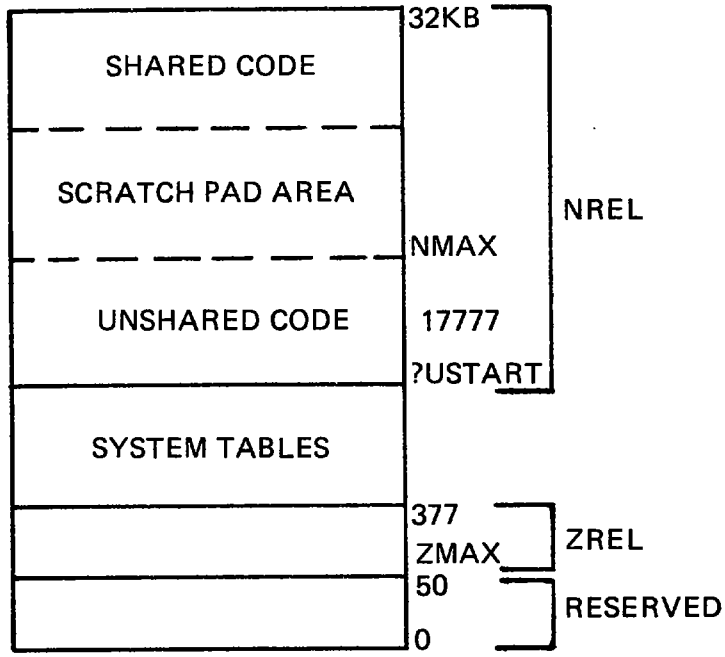
USING THE BINDER UTILITY

- Format of Binder Command:

```
XEQ BIND/[SWITCH] OBJECT MODULE/[SWITCH]...
                    OR
XEQ BIND/[SWITCH] OBJECT MODULE COMMANDFILE/C
```

- Switches applied directly to command are called FUNCTION SWITCHES
- Switches applied to arguments are called ARGUMENT SWITCHES
- Command file is a file which contains information needed to build overlay files. The argument switch /C is always applied to a command file name.

CONTENTS OF A PROGRAM FILE



ZMAX = highest used ZREL ADDRESS +1

NMAX = highest unused, unshared NREL ADDRESS +1

SYSTEM TABLE = USER STATUS TABLE
TASK CONTROL BLOCK(S)
SHARED LIBRARY TABLE
OVERLAY INFORMATION TABLES
UNLABELLED COMMON

OVERLAYS

- Overlays make it possible for a program whose length is greater than 64KBytes to run on Data General computers.

- It is possible to have 63 overlay areas for each program.

- 1023 seperate overlays can be associated with each overlay area

- Overlay size:
  - unshared allocated in multiples of 512 bytes
  - shared allocated in multiples of 2048 bytes

LIBRARY FILE EDITOR

- Makes it possible for the user to create and maintain a library contain object modules (.OB)

- The object modules may contain both shared and unshared code

- To use modules that reside in a Library.

    1. the module must be referenced by another object module

    2. the library name containing those modules must be included in the bind line

        i.e., URT.LB
        FORT.LB

- To invoke LFE utility:

    XEQ LFE FUNCTION LIBRARY NAME

    i.e., XEQ LFE A URT.LB

DISPLAY

The DISPLAY utility is used to print the contents of a disk file.

The default is that each line contains the octal and ASCII representation of eight 16-bit words.

If the ASCII character is unprintable, DISPLAY writes it as ⟨nnn⟩ where "nnn" is the ASCII code for the character. This applies to the following characters: FORM FEED, TAB, NEW LINE, and CARRIAGE RETURN. If a byte contains a quantity which does not have an ASCII equivalent, a blank is printed.

If a line contains all octal zeros the line is not printed. Instead a series of asterisks (**) is printed on the left hand side.

To display a file in Hexadecimal and ASCII a /H switch can be used.

DISK FILE EDITOR (DEDIT)

The utility is used to examine or modify locations in any AOS disk file.

The most common use of DEDIT is to patch a ".PR" file to avoid having to reassemble and rebind the source code.

Common Commands:

DISP start;end	Display a range of locations
[address] :	Displays the contents of a location
carriage return	Displays the contents of the next location
[address] ; new value	Modify the contents of location
?M	Display current modes (refer next page)
MODE character	Change the modes (refer next page)
mode character (ESC)	Display last item with a different mode
BYE	Terminate DEDIT

Format Submodes

	Character	Displays data as
*	F	A 16-bit numeric constant
	H	An 8-bit numeric constant
	A	A pair of ASCII characters
	S	A symbol plus offset
	N	An instruction
	P	Single-precision floating point data
	Q	Double-precision floating point data

Radix Submodes

	Character	Action
	B	Binary format
*	O	Octal
	D	Decimal with trailing period (.)
	X	Hexadecimal

Sign Submodes

	Character	Display Data
*	VS	Without regard to sign
	SI	Interpret sign bit. Display a minum sign prior to number if it is negative.

(Note the "*" indicates the default modes.)

SCOM

Compare two ASCII text files.

Format;

XEQ SCOM sourcefile$_1$ sourcefile$_2$

This program scans each line from both files. If it finds differences, it either outputs the difference or a message (see command switches below). The program then attempts to get back into synchronization. Synchronization is defined as finding n lines in a row that match (where n is called the matchsize). By default, the matchsize is automatically set to four.

For example, the following sequence could be found several times in one source file:

```
ISZ     ?ORTN,3     ;INDICATE SUCCESS
RTN                 :RETURN
```

Thus, SCOM would get back into synchronization only if the next two lines following the above series matched.

FILCOM –

This utility compares any two files word by word. If the files differ, the word number, and the contents of each word is displayed.

If one file is longer than the other, each excess word in the longer file is displayed and dashes are displayed for the shorter file.

Format:

XEQ FILCOM pathname$_1$pathname$_2$

Compare pathname$_1$ to pathname$_2$

FILCOM Switches:

/L	Write output to @LIST instead of @OUTPUT.
/L=name	Write output to the file name instead of @OUTPUT.

CONVERT

Convert an RDOS .RB to an AOS .OB.

Format:

XEQ CONVERT pathname

The CONVERT utility can convert an RDOS.RB (relocatable binary) file to an AOS object file. The command line takes one argument, the input pathname (you can omit the .RB extension). The RDOS file is not modified and an AOS object file is created with the same name but with the .OB extension.

Examples:

```
) XEQ CONVERT PLUS24↓
PLUS24.RB
)
```

This command line produces an AOS object file named PLUS24.OB from an RDOS object file named PLUS24.RB in the working directory. (The command RDOS LOAD might have loaded the original .RB file.) The CONVERT program displays the message PLUS24.RB when it opens the input file.

RDOS

Read or write an RDOS dumpfile on disk.

The RDOS utility copies or lists files between the working directory and an RDOS dumpfile or disk. To convert the file, include argument switch /C. You can use AOS templates (but not RDOS templates) in any command. There are five functions of RDOS – LOAD, DUMP, GET, PUT, and LIST.

The LOAD and DUMP commands load or dump an RDOS file (generally from mag tape) into your working directory. The GET, PUT, and LIST commands assume that an RDOS disk is on your system. To place an RDOS disk in the system, insert it in a suitable drive and type the XEQ RDOS command. The drive must have been specified as part of your system during AOSGEN. Don't try to INITIALIZE the RDOS disk.

If you omit the optional rdos-directory specifier in the GET, PUT, and LIST commands, they will access the primary partition of the RDOS disk.

1. LOAD one or more RDOS files from an RDOS-format dumpfile. The .DR extension is dropped from directories and the trailing period is dropped from filenames ending in one.

   Format:

   XEQ RDOS LOAD rdos-dumpfile&↓
   &) [pathname(s) [] templates(s)]

2. DUMP one or more user data or program files to an RDOS format dumpfile. Note that you cannot dump links, directories, and file types other than user data and program.

   Format:

   XEQ RDOS DUMP rdos-dumpfile&↓
   &) [filename(s)] [template(s)]

3. GET one or more files from the primary or secondary partition or subdirectory of an RDOS disk.

   Format:

   XEQ RDOS GET/DISK = rdos-diskunit&↓
   &) [/DIR = rdos-directory] filename(s)

4. PUT one or more files on an RDOS format disk. Note that you can put files into one subdirectory or partition only.

   Format:

   XEQ RDOS PUT/DISK = rdos-diskunit&)
   &) [/DIR = rdos-directory] &)
   [templates] filenames

5. LIST the names of all the files on a RDOS format disk. This command corresponds to the RDOS CLI LIST/E/A command.

   Format:

   XEQ RDOS LIST/DISK = rdos-diskunit&)
   &) [DIR = rdos-directory] &)
   argument(s)

Function Switches:

/A	Abort on an ABORT condition.
/D	LOAD only. If a file exists with the same name, delete it then load the new one.
/DIR=rdos-directory	Access this RDOS directory.
/DISK=rdos-diskunit	Access the RDOS disk in this unit.
/L	List files on LISTFILE.
/L=pathname	List files on file specified by pathname.
/N	(LOAD and GET only). Do not load, just verify filenames.
/T	(GET and LIST only). Access all files in the specified rdos-directory and its subordinate directories (if any).
/V	Verify each file transferred.

Argument Switches:

/C	On LOAD and GET, convert carriage returns to new lines; on DUMP and PUT, convert new lines to carriage returns.
/N	Do not transfer files matching this template.

RDOSBIND

Produce an RDOS save file from one or more AOS object or library files.

Format:

XEQ RDOSBIND pathname [pathname]. . .

This utility runs on your AOS system but produces a save file that will run on an RDOS system. It requires object files and libraries as input in AOS format. You use the AOS Macroassembler and Library File Editor to prepare input to RDOSBIND.

Function Switches:

/B	Sort symbol listing both alphabetically and by value.
/D	Load an RDOS user debugger and enter global symbols in the save file.
/E	Produce a load map on @OUTPUT.
/G=n	Designate that n channels are required.
/H	Print numbers in hexadecimal.
/K=n	Allocate n TCB's (this overrides a .TSK pseudo-op).
/L	Produce a listing on the current @LIST file.
/L=x	Produce a listing on file x.
/N	Do not search the system library (ASYOB.LB).
/O	Suppress load overwrite error messages (if any).
/P=X	Create a save file named "x.SV".

