Level 68

MULTICS CONCEPTS AND UTILIZATION

Student Handbook
Course Code F01

Honeywell Marketing Education

ISSUE DATE: June 27, 1977

REVISION: 4.1

REVISION DATE: October 1978

COURSE DESCRIPTION

F01 Multics Concepts and Utilization

Duration: Five Days

Intended For: Personnel requiring capability to use the Multics system.

Course Synopsis: This course presents the basic information needed to use the Multics system, including discussions and examples of: the typing conventions used for Multics terminals; a Multics text editor; commands used to write, compile and execute programs; utility command which provide of the user environment, manipulation of files, and inter-user communication; and the types of errors which can occur, with procedures for recovery. A general overview of Multics hardware and software facilities is also presented.

Interactive workshops are included to reinforce the material presented.

Objectives: Upon completion of this course, the student should be able to:

1. Login to and logout from the Multics system.

2. Create and edit files in the storage system with the qedx text editor.

3. Compile, run, and debug simple programs, and manipulate the run-time environment.

4. Use other system commands to manipulate files, tailor the user environment, and communicate with other users.

Prerequisite: Programming Logic and Flowcharting (G024) or previous data processing experience.

F01 TOPIC MAP

DAY	MORNING TOPICS	AFTERNOON TOPICS
1	Welcome/Administration Multics Overview How to Access Multics Workshop #1	qedx Basics Workshop #2 More qedx
2	The Storage System Storage System Commands Workshop #3	The Command Language Exec_com Basics The Abbrev Processor Workshop #4
3	Programming on Multics Workshop #5	Access Control User Communication Input/Output Facilities Workshop #6
4	More Abbrev Processor Active Functions More About Exec_com's Workshop #7	Absentee Usage Software Conventions Workshop #8
5	Additional Commands Workshop #9	Software Overview Hardware Overview Questions

CONTENTS

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

This page has intentionally
been left blank.

STUDENT BACKGROUND

Multics Concepts and Utilization (F01)

NAME: ______________________ PHONE: __________

TITLE: ______________________________

COMPANY ADDRESS: ______________________

____

____

MANAGER: ______________________ OFFICE PHONE: __________

INSTRUCTOR'S NAME: DANNY CHAPIN

1. Do you meet the prerequisite as stated in the "Course Description" of the student text? If yes, check "a" or "b". If no, check "c" or "d".

   a [ ] Prerequisite satisfied by attending course indicated in "Course Description".

   b [ ] Meet prerequisite by equivalent experience (explain briefly)

   ____

   c [ ] Elected or instructed to attend course anyway.

   d [ ] Was not aware of prerequisite.

2. What related Honeywell courses have you attended? Furnish dates and instructors if possible.

   ____

   ____

   ____

   ____

(PLEASE TURN OVER)

STUDENT BACKGROUND

3. Check the boxes for which you have <u>any</u> related experience. (May be other than Honeywell's)

   [ ] PL1 [ ] COBOL [ ] FORTRAN [ ] ASSEMBLY

   [ ] JCL [ ] OPERATIONS [ ] GCOS [ ] MULTICS

   [ ] OTHER COMPUTER RELATED

4. Detail any Multics experience you have had:

5. Objectives for attending this course (May check more than one).

   [ ] Require information to provide support for a Multics system

   [ ] To maintain an awareness of this product

   [ ] To evaluate or compare its potentials

   [ ] Required to use or implement

   [ ] Need update from a previous release

   [ ] Require a refresher

   [ ] Other:

TOPIC I

MULTICS OVERVIEW

This page has intentionally
been left blank.

SOME BASIC TERMINOLOGY

- HARDWARE

  - REFERS TO THE PHYSICAL COMPONENTS OF A COMPUTER; ESPECIALLY TO THE ELECTRONIC CIRCUITRY

  - MAJOR COMPONENTS: CENTRAL PROCESSOR UNIT (CPU), MAIN MEMORY, DISK DRIVES, TAPE UNITS, PRINTER, CARD READER, CARD PUNCH

- SOFTWARE

  - REFERS TO THE PROGRAMS, PROCEDURE, CODE, OR INSTRUCTIONS THAT EXECUTE ON THE HARDWARE

  - EXAMPLES: A FORTRAN PROGRAM, THE PL/I COMPILER, THE DATA BASE MANAGER, THE OPERATING SYSTEM

SOME BASIC TERMINOLOGY

- SYSTEM RESOURCES

  - CENTRAL PROCESSOR / CPU (TIME OCCUPIED)

  - MAIN MEMORY / PRIMARY MEMORY / CORE (AMOUNT OCCUPIED * TIME OCCUPIED)

  - PERIPHERAL DEVICES

    - DISK DRIVES (AMOUNT OF DATA TRANSFERRED)

    - TAPE DRIVES (AMOUNT OF DATA TRANSFERRED)

    - CARD PUNCH (AMOUNT OF DATA PUNCHED)

    - PRINTER (AMOUNT OF DATA PRINTED)