FILE BACKUP UTILITIES

PCOPY –

This utility is a fast, stand-alone disk DUMP/LOAD program.

PCOPY does not dump unused disk blocks, nor any block occupied by the installed operating system (invisible space).

It is faster than using CLI commands.

It lets you dump onto successive reels of Magnetic Tape.

You can use PCOPY even if your Logical Disk has not been fixed with FIXUP.

An entire Logical Disk must be dumped or restored.

Specific directories or files cannot be selected.

A copy of the utility is kept in the util directory. The complete pathname to it is :UTIL:PCOPY.

A copy of this program has to be loaded into low core for execution to begin.

FILE BACKUP UTILITIES

INSTALLER (INSTL) –

Installs an operating system that you specify into the area of the disk which was reserved by the FORMATTER utility (invisible space).

The system must be in copy format on a Magnetic Tape or Diskette.

SORT/MERGE UTILITY

SORTING	—	The utility can do four different types of sorts. 1. STANDARD 2. STABLE 3. TAG SORT 4. STABLE TAG SORT
MERGING	—	Combines the contents of two or more sorted input files into a single output file.
REFORMATTING	—	It is possible to rearrange or delete fields of a record.
REPLACEMENT	—	The contents of selected fields can be replaced with new values.
PADDING	—	Variable length records can be converted to fixed length records. The user specifies the pad character.
TRANSLATION	—	Records can be translated from ASCII to EBCDIC or from EBCDIC to ASCII.

SECTION VII

MEMORY MANAGEMENT

SECTION VII: MEMORY MANAGEMENT

GOAL: To understand process context, specifically shared and unshared partitions, and the role of the MAP in memory management.

OBJECTIVES:

To be able to define shared and unshared memory partitions.

To be able to list several advantages of "sharing".

To be able to define the MAP and list the functions performed by it

To be able to list and to differentiate among the four ways to share information within AOS.

To be able to define Use Counts, LRU, and Memory Classes.

MEMORY MANAGEMENT

Memory is divided into pages

Each page of memory is 2K bytes

The maximum program size is 32 pages

The minimum memory requirement for a program is one unshared page.

MAP (MEMORY ALLOCATION AND PROTECTION) UNIT

The MAP units functions are:

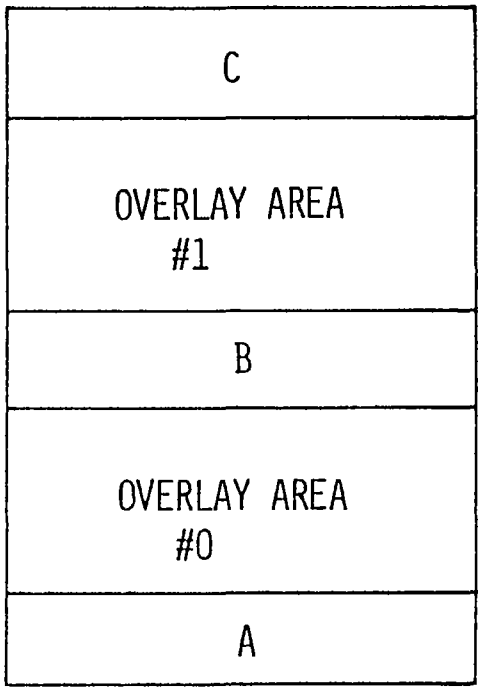
1. Address translation.

2. Prevent user program from doing direct I/O.

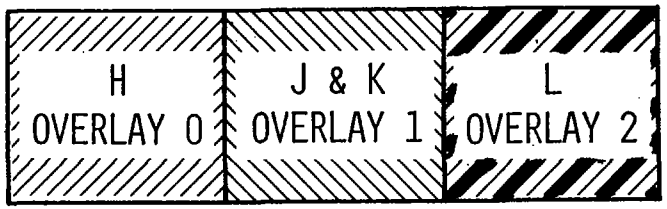
3. Write protect pages of a program.