    - TERMINALS (CONNECT TIME & AMOUNT OF DATA TRANSFERRED)

  - MEDIA

    - DISK PACKS / SECONDARY MEMORY (AMOUNT OF DATA STORED)

    - TAPES (NUMBER USED)

    - CARDS (NUMBER PUNCHED)

    - PRINTER PAPER (AMOUNT USED)

  - SYSTEM AND SITE PROVIDED SOFTWARE (RENTAL)

SOME BASIC TERMINOLOGY

- BIT
  - THE MOST FUNDAMENTAL UNIT OF INFORMATION
  - A BIT IS EITHER ON OR OFF (1 OR 0)
- BYTE
  - A SMALL UNIT FOR MEASURING THE AMOUNT OF MEMORY, THE SIZE OF A PROGRAM OR FILE, OR THE SPACE ON A TAPE OR DISK PACK
  - ON MOST LARGE SCALE HONEYWELL EQUIPMENT, A BYTE EQUALS 9 BITS
  - ON MULTICS, ONE ASCII CHARACTER OCCUPIES ONE BYTE
- WORD
  - A LARGER UNIT FOR MEASURING THE AMOUNT OF MEMORY, THE SIZE OF A PROGRAM OR FILE, OR THE SPACE ON A TAPE OR DISK PACK
  - ON MOST LARGE SCALE HONEYWELL EQUIPMENT, A WORD EQUALS 4 BYTES OR 36 BITS

NUMBERS/NON ASCII

SOME BASIC TERMINOLOGY

- USER

  - ONE WHO USES THE COMPUTER FACILITY

- PROJECT

  - A SET OF USERS GROUPED TOGETHER FOR ACCOUNTING AND ACCESS PURPOSES

  - A PROJECT IS OFTEN A GROUP OF USERS WORKING TOWARDS A COMMON GOAL

  - USERS ON A PROJECT OFTEN WORK FOR THE SAME DEPARTMENT OR UNIT

DEVELOPMENT HISTORY

- MULTICS

  - MULTIPLEXED INFORMATION AND COMPUTING SERVICE

- 1960-1963: CTSS DEVELOPMENT

  - FIRST LARGE-SCALE TIME-SHARING SYSTEM

  - EXPERIENCED GAINED WAS USED LATER IN THE MULTICS PROJECT

  - CTSS WAS USED TO DEVELOP MULTICS SOFTWARE

- 1964: INITIAL SPECIFICATIONS FOR MULTICS

DEVELOPMENT HISTORY

- 1964-1965: INITIAL MULTICS DEVELOPMENT WAS A JOINT EFFORT BY:
    - MASSACHUSETTS INSTITUTE OF TECHNOLOGY (PROJECT MAC)
        - OVERALL PROJECT CO-ORDINATION
        - OPERATING SYSTEM
        - SELECTION AND DESIGN MODIFICATIONS OF THE HARDWARE (GE 635)
    - BELL TELEPHONE LABORATORIES
        - COMPILERS AND ASSEMBLER
        - FILE SYSTEM
    - GENERAL ELECTRIC COMPUTER DIVISION (HONEYWELL)
        - HARDWARE MODIFICATION

- 1965: FALL JOINT COMPUTER CONFERENCE
    - PRIMARY TECHNICAL PAPERS PRESENTED

DEVELOPMENT HISTORY

- 1967: GE 645 HARDWARE AVAILABLE (MODIFIED GE 635)

- 1967: SOFTWARE DEVELOPMENT UNDERWAY

- 1968: AVAILABLE TO SYSTEM PROGRAMMERS

- 1969: AVAILABLE FOR USE AT MIT

- 1971: PROJECT MAC FUNDING STOPPED. HIS ADOPTS.

- 1972: 6180 HARDWARE AVAILABLE

  - ▯ BULK STORE REPLACED DRUM

  - ▯ EXTENDED INSTRUCTION SET (EIS)

  - ▯ RING MECHANISM MOVED TO HARDWARE

DEVELOPMENT HISTORY

- 1973: ANNOUNCED AS A "STANDARD" HIS PRODUCT

- 1974: LEVEL 68 HARDWARE

  - 68/60 MOS MEMORY

  - 68/80 MOS MEMORY AND CACHE STORE

- 1975: COBOL-74

- 1975: SORT/MERGE

- 1976: MULTICS DATA BASE MANAGER (MDBM)

- 1977: WORD PROCESSING SYSTEM (WORDPRO)

MULTICS DESIGN GOALS AND RESULTS

- REMOTE TERMINAL ACCESS AS NORMAL USAGE MODE
  - ALL SYSTEM RESOURCES AVAILABLE VIA TERMINALS
  - USAGE OF CARDS IS MINIMAL
  - INTERACTIVE AND BATCH ENVIRONMENTS ARE COMPATIBLE
  - INTER-USER COMMUNICATION

- CONTINUOUS/UTILITY GRADE OPERATION
  - EXTREMELY STABLE OPERATING SYSTEM
    - MODULAR DESIGN
    - WRITTEN IN PL/I
  - ON LINE METERING, ACCOUNTING, BILLING AND SOFTWARE INSTALLATION
  - OPERATORS REQUIRED: ONE
  - UNATTENDED OPERATION MODE