4. Prevent more than 16 levels of indirect addressing.

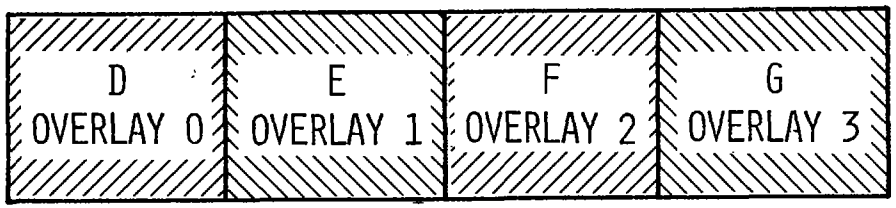
PROGRAM SEGMENTATION OVERLAYS



USER PROGRAM



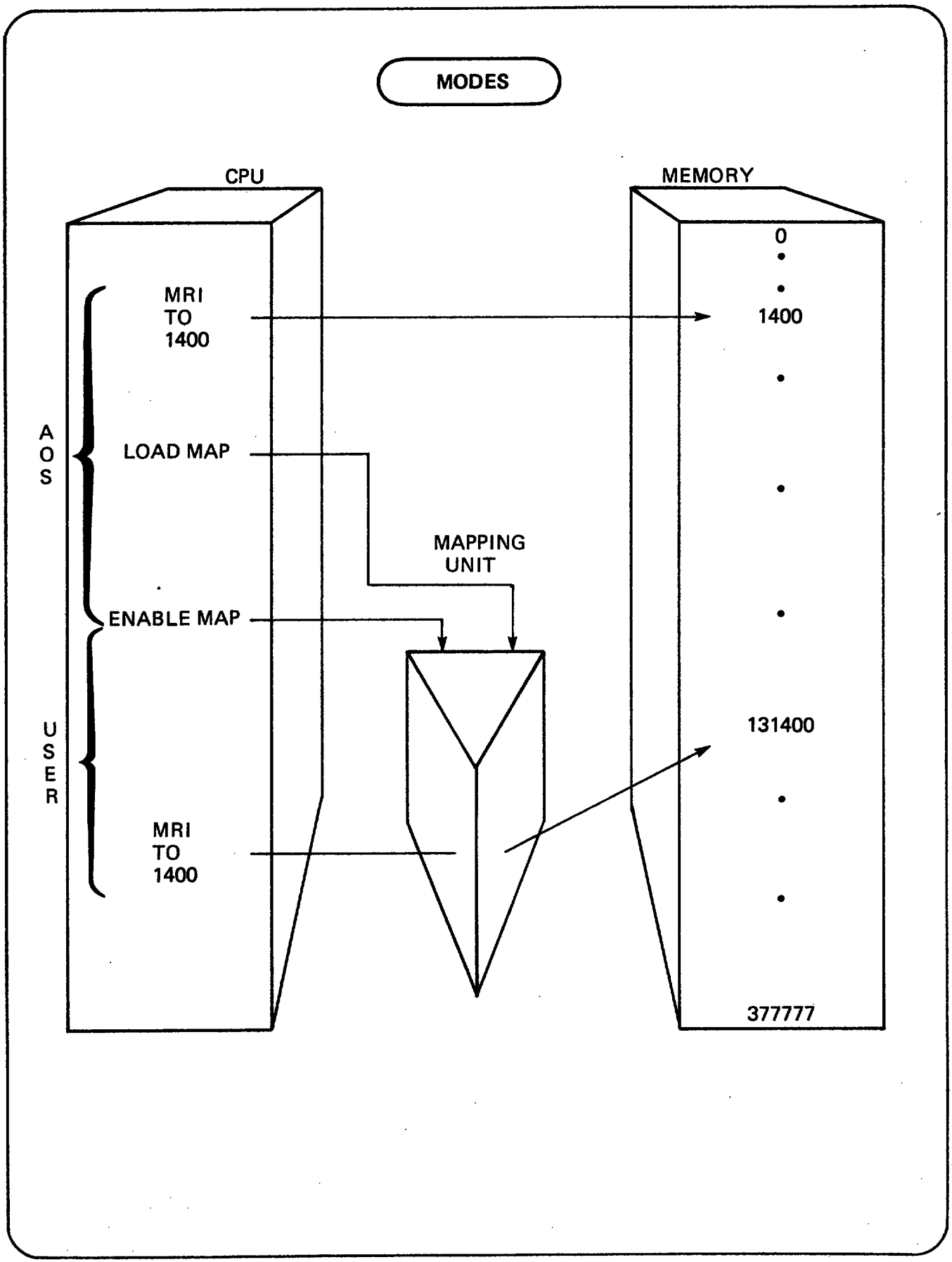
OVERLAY SEGMENT #1



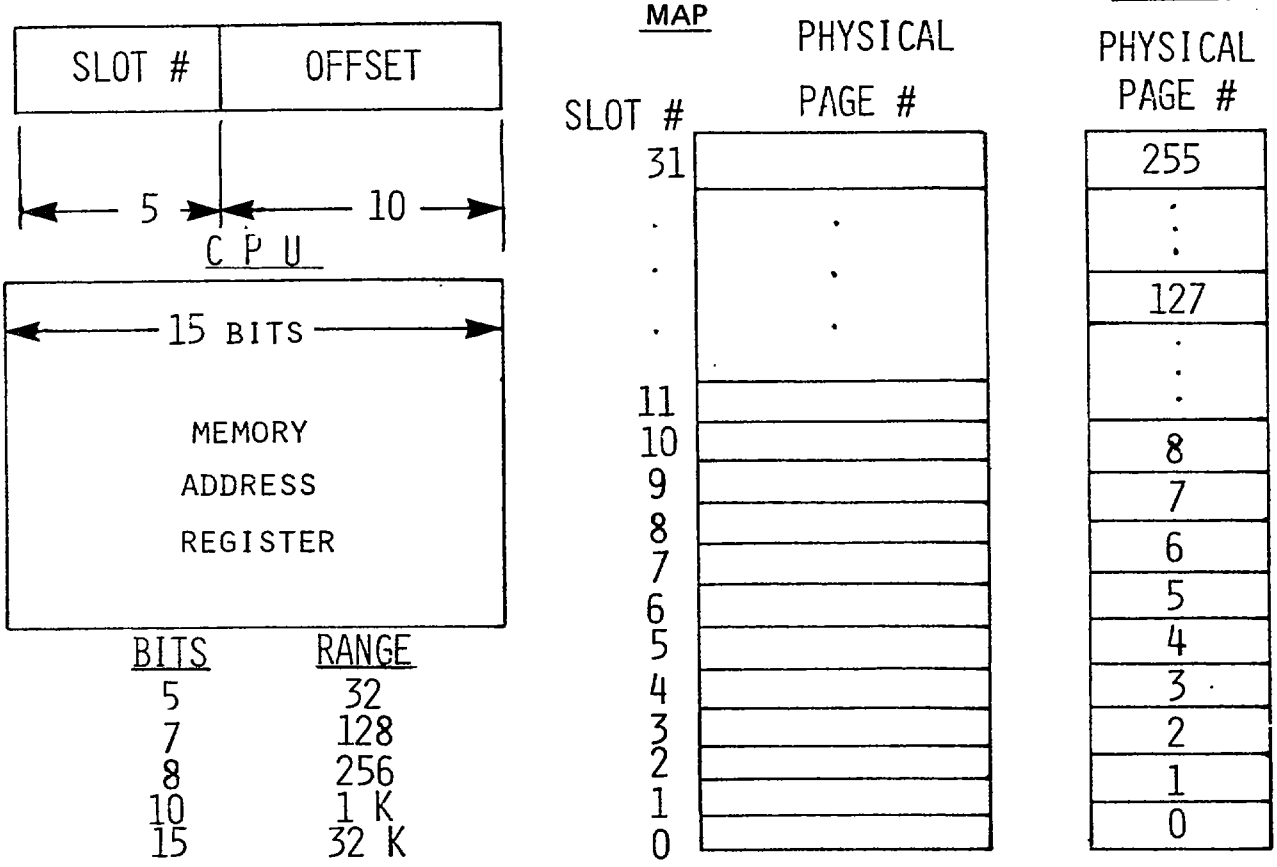
OVERLAY SEGMENT #0

PROGRAM OVERLAYS RESIDE IN CONTIGUOUS BLOCKS ON DISC.

BIND A [D,E,F,G,] B [H,J K,L] C


MODES
CPU
MEMORY
0
1400
131400
377777
MRI
TO
1400
LOAD MAP
ENABLE MAP
MAPPING
UNIT
MRI
TO
1400
AOS
USER

MAPPING



BITS	RANGE
5	32
7	128
8	256
10	1 K
15	32 K

1. REPLACE LEADING 5 BITS WITH CONTENTS OF THAT MAP SLOT
2. ADD OFFSET

IF MAP SLOT 3 CONTAINED 70_8

AND THE CPU ACCESSED ADDRESS 06022_8=000,11 0,000,010,010_2

THE PHYSICAL LOCATION IS 160022_8=1,110,00 0,000,010,010_2

UNSHARED MEMORY

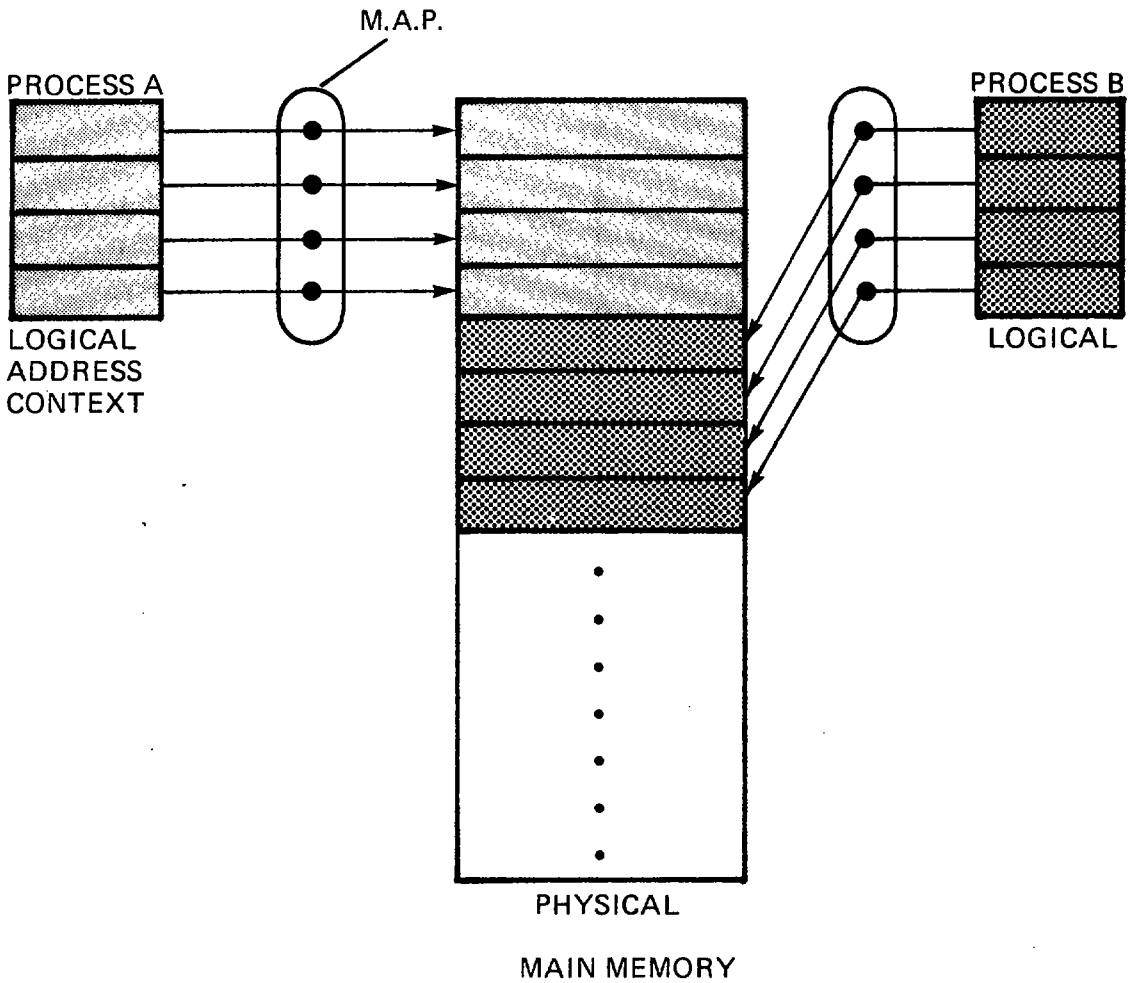
Each process has at least 1 page (2 kb) of unshared memory partition.

System data structures, unlabelled common, and lower page zero (ZREL) start at low memory addresses. This is on a per-process basis.

Although it's easy to implement, it may also unnecessarily waste memory resources

MAP APPLICATIONS

SEPARATE – UNSHARED



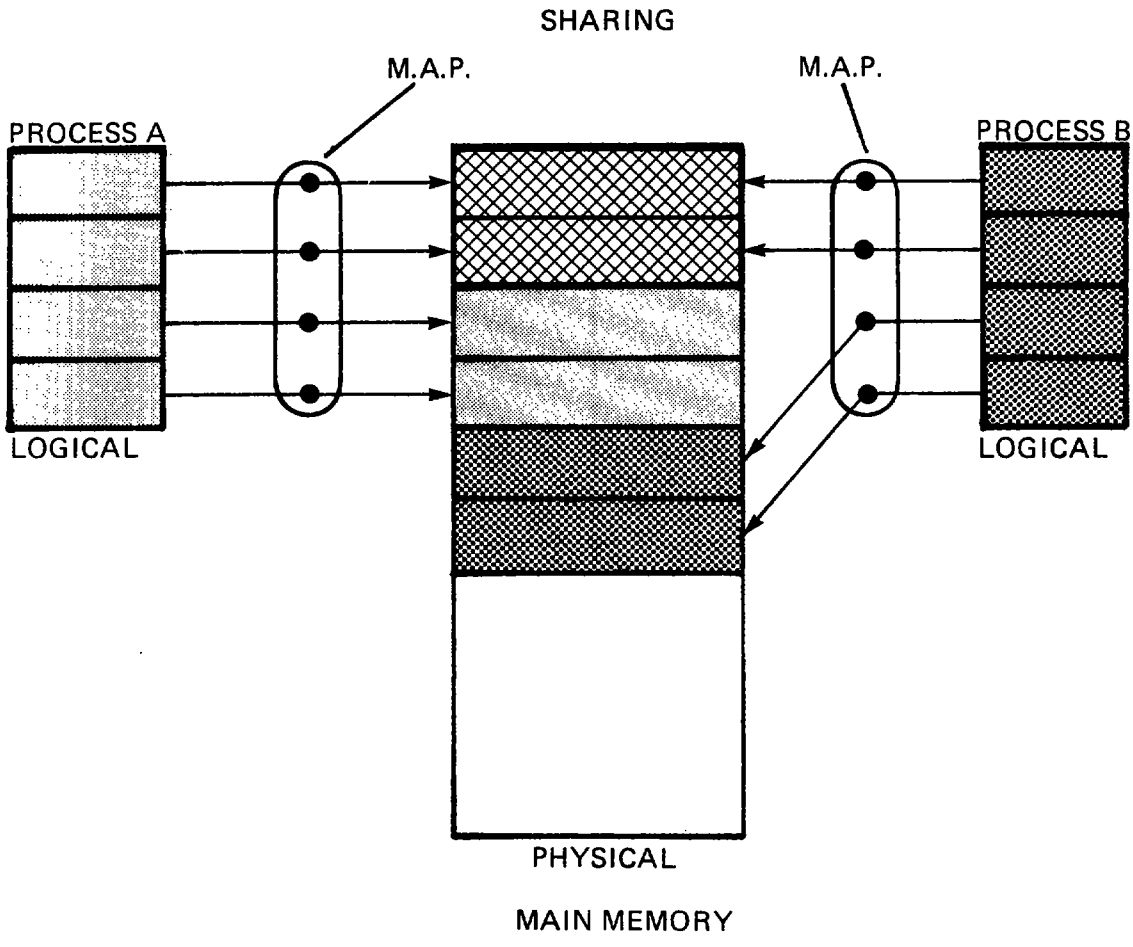
SHARING

All or portions of disk files can be read into memory pages and be shared among processes according to their access privileges.

Efficiency is the key role of sharing.

- Remapping resident shared pages reduces disk I/O.
- Read-only shared pages will not be rewritten to disk when no longer used.
- Less main memory and disk space needed for same code.
- Saves swap file space and time

MAP APPLICATIONS



SHARED INLINE CODE

PSEUDO–OP .NREL 1 SETS UP SHARED PARTITION

PSEUDO–OP .NREL 0 SETS UP UNSHARED PARTITION

AOS KEEPS TRACK OF SHARED PROGRAMS BY PATHNAME OF EXECUTING PROGRAM.

SHARED CODE WHICH GENERATES DATA SHOULD STORE IT IN UNSHARED AREAS (REENTRANT).

SHARED DATA

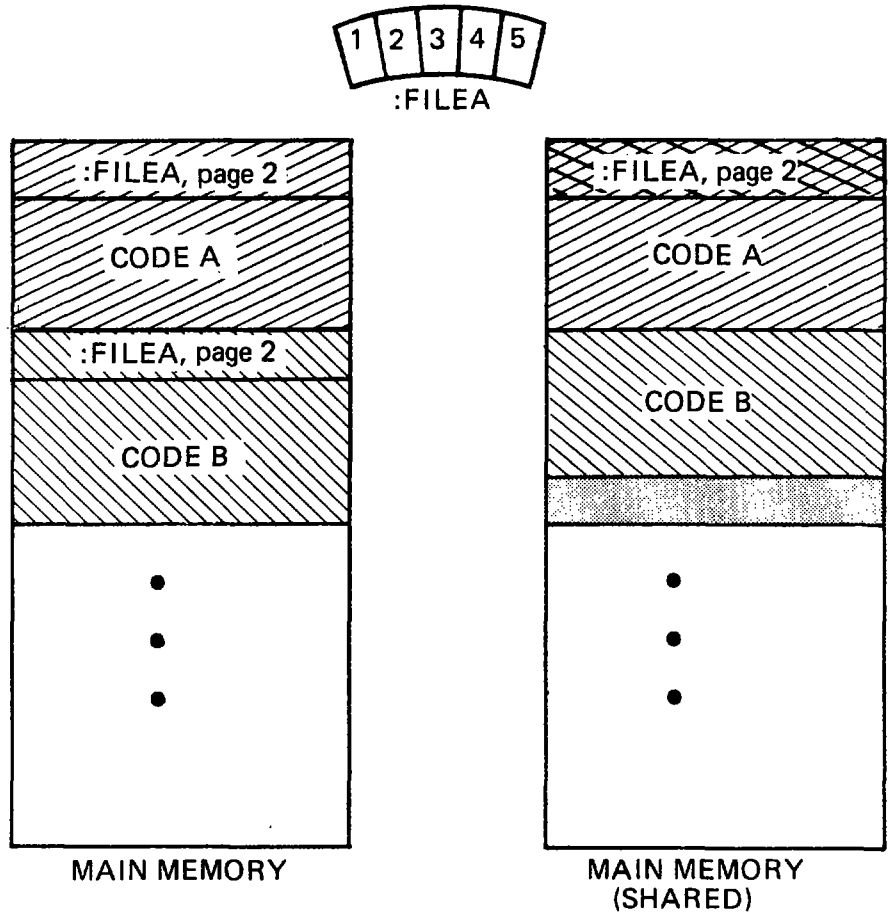
Shared data is information in some disk file, which will be used independently by two or more processes.

The data-file name is the key used by AOS to manage this facility.

Shared data can be flushed to disk without releasing the shared page.

SHARED DATA

Same data, different processes



RESOURCE CALLS

?RCALL	—	Release one resources area and acquire a new one.
?KCALL	—	Retain the calling resources area; and acquire a new one.
?RCHAIN	—	Chain to a new resource. The calling resource is released and forgotten about.

SHARED OVERLAYS

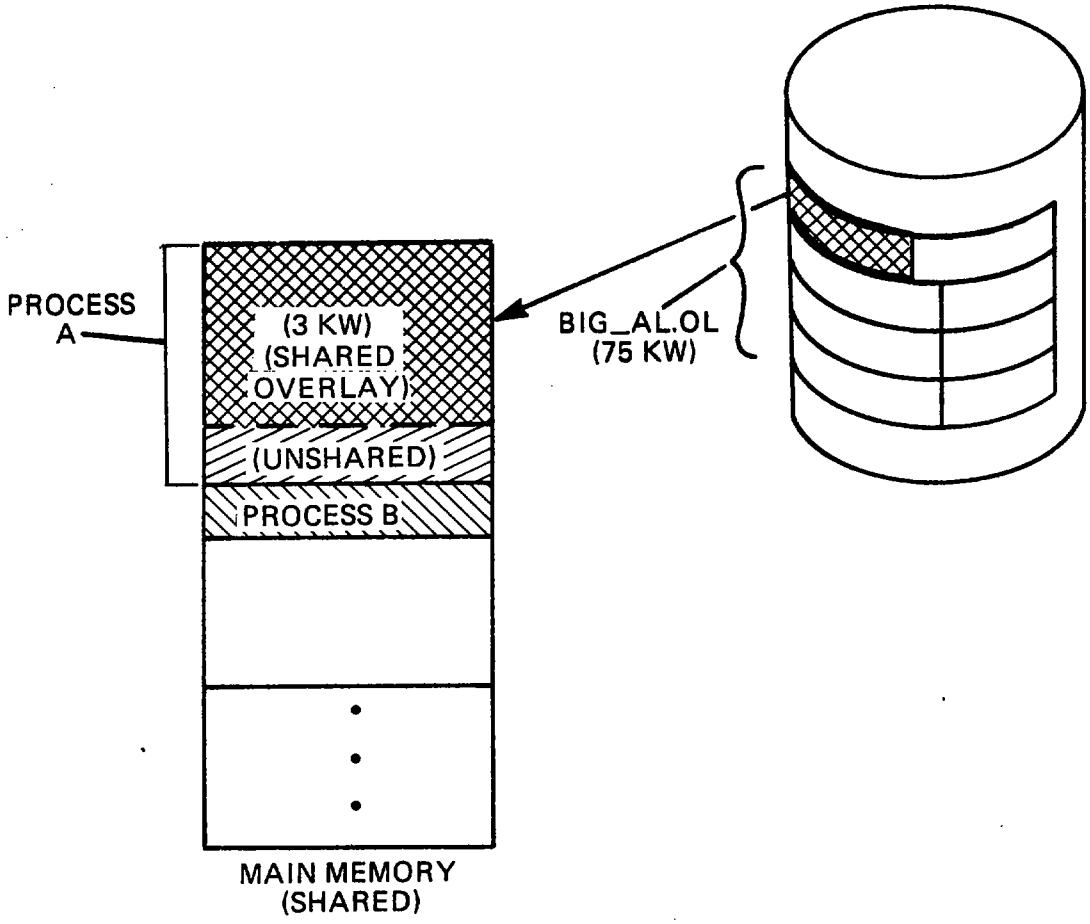
A shared overlay is one of several modules of code on disk, all of which will occupy the same logical place in the context of the process which calls it.

Resource calls (?RCALL, ?KCALL, ?CHAIN) represent a generalized procedure-call mechanism which permits programmers to access overlays.

Shared overlays are built in page multiples, should be re-entrant, and may contain other resource calls. There may be up to 511 separate overlays per overlay area created at BIND time in a program (.PR) file.

SHARED OVERLAYS

Same program (BIG_AL.PR)32K) different processes



SHARED ROUTINES

Shared routines are dynamically relocatable modules found in shared libraries.

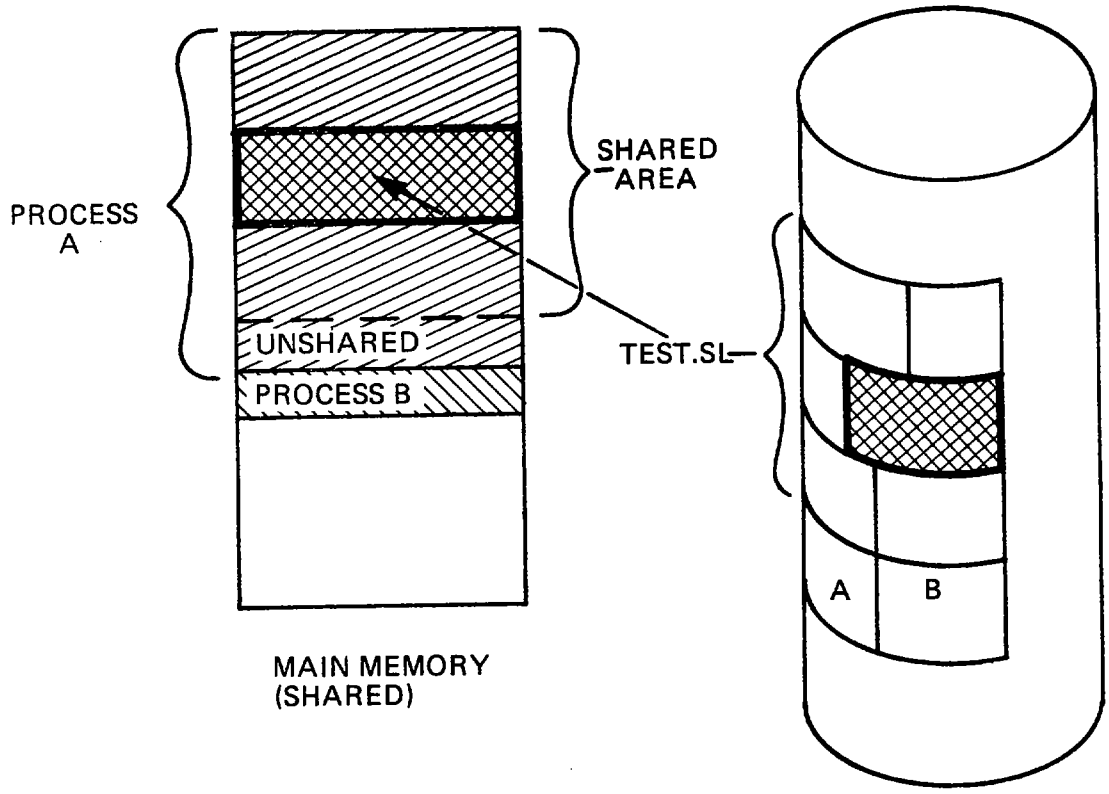
Like shared overlays, they are built in multiples of 1 page, must be re-entrant, have area reserved for them at BIND time (with /M switch), and are accessed by using resource calls.

Furthermore, they must be position-independent since they may be loaded into any shared page.

The key to AOS management is the name of each routine and the name of the shared library containing it.

Different programs may invoke the same shared routine.

SHARED ROUTINES



DYNAMICALLY RELOCATABLE MODULES FOUND IN SHARED LIBRARIES, SAME SUBPROGRAMS, DIFFERENT PROCESSES

LRU CHAIN

"USE COUNTS" ARE AOS–GENERATED NUMBERS REPRESENTING THE NUMBER OF PROCESSES CONCURRENTLY UTILIZING A SHARED MEMORY PAGE.

THE SYSTEM ALSO MAINTAINS A CHAIN THAT LISTS, IN LEAST RECENTLY USED (LRU) ORDER, ALL SHARED PAGES STILL RESIDING IN MAIN MEMORY, WHICH ARE CURRENTLY NOT BEING USED (USE COUNTS = 0)

MEMORY PAGES ON THE LRU CHAIN ARE CANDIDATES FOR REUSE, SHOULD MAIN MEMORY BE REQUIRED BY ANOTHER PROCESS.

WHEN A PROCESS IS SWAPPED TO DISK THE USE COUNT ON ITS SHARED PAGES IS DECREMENTED– AND THOSE PAGES WITH A USE COUNT OF ZERO ARE PLACED AT THE END OF THE LRU CHAIN

MEMORY CLASSES

AOS allocates needed memory pages to process in the following order:

1. Free page
2. Candidate on LRU chain
3. Part of a blocked process context
4. Part of an unblocked eligible process context

LAB GUIDE

Course Title: **S209 AOS USER**

LAB Exercise Title: **CLI FILES**

Objectives:

To demonstrate the AOS Command Line Interpreter utility.

To demonstrate the static AOS file hierarchy

To demonstrate tape backup techniques.

To demonstrate the HELP facility.

Required Materials /Equipment:

Course Title: **S209 AOS USER**

LAB Exercise Title: **CLI FILES**

CLI FILES LAB

* LOG on

* Use "HELP/V, WHO" to get a description of the WHO command. Use the WHO command.

  What's your PID? _____

  USERNAME? _____

  process name? _____

  program name? _____

* When you logged on, EXEC created a directory from the description in your profile. How much "SPACE" did EXEC give you? _____ blocks

* To see the name of your working directory input "DIRECTORY"?
  What's CLI's response? _____

* Like all CLI commands, DIRECTORY can be truncated to the minimum number of letters that CLI needs to distinguish this command from all others. What response does CLI give to just the letter "D"?

  _____

  Is "DI" long enough? _____

  What about "DIR"? _____

Course Title: S209 AOS USER

LAB Exercise Title: CLI FILES

* "FILESTATUS" shows the bookkeeping of files in the working directory. How many files does it report on in your current directory?_____

* There's not much in your directory, so focus CLI's attention on the parent of your current working directory. You already know the command to do this. The DIR command can also be used to change the working directory. "DIR, ^ " is the command to take a step closer to the root directory. ^ is either "shift N" or "shift 6" depending on your terminal. After you use the command, what is the name of the new working directory? _____

* EXEC created the User Directories Directory to hold all the directories it creates for log-on. What is AOS's response to a "SPACE" command here?

_____

* (The only difference between Control Point & regular DIRS is a limit on CPD's)

* Because of the way EXEC names directories, you could discover other people's usernames by using FILESTATUS while you're in :UDD. What happens when you try it?

_____

* The READ (R) access to a directory is simply the ability to do FILESTATUS or ACL. Since you can't find out about the contents of :UDD, try another step toward the root directory. What's the name of your new working directory?

_____

Course Title: **S209 AOS USER**

LAB Exercise Title: **CLI FILES**

* Do you have R access to this directory? _____

* You should be in the root directory (:) by now. Does another "DIR, ^ " work? _____

* You've gone from one branch of the directory tree "up" to the root. Now go "down" another branch. Use the DIR command to make :UTIL your working directory. Does FILESTATUS work here? _____

* What is the minimum truncation of FILESTATUS?

  (for DIRECTORY it's DIR) _____

* CLI normally spaces output to make it easier to read. This limits what you can fit on a screen. Input "SQUEEZE, ON". Try the "F/LE/TCR" (yes even switches can be truncated! Does the output look different?

  _____

* CLI keeps track of many values. Type "CURRENT" to see all the current values that CLI is remembering —

  What's the DIRECTORY? _____
  SEARCHLIST _____
  SQUEEZE _____

* "SQUEEZE OFF" to make the output more readable.

Course Title: **S209 AOS USER**

LAB Exercise Title: **CLI FILES**

* Another way to control output is by "control keyes". Holding the "CTRL" key down and hitting "S" will stop output. "CTRL" and "Q" continue it. For the next FILESTATUS, use the control keyes to look at the output before it gets bumped off your screen. According to HELP/V for the FILESTATUS command, what values does the ASSORTMENT switch show?

_____

* Now try F/AS and use CTRL S & Q to stop output before it runs off your screen.

* FILESTATUS can report on specific files if you use the files' pathnames as arguments to the command. In fact CLI lets you use templates to easily specify groups of files in such commands. The "+" template matches any number of characters. For example to get the bookkeeping info on all "program" files, input "F , + .PR".

  What did all the files shown have in common?

_____

* How many files start with any characters at all but end with the characters .CLI? _____

* ACCESS CONTROL LIST
  (each file and each directory control which usernames may access it)

Course Title: **S209 AOS USER**

LAB Exercise Title: **CLI FILES**

* Another command that accepts templates is the ACL command. Input "ACL, +.CLI". Did the output tell you which access control list belonged to which file?_____

* HELP can tell what switch to use to include file names in ACL's output. What switch is it? _____

* Use the switch with ACL to discover the ACL's of the .CLI files. Notice the use by ACL of templates. What names besides filenames can templates be used for? _____

* Pick a .CLI that doesn't allow your username access.

  (Remember + includes your name)

  Try using this file as an argument to the TYPE command. What error message appeared?

  _____

* Try TYPEing a file you can read. Did it work? _____

* "R" access lets you duplicate files as well as TYPEing them. Use the file you can read in the following command: "COPY, NEW.NAME, can-read-file"

  What happened? _____

Course Title: **S209 AOS USER**

LAB Exercise Title: **CLI FILES**

* **Try the COPY command again, but with the new file in your directory — "COPY,:UDD:username:new, can-read-file"**

  It should work, did it? _____

* **Just because you have only one working directory doesn't mean that you can't use others. As long as you have access rights and the time to input long pathnames, you can affect any number of directories.**

* **Try "COPY, :UDD: your-dir:ANOTHER,can't-readfile"**
  What message? _____

* **COPY only creates one file at a time. To move copies of all the +.CLI files and their bookkeeping, input "MOVE/NACL,:UDD:your-dir,+.CLI"**
  Which files didn't work? _____
  Why? _____

* **According to HELP, what does /NACL do to the MOVE?**

  _____

(default-current processes username)

Course Title: S209 AOS USER

LAB Exercise Title: CLI FILES

* Use the DIRECTORY command to get back to your original working directory. Do a FILESTATUS. Notice that the names and other bookkeeping for the files has been moved as well. Is today's date the date of creation for the files in your directory?_____

* Note also that even the files that denied you R access were reported on in FILESTATUS output from your directory. This shows the 2 step nature of the MOVE command. First the bookkeeping was transferred. That worked because you had read access of the bookkeeping in :UTIL. Only the contents of the files were protected. How long are your copies of the files you MOVED for which "READ ACCESS" was "DENIED"

_____

* COPY and MOVE are used to duplicate files for immediate use. For backing up a disk in case of crash use DUMP & LOAD

* DUMP all your macro files (+.CLI) into a file called LIKE.TAPE (normally you would dump to a mag tape file).

* To check which files have been put in the file, try LOAD/N of LIKE.TAPE

* "DELETE/C, +.CLI" will ask you which of the .CLI files to erase. Delete all but 2 of the files.

* LOAD/V only those macros accessed before JANUARY 1.

Course Title: **S209 AOS USER**

LAB Exercise Title: **CLI FILES**

* LOAD one of the files in LIKE.TAPE by using that files name as an argument.

* Now load the entire contents of LIKE.TAPE by "LOAD LIKE.TAPE" What messages did you get?

_____

* What is the length in bytes of the LIKE.TAPE file? _____

* How does its length compare with the sum of the lengths of the +.CLI files?

_____

* Delete about half of your .CLI files. Now LOAD/R from LIKE.TAPE. The /R switch tells CLI that you expect to find some files already existing when CLI tries to LOAD them. Only now CLI won't twll about it - you've told CLI to compare creation dates and load the most recent. This switch is used to replace system files from UPDATE mag. tapes.

Course Title: **S209 AOS USER**

LAB Exercise Title: **MACROS**

Objectives:

To demonstrate the flexibility and power of the AOS macro facility

To demonstrate the CLI "environment" feature.

Required Materials /Equipment:

Course Title: **S209 AOS USER**

LAB Exercise Title: **MACROS**

CLI MACRO LAB

* Log on

* Push the current environment

* "PROMPT,TIME,DIR" will add the time of day and working directory to each prompt. This will help you keep track of where you are.

* CLI by default sends all reports to its OUTPUT file. EXEC defines OUTPUT as the console you logged on at. You can send these reports to a LIST file instead by using the /L switch. A LIST file is a good way to produce written reports about the disk. Try "WHO/L".
  What happened? _____

* EXEC doesn't set up a LIST file for you. Use the CREATE command to build a file called HOLD.LIST. What directory's is HOLD.LIST in? ("PATHNAME,HOLD.LIST")

  _____

* How long is HOLD.LIST? _____

* Use HOLD.LIST as an argument for the LIST FILE command and verify the setting by looking at the current environmental value.

Course Title: **S209 AOS USER**

LAB Exercise Title: **MACROS**

* **Try "WHO/L" again. How long is HOLD.LIST now? _____**

  CLI holds /L output until you have a block's worth, then send it to the file.

* **Change your working directory to :UTIL. Was the change shown in CLI's prompt? _____**

* **"F/AS/L;ACL/V/L" gets CLI to write to HOLD.LIST.**

* **Take a look at what's in HOLD.LIST. Try "TYPE,HOLD.LIST"**
  What happened? _____

* **Whenever you give AOS a filename rather than a pathname (include directory names), AOS first looks for it in the working directory. Is HOLD.LIST in the working directory?**

  _____

* **Try telling AOS the whole pathname (:UDD:dir:HOLD.LIST)**
  Did it work this time?

  _____

Course Title: **S209 AOS USER**

LAB Exercise Title: **MACROS**

* For some CLI commands, there is an alternative to constantly changing working directories or endless typing. After not finding a filename in the working directory, AOS checks all the directories on the searchlist before giving up. What directories are currently on the "SEARCHLIST"?

_____

* Is the directory containing HOLD.LIST on the searchlist?

_____

* Like most environmental value, there is one command to both display and set the current searchlist. Change the searchlist to include the directory that HOLD.LIST lives in. Try TYPEing HOLD.LIST. Was it found? _____

* Input "CURRENT/L" to get a report of all the environmental changes you've made. Now pop back to level 0. Did you notice any immediate change?

_____

* "QPRINT HOLD.LIST" Why was HOLD.LIST easy for AOS to find?

_____

* Compare current environment values with those printed from HOLD.LIST. What values differ?

_____

* Does the print out include the CLI prompts & the commands you gave or just CLI's responses?

_____

Course Title: **S209 AOS USER**

LAB Exercise Title: **MACROS**

ERROR REACTION

* Create a subordinate process that takes over your console & runs CLI "X CLI"

* What is your PID? _____

  process name? _____

* This new CLI is for safety. If you abort it, you will simply return to your original CLI. If you had aborted the original CLI, you would have been logged off. To keep things straight set STRING to SAFE-CLI and set PROMPT to STRING.

  "STRING,SAFE-CLI;PROMPT,STRING"

* Set CLASS1 (environmental errors) to IGNORE. Input the following series of 4 commands: "POP;DIR;DATE;TIME" The POP command is wrong because you haven't pushed anything. Did CLI report this error to you? _____

  Were the remaining commands executed? _____

* Set CLASS1 to WARNING & input the command series.

  Error reported? _____

  Other commands executed? _____

Course Title: **S209 AOS USER**

LAB Exercise Title: **MACROS**

* **Set CLASS1 to ERROR. Input the series.**

  Error reported? _____

  Others Done? _____

* **Set CLASS1 to ABORT. Input series.**

  What happened? _____

* **Aren't you glad you had a safety CLI?** _____

MACROS

* **It was a bother typing in that command series over and over. Anytime you want to repeat CLI commands, you should write a macro (a CLI mini program) to save typing. Try a macro called TRIO –**

```
"CREATE/I TRIO.CLI
DIRECTORY
DATE
TIME
)"
```

Course Title: **S209 AOS USER**

LAB Exercise Title: **MACROS**

The final right parenthesis is just to tell the insert (/I) mode of the CREATE to stop.

Input "TRIO". Did it work? _____

(it should)

* **Whenever CLI doesn't recognize a command, it tacks a "CLI" on the end and searches for a file by that name. Try inputing "STRANGE"**

What messages appeared? _____

* **Create a macro called SHOW.CLI that displays working directory, searchlist and assortment of file bookkeeping info. Test it in your directory. How many CLI commands does it contain?** _____

* **PUSH and change working directory to :UTIL. Input "SHOW" to try your macro here.**

What happened? _____

* **Remember that macros are files and for AOS to find them either the working directory or to search list has to contain the macro files directory. Change the searchlist to SHOW.CLI's directory. Try "SHOW" again. Did it work?**

_____

Course Title: S209 AOS USER

LAB Exercise Title: MACROS

* POP back to your directory. Change SHOW.CLI's access control list to give your username just OE. Try SHOW.

  What access must you have to use a macro? _____

  Change SHOW.CLI's ACL to allow you to use it.

* As a part of running SHOW in :UTIL, you PUSHed, changed directory and searchlist, ran SHOW and POPed back. Write a macro to do this for you but write it to go into any directory, not just :UTIL. To do this, you'll have to use a "macro argument" in the DIR command.

```
"CRE/I  LOOK.CLI
PUSH
SEARCHLIST :UDD:your-directory
DIRECTORY %1%
SHOW
POP
) "
```

When you use this macro be sure to give it one argument -- the name of the directory to look at. %1% is replaced by the first argument after the macro name.

Try "LOOK :UTIL"
Try "LOOK :PER"
Try "LOOK :HELP"
Try "LOOK :PROC" (can't win em all)

Course Title: **S209 AOS USER**

LAB Exercise Title: **MACROS**

* PUSH and POP are used in macros that change environmental values while they're running. This way the macro can leave things as it found them and still have the freedom to temporarily change them.

* Once you write a macro and set up the proper access lists and searchlists, you can use the macro like any built-in CLI command. You know that one macro can call another — LOOK calls SHOW. You will write a macro that calls itself.

```
"CRE/I   LOOP_TIME.CLI
TIME
PAUSE 1
LOOP_TIME
) "
```

* Now run it. It won't stop itself - you have to interrupt it. Hold down CTRL and hit C. Now hold down CTRL and hit A.
What message appeared? _____

* Looping macros should be to stop themselves when conditions are right. For example you will write a macro that you call with a bunch of arguments. It will call itself but drop an argument each time it does. When it finally runs out of arguments it will stop. To make the tests, use conditional psuedo-macro :EQUAL. Get HELP to describe this command.

```
"CRE/I  LOOP_STOP.CLI
[:EQ,,%1%]
WRITE DONE&
[!ELSE]
WRITE  ARGS  LEFT  %2-%
LOOP_STOP %2-%
[!END]
) "
```

Run LOOP_STOP with different numbers of arguments. The :EQ line tests to see if the first argument (%1%) is there or not (equal to nothing).

* Write a macro called LOOP_BANG.CLI that if you call like "LOOP_BANG,3,2,1" will write out:

```
SELF DESTRUCT IN 3 SECONDS
SELF DESTRUCT IN 2 SECONDS
SELF DESTRUCT IN 1 SECOND
BANG!
```

* Write another one that outputs "ONE SECOND LEFT" instead of "SELF DESTRUCT IN 1 SECOND"

Course Title: **S209 AOS USER**

LAB Exercise Title: **MACROS**

HINT: You need a separate [!END] for each [!EQ]

Besides just testing for the existence of an argument you can test for its value - [!EQ,%5%,A] tests to see if the fifth argument is the letter "A" The %2-% represents the second and all following arguments.

Course Title: **S209 AOS USER**

LAB Exercise Title: **GENERIC FILES**

Objectives:

To demonstrate the existence and flexibility of the AOS generic files' feature.

To demonstrate the AOS feature, via the CLI, of process creation.

Required Materials /Equipment:

Course Title: **S209 AOS USER**

LAB Exercise Title: **GENERIC FILES**

GENERIC FILES LAB

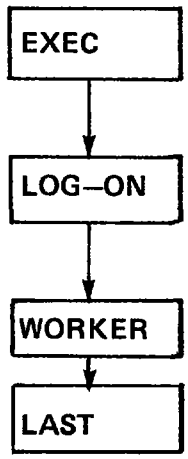
* You will need the "create without block" privilege in your profile. Additionally, your profile should permit you at least 2 sons.

* When you log on, you get the same CLI code as everyone else. How does CLI keep track of what terminal you're at? It doesn't! AOS remembers 5 "GENERIC FILES" for each process: @INPUT,@OUTPUT,@CONSOLE,@DATA and @LIST. When CLI needs a command, it gets it from "whatever the @INPUT file is for my process". The same request from different processes uses different files. In this lab you'll work with 3 levels of processes. The first will be the process created by EXEC when you log on. I'll call this your LOG–ON process. Next you will create a process yourself called WORKER because you'll do most of your work through this one. The third level I'll call LAST.

EXEC is allowed "unlimited SONS"

LOG–ON is allowed 2 SONS in your profile

When you create this, the PROC command allows WORKER a single SON



Course Title: **S209 AOS USER**

LAB Exercise Title: **GENERIC FILES**

* Generic files can be set by a parent process when it creates its offspring. EXEC sets your @INPUT,@OUTPUT and @CONSOLE files to the console you logged in at. the PROCESS command has switches to control the generic files of processes you create. What does the /IOC switch do?

_____

(you may need help for this one)

* Try "PROC/IOC,CLI" to create an offspring CLI under your LOG-ON. What message did you get?

_____

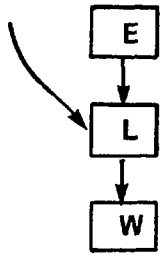
* Only one active process can have your terminal as a generic file. The only way that a parent can "share" generic files with an offspring is to BLOCK itself until the offspring finishes. Add a /BLOCK and a "/DEFAULT" AND "/SONS=1" to the PROC and try again. This is similar to the X command. What is WORKER's PID? _____

* Use "TREE" to find LOG–ON's PID. (father of WORKER process)
Use LOG–ON's PID as an argument to RUNTIME.
How much CPU time has it used so far? _____

Course Title: **S209 AOS USER**

LAB Exercise Title: **GENERIC FILES**

* Use LOG---ONs PID as an argument to WHO.
  What is your original process' name? _____
  CLI doesn't tell you what your generic files are so its great that EXEC names your log on process after those files



* A /IOc can also be used to give the offspring a set of generic files different from those of the parent. Try "PROC/IOc=@CONO, CLI" What message did you get?

  _____

* @CONO is already owned by another process. Once a process owns a device, no one (not even OP) can grab it away. There aren't many free consoles, so use disk files as generic files. The program won't mind the difference. Create a file called DO.IT. (no .CLI - this isn't a macro). In DO.IT set the prompt to time of day, find out who you are, get the current environment, get your family tree, pause 10 seconds, get an assortment of sorted filestatus information and finally pause another 10 seconds. Now "PROC/IN=DO.IT,CLI"
  What happened? _____

  The LAST CLI has died but you're not told until you ask about its termination. Input "CHECKTERMS". What was the problem?

  _____

Course Title: **S209 AOS USER**

LAB Exercise Title: **GENERIC FILES**

* CLI must have other generic files to run. Add a "/OUTPUT" and "/CONSOLE" (WORKER process' generic files to the LAST process.) What other switch must you add as well?

_____

* Did copies of the @INPUT files (DO.IT's) commands show up on your console?

_____

* CLI sends all prompts to @OUTPUT. AOS automatically sends @OUTPUT a copy of everything any program reads from @INPUT. Did you get a message telling you when the LAST process stopped?

_____

_____

* What is your current CLI's response to "CHECKTERMS"?

_____

* A normal death isn't reported in CHECKTERMS.
Set your prompt to checkterms to get automatic reports.

* If you PROC a CLI with @OUTPUT going to a disk file, you won't have to block your current CLI to do it - no fight for generic files. Try "PROC/IN=DO.IT /OUT=SHOW.OUT,CLI" What message did you get?

_____

Course Title: **S209 AOS USER**

LAB Exercise Title: **GENERIC FILES**

* Create SHOWOUT and try it again. What message did you get now?

_____

* Do a "TREE" to verify that the non-console CLI (LAST) is running. Do a RUNTIME on the new (LAST) PID. How many I/O blocks has it moved?

_____

* How long is SHOW.OUT? _____

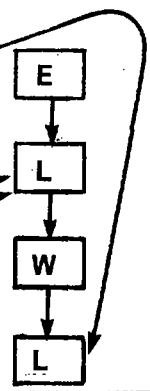
* When your lowliest process dies, type SHOW.OUT. How does this file differ from what you would see on a screen if you had just typed in the commands of DO.IT?

_____

* CLI sends the same stuff to @OUTPUT no matter where it leads. You can use a disk file driven CLI to automate repeated actions - like backing up all new files every hour or monitoring the activity of other processes. Write a macro called SPY.CLI. Have it get the time, get the runtime statistics of WORKER's PID, pause 10 seconds and call SPY. SPY has to be a macro to allow you to repeat it by simply inputing "SPY" as the last command. This spies on WORKER every 10 secs. PROC a CLI with SPY.CLI as @INPUT and SHOW.OUT as @OUTPUT. What's the new PID? _____

* Use "TREE" to verify the new process (SON) what's PID of your LOG-ON process (FATHER)? _____

* Get runtime statistics of your original process. How much CPU time now?



Course Title: **S209 AOS USER**

LAB Exercise Title: **GENERIC FILES**

* This original process has been blocked waiting for your current process to die and return the generic files. How much CPU time has it used since the first RUNTIME you did on it (page 2)? _____

* Try to kill your LOG–ON process by using its PID as an argument to the TERM command. What message did you get?

_____

* Normally a process can only kill itself and its offspring (not parent). Besides, when a process goes, it takes its kids with it. Use the TERMINATE command on WORKER process ID. Who are you now?
PID _____ process name _____

* What does CHECKTERMS return? _____

* TYPE a copy of SHOW.OUT to all what LAST's CLI reported about WORKER's CLI activity. Changes in the report show that both LAST and WORKER were running during the same period. The privilege to create without block allows you to do this.

Course Title: **S209 AOS USER**

LAB Exercise Title: **BATCH**

Objectives:

To demonstrate the EXEC "Batch" facility.

To demonstrate the CLI commands which interface to the Batch facility.

Required Materials /Equipment:

Course Title: **S209 AOS USER**

LAB Exercise Title: **BATCH**

BATCH LAB

EXEC treats BATCH jobs almost exactly the same as log ons. EXEC creates a process to run a program according to a user profile. The only differences involve the generic files and just when the process will be created. With log on, CLI talks to a console, at BATCH it talks to disk files. At log on you get a process right away, at BATCH the operator decides when a process is to be built.

* Because you're profiled to run CLI, any job you submit to BATCH has to be made up of CLI commands. This isn't so bad - you can create other processes to run other programs with the X and PR C commands. CREATE/I a file full of CLI commands to submit to BATCH. Call it BTEST. In it look at CLI's starting condition and investigate the files in the utility and peripheral directories:

```
)CR/I BTEST
))PUSH
))WHO
))CURRENT
))DIR :UTIL
))F/ELE/INDEX ; ACL/V/L,+
))DIR :PER
))F/L/AS
))POP
)))
```

Course Title: **S209 AOS USER**

LAB Exercise Title: **BATCH**

* "QSUBMIT/HOLD,BTEST" What message did you get? _____

_____

* Do a "QDISPLAY". What Q name is your request held in?

_____

(i.e., LPT,MOUNT,BATCH_?)

* Try "QDIS/V/QUE=BATCH_INPUT" What kinds of information does the Q keep about your request?

_____

_____

* Try QSUBMITing a file that is empty.

"CRE,NOTHING ; QSUBMIT/HOLD,NOTHING"

What sequence # ? _____ Now, "DEL/V,NOTHING"

* Use the "QUNHOLD, sequence#" to ready both requests. the results will be printed under the BATCH_OUTPUT and BATCH_LIST spooling Qs. Look at BTEST's listing - What is the INPUT file? _____

OUTPUT file? _____

LIST file? _____

Course Title: **S209 AOS USER**

LAB Exercise Title: **BATCH**

* **EXEC likes to create files with long, strange names for you. If you only want to do a BATCH job once, you can get EXEC to create even your initial command file for you.**

```
"QBATCH/V/I
WHO
CURRENT
F/TLA
)"
```

What is the name EXEC makes up for you?

_____

* **Where do you think the numbers in the name come from?**

_____

* **This job had no "/L" switches on any command. Did @LIST get printed through BATCH_LIST?** _____

* **What differs between the CURRENT under BATCH and under console log on?**

_____

Course Title: **S209 AOS USER**

LAB Exercise Title: **BATCH**

* What command did CLI add to the three you QBATCHed?

_____

* Even though CLI never tells you what its @/INPUT,@OUTPUT and @CONSOLE files are, CLI can find out for itself. By asking AOS about generic files, CLI can tell if it was logged on (=TOC defined) or BATCH (no@CONSOLE,but@LIST).

* An important part of each BATCH request is a USERNAME. EXEC uses it to locate a profile. In the requests you have submitted, did you include your username as an argument? _____

EXEC got the username from the CLI you did the Q stuff from.

* There is a way to make BATCH requests other than through a CLI already being charged against your username. A program called STACKER can submit jobs for many different usernames (its username is usually OP) STACKER owns a file like card reader, remote console, disk file. Through this file can flow any number of BATCH job files. To use the STACKER, you have to include your username with each job so EXEC can find your profile - STACKER won't use its own username. Create a file for stacking - call it "STACKED" and fill it with the following:

```
"$$JOB ,                    your-username
 $$PASSWORD ,               your-password
   WHO
   CURRENT
   DIR :HELP
   F/LEN
$$END"
```

Course Title: **S209 AOS USER**

LAB Exercise Title: **BATCH**

* Because BATCH is a job for EXEC, to start STACKER you have to tell EXEC what to do. EXEC only accepts such commands through IPC's from processes with EXEC's username. Go to @CON0 (or any process running CLI under username "OP") Now "CONT,@EXEC,STACK, :UDD:your dir:STACKED" you'll get a bunch of messages from EXEC using the SEND command. If you get illegal sort or job not found - something went wrong. Rip off the responses and keep with this lab.

* What filename did the stacker make up for your request in BATCH_INPUT? _____

* If you had put 5 job requests in STACKED, stacker would have created 5 separate BATCH_INPUT request files for you. One per $$JOB. From the printout, what was your BATCH processes' files for:
  @INPUT _____
  @OUTPUT _____
  @LIST _____

* STACKER killed itself when it tryed to continue reading your file and AOS reported END-OF-FILE. For a card reader STACKER would wait for more cards to be put in the hopper till a multipunch-in-column1 (EOF for card reader) "Control D" is console EOF. Type in a couple "CONTROL D"s.

LAB GUIDE

Course Title: **S209 AOS USER**

LAB Exercise Title: **EDITORS**

Objectives:

To demonstrate the usefulness and procedure for using an AOS editor utility.

Required Materials /Equipment:

Suggest that the student use whichever editor they are unfamiliar with. Both handouts modify the same lab file with the same intended result.

Course Title: **S209 AOS USER**

LAB Exercise Title: **EDITOR**

EDITOR LAB "A" -- LINEDIT UTILITY

Bring a copy of lab text into your EXEC profile working directory :AOS.LABS:TEXTEDIT.LAB. Lab text may be found in file.

Execute the line editor program. "X LINEDIT your-filename"
Notice that a new prompt (?) is now used.

The text file is made up of how many pages? _____
Combine these pages into one page.

The Text is FORTRAN source code. Any line beginning with a 'C' is a comment line. Numbers on the extreme left hand side of a line are labels. All labels should begin in columns one or two. The actual commands should start in column seven.

The following changes need to be applied to the text.

1. **The label '12' should be changed to '21'**

2. **After the line labeled '200' add the following lines:**

col1	col7
	READ (11) IR
	IF (IR) 300,300,5
300	END

Course Title: **S209 AOS USER**

LAB Exercise Title: **EDITOR**

3. **Change all occurrances of 'INTRREREST' to 'INTEREST'**

4. **The line immediately preceding line '100' is really two lines and should read:**

col1	col7
	GO TO 22
50	WRITE (10,100) PAY

5. **The line P,R,L,IFULL should really be**

   READ (11) P,R.L,IFULL

6. **Following the line 'C CHANGE LOAN LIFE TO MONTHS' add the line:**

col1	col7
	N= 12*L

Read the PROGRAM comments carefully then Compile, Bind and Execute the program.

Course Title: **S209 AOS USER**

LAB Exercise Title: **EDITOR**

EDITOR LAB "B" – SPEED UTILITY

Bring a copy of lab text into your EXEC profile working directory. Lab text may be found in file AOS.LABS:TEXTEDIT.LAB

Execute speed against the file you just copied.

Notice that a new prompt (!) is now used.

The text is FORTRAN source code. Any line beginning with a 'C' is a comment line. Numbers on the extreme left hand side of a line is a label. All labels should begin in columns one or two. The actual commands should start in column seven.

The following changes need to be applied to the text:

1. The label '12' should be changed to '21'.

2. After the line labelled '200' add the following lines:

```
                READ (11)  IR
                IF (IR) 300,300,5
300             END
```

3. Change all occurances of 'INTRREREST' to 'INTEREST'

Course Title: **S209 AOS USER**

LAB Exercise Title: **EDITOR**

4. **The line immediately preceding line '100' is really two lines and should read:**

```
                GO TO 22
50              WRITE (10,100) PAY
```

5. **The line P,R,L,IFULL should really be:**

```
READ (11) P,R,L,IFULL
```

6. **Following the line 'C CHANGE LOAN LIFE TO MONTHS' add the line:**

```
N=12*L
```

Read the comments carefully then Compile, Bind and Execute the program.

Course Title: **S209 AOS USER**

LAB Exercise Title: **LFE (LIBRARY FILE EDITOR)**

Objectives:

To demonstrate the existance, value, and proper usage of the LFE Utility.

Required Materials/Equipment:

AOS files	**WRONG.LB**
	TEST.OB

Course Title: **S209 AOS USER**

LAB Exercise Title: **LFE (LIBRARY FILE EDITOR)**

LFE (LIBRARY FILE EDITOR) LAB

NOT SHARED ROUTINE LIBRARIES!

* MOVE :AOS.LABS:WRONG.LB to your directory & DIR there.

* "X, LFE/L=@LPT,A,WRONG.LB" will run the LFE to Analyze the library. Remember that externals (XT) should appear before the entries (EN) that define the externals. In a good library the hooks (XT) have to be out before the fish (EN) swim by. A "P"hase error indicates a badly ordered library. Which entries have this "P" error reported?

_____

* Entry points are used to catch object modules. What entries (EN) are defined in the mods of WRONG.LB?

_____

_____

* Titles (T) of library entries are only important during the construction of libraries. What titles do you find here?

_____

(LFE always sends output to the listfile)

* "X,LFE,X,WRONG.LB,title" extracts the module whose title you gave. Extract <u>all</u> the modules.

Course Title: **S209 AOS USER**

LAB Exercise Title: **LFE (LIBRARY FILE EDITOR)**

* **"X,LFE,N,RIGHT/O,filea.OB,fileb.OB,..." builds a new library from the object files listed. Build a new library from the extracted mods, but put them in the correct order (i.e., no phase errors).**

  What order did you put the titles in?

  _____

  _____

* **"X,LFE,T,RIGHT.LB" will verify the order.**

* **MOVE :AOS.LABS:TEST.OB to your directory. Do two binds-one with the good library one with the original library. First-a BATCH job for original "QBATCH/V ,X,BIND/L,TEST,WRONG.LB". Now set your listfile to LPT and "X,BIND/L/B,TEST,RIGHT.LB"**

  What errors were reported from the BATCH BIND?

  _____

  _____

  _____

* **Do an LFE analyze on :UTIL:URT.LB**
  Modules from this library are a part of every user program.

Course Title: **S209 AOS USER**

LAB Exercise Title: **DISK EDIT**

Objectives:

To demonstrate the value of, and procedures for using, the AOS Utility "DEDIT" (Disk File Editor).

Required Materials /Equipment:

AOS file DISKEDIT.SR

Course Title: **S209 AOS USER**

LAB Exercise Title: **DEDIT (DISK FILE EDITOR)**

PROGRAM DEVELOPMENT/DISK EDIT LAB

Bring a copy of the file :AOS.LABS:DISKEDIT.SR into your initial working directory. Record the new name of the copied file.

_____

Create a disk file containing the CLI commands to assemble and bind the source code file you have just copied. The /L= switch should be used with both utilities to gather any output into a single disk file. The appropriate switches should be employed to insure that a symbol table (.ST file) will be created. The last command in the CLI file should QPRINT. The file containing the output of the utilities. Record the names of the:

CLI COMMAND FILE _____

THE /L OUTPUT FILE _____

QSUBMIT the command file.

Hint:

XEQ MASM/L=disk file/U program name

XEQ BIND/L=disk file program name/U

Course Title: **S209 AOS USER**

LAB Exercise Title: **DEDIT (DISK FILE EDITOR)**

What are the advantages of having COMPILES, ASSEMBLIES and BIND's executed in a batch stream?

_____

_____

_____

_____

Once you have received your listing execute the program from your terminal. What is the displayed output?

_____

_____

Start the disk editor (DEDIT) using the .PR file of the program as input.

Display the location 462 in the following submodes.

F _____

H _____

N _____

Change the contents of the following locations to the indicated value.

Location	New Value
523	043517
524	047504
525	026502
526	054505

Course Title: **S209 AOS USER**

LAB Exercise Title: **DEDIT (DISK FILE EDITOR)**

What are the current display MODES that are in effect? _____

Change the format mode to A

What is the contents of these locations?

523 _____

524 _____

525 _____

526 _____

Indicate the symbol table for this program to the disk editor.

Set the display mode to F.

What is the contents of the field 'FLAG'?

Execute the following DEDIT command: XXX–7 =

Where XXX is the contents of the field FLAG (above).

Place the derived value into the location FLAG.

Course Title: **S209 AOS USER**

LAB Exercise Title: **DEDIT (DISK FILE EDITOR)**

In order to get the second message (GOOD ENDING) to appear the length of the message has to be stated within the program. What the above command does is shorted the length of the message by 7 characters.

Terminate the disk editor.

Once again execute the program from your terminal. What is the displayed output?

_____

_____

Course Title: **S209 AOS USER**

LAB Exercise Title: **SHARED ROUTINES**

Objectives:

To demonstrate the shared routine facility of AOS.

To demonstrate the proper usage of switches and arguments in the AOS utilities SLB (Shared Library Builder) and BIND (Binder)

Required Materials /Equipment:

AOS files	**SRA.OB**
	SRB.OB
	SRC.OB
	SRMAIN.OB
	UTILITIES.OB

N.B. The utilities.OB time delay and error handler routines.

Course Title: **S209 AOS USER**

LAB Exercise Title: **SHARED ROUTINES**

SHARED ROUTINE LAB

IT WOULD BE HELPFUL TO HAVE A COPY OF THE BINDER MANUAL HANDY.

* **MOVE COPIES OF SRA.OB, SRB.OB, SRC.OB, SRMAIN.OB AND UTILITIES.OB FROM :AOS.LABS TO YOUR WORKING DIRECTORY.**

* **WHILE IN YOUR WORKING DIRECTORY SET THE LISTFILE TO A NEW FILE (LISTFILE,HOLDIT)**

* **INDIVIDUALLY BIND EACH MODULE NAMED SRA.OB, SRB.OB AND SRC.OB USING GLOBAL SWITCHES /S AND /L ON THE BIND.**

* **TERMINATE THE LISTFILE AND PRINT IT. (LISTFILE/K;QPRINT HOLDIT)**

* **USING THE BIND LISTINGS FIND OUT HOW LONG IS "LENGTH OF SHARED?"**

  SRA? _____

  SRB? _____

  SRC? _____

* **IT TAKES 2000 (BASE 8) LOCATIONS TO EQUAL ONE PAGE. HOW MANY PAGES ARE REQUIRED FOR**

  SRA? _____

  SRB? _____

  SRC? _____

Course Title: **S209 AOS USER**

LAB Exercise Title: **SHARED ROUTINES**

* BUILD A SHARED LIBRARY OF .PR FILES SRA, SRB AND SRC.
  FOR EXAMPLE

  "XEQ,SLB/O=LIBNAME.SL,N,SRA,SRB,SRC"

  THE SYMBOL N REPRESENTS THE LIBRARY NUMBER. IT CAN EQUAL THE VALUES 2 THRU 63.

* BIND SRMAIN.OB AND UTILITIES.OB TOGETHER WITH YOUR SHARED LIBRARY. DON'T USE A /M SWITCH. BE SURE TO SPECIFY SRMAIN AS THE FIRST ARGUMENT.

* EXECUTE THE PROGRAM "SRMAIN".
  WHAT HAPPENED?

  _____

  _____

  _____

  IN THE BIND, NO SPACE WAS SET ASIDE TO BE USED BY SHARED ROUTINES. DO ANOTHER BIND, THIS TIME USE THE /M=1 SWITCH. ONE PAGE MUST BE SET ASIDE TO HOLD SHARED ROUTINES. ONE PAGE IS ALL THATS REQUIRED BECAUSE THE ROUTINES ARE ONE PAGE LONG AND ARE IMPLEMENTED WITH ?RCALL'S. EXECUTE THE NEW SRMAIN PROGRAM. WHAT HAPPENED?

  _____

  _____

  _____

Course Title: **S209 AOS USER**

LAB Exercise Title: **SHARED ROUTINES** | Page **3** of **3**

* THE PROGRAM FLOW IS AS FOLLOWS.

```
SRMAIN   ?RCALL'S   SRA.
SRA      ?RCALL'S   SRB.
SRB      ?RCALL'S   SRC.
SRC   RETURNS    CONTROL   TO   SRB.
SRB   RETURNS    CONTROL   TO   SRA.
SRA   RETURNS    CONTROL   TO   SRMAIN.
```

SINCE ROUTINES ARE BEING NESTED THE PROGRAM STACK WILL HAVE TO BE USED TO MAINTAIN THE STATE OF THE PARENT ROUTINE. BY DEFAULT A STACK OF 25 WORDS IS SET ASIDE. THIS IS NOT LARGE ENOUGH FOR THIS PROGRAM. USE THE /Z= SWITCH SET TO FIFTY (50) TO ALLOCATE ENOUGH SPACE ON THE STACK IN ADDITION TO THE GLOBAL /M SWITCH

* EXECUTE THE PROGRAM ONCE MORE. IF YOU DID EVERYTHING CORRECTLY IT SHOULD WORK.

* THE FOLLOWING IS A RECAP OF THE STEPS YOU SHOULD HAVE USED'

```
XEQ,BIND/S/L,SRA
XEQ,BIND/S/L,SRB
XEQ,BIND/S/L,SRC
XEQ,SLB/O=LIBNAME.SL,3,SRA,SRB,SRC
XEQ ,BIND/M=1/Z=50,SRMAIN,UTILITIES,LIBNAME.SL
```