MULTICS DESIGN GOALS AND RESULTS

- ABILITY TO GROW AND CONTRACT WITHOUT SYSTEM OR USER REORGANIZATION

  - SYSTEM SIZE - TRANSPARENT TO USERS, PROGRAMS, AND OPERATING SYSTEM

  - CHANGES TO SYSTEM SIZE ARE MADE WITHOUT REGENERATING THE OPERATING SYSTEM OR USER PROGRAMS

  - DYNAMIC RECONFIGURATION - TRANSPARENT TO USERS AND PROGRAMS

- DECENTRALIZATION OF THE SYSTEM'S ADMINISTRATION

  - HIERARCHY OF ADMINISTRATORS:

    - THE SYSTEM ADMINISTRATOR

    - PROJECT ADMINISTRATORS

    - USERS

MULTICS DESIGN GOALS AND RESULTS

- DECENTRALIZATION OF THE SYSTEM'S LOGICAL FILE SPACE

  - HIERARCHY OF DIRECTORIES:

    - ROOT DIRECTORY

    - SYSTEM DIRECTORIES

    - PROJECT DIRECTORIES

    - USER DIRECTORIES

- RELIABLE FILE SYSTEM

  - INCREMENTAL BACKUP SYSTEM (Backup SysDaemon)

  - AUTOMATED RETRIEVAL

  - SALVAGE SUBSYSTEM

MULTICS DESIGN GOALS AND RESULTS

- ACCESS CONTROLS THAT ALLOW SELECTIVE SHARING OF INFORMATION AND SERVICES

  - MULTICS IS THE MOST SECURE COMMERCIAL OPERATING SYSTEM AVAILABLE

  - USER AUTHENTICATION (BY PASSWORD)

  - ACCESS CONTROL LIST - ACL (BY NAME AND PROJECT)

  - ACCESS ISOLATION MECHANISM - AIM (BY SECURITY LEVEL AND NEED TO KNOW)

  - RING STRUCTURE (8 LEVEL MASTER/SLAVE HIERARCHY)

- SERVES BOTH LARGE AND SMALL USERS EFFICIENTLY

  - RESOURCES ARE AUTOMATICALLY ALLOCATED ON DEMAND - WHEN NEEDED, AND IN PROPORTION TO THE SIZE OF THE TASK

MULTICS DESIGN GOALS AND RESULTS

- COMBINE SYSTEM FLEXIBILITY WITH EASE OF USE
  - VIRTUAL MEMORY
  - ONLY ONE STORAGE SYSTEM FOR BOTH USERS AND OPERATING SYSTEM
  - ASCII CHARACTER SET USED THROUGHOUT
  - INTERACTIVE AND BATCH ENVIRONMENTS ARE COMPATIBLE
  - EXEMPLARY COMMAND LANGUAGE (NO JCL!)
    - COMMAND NAMES
      - FULL NAME (DESCRIPTIVE)
      - SHORT NAME (CONVENIENT)
    - CONTROL ARGUMENTS WITH INTELLIGENT DEFAULTS
    - ACTIVE FUNCTIONS
    - STAR AND EQUAL CONVENTION
    - COMMAND ITERATION
  - ABBREV PROCESSOR
  - EXEC_COMS (COMMAND FILES)

MULTICS DESIGN GOALS AND RESULTS

- DIFFERENT ENVIRONMENTS AND HUMAN INTERFACES WITHIN A SINGLE SYSTEM
  - STANDARD SERVICE (FULL MULTICS)
  - SUBSYSTEM (E.G. FAST, BASIC, APL)
  - LIMITED SERVICE
  - TAILORED ENVIRONMENT

- EVOLUTIONARY DESIGN ABLE TO INCORPORATE TECHNOLOGICAL IMPROVEMENTS AND TO MEET GROWING USER EXPECTATIONS
  - MODULAR DESIGN
  - GENERAL (NOT SPECIFIC) SOLUTIONS
  - COMPATIBLE EXPANSIONS

ADMINISTRATION

- THE EFFECTIVE ADMINISTRATION OF A LARGE COMPUTER INSTALLATION CAN BE A DIFFICULT JOB

- FOR MULTICS, A HIERARCHY OF ADMINISTRATORS HAS BEEN DEFINED IN ORDER TO:

  - DECENTRALIZE CONTROL OF SYSTEM RESOURCES BY DISTRIBUTING AUTHORITY AND RESPONSIBILITY TO LOWER LEVELS

- THE MULTICS ADMINISTRATION HIERARCHY CONSIST OF THREE LEVELS:

  - SYSTEM ADMINISTRATOR(1)

    - PHYSICAL, ADMINISTRATIVE AND FINANCIAL CONCERNS OF THE SYSTEM

  - PROJECT ADMINISTRATOR(2)

    - ADMINISTRATIVE AND FINANCIAL CONCERNS OF THE PROJECT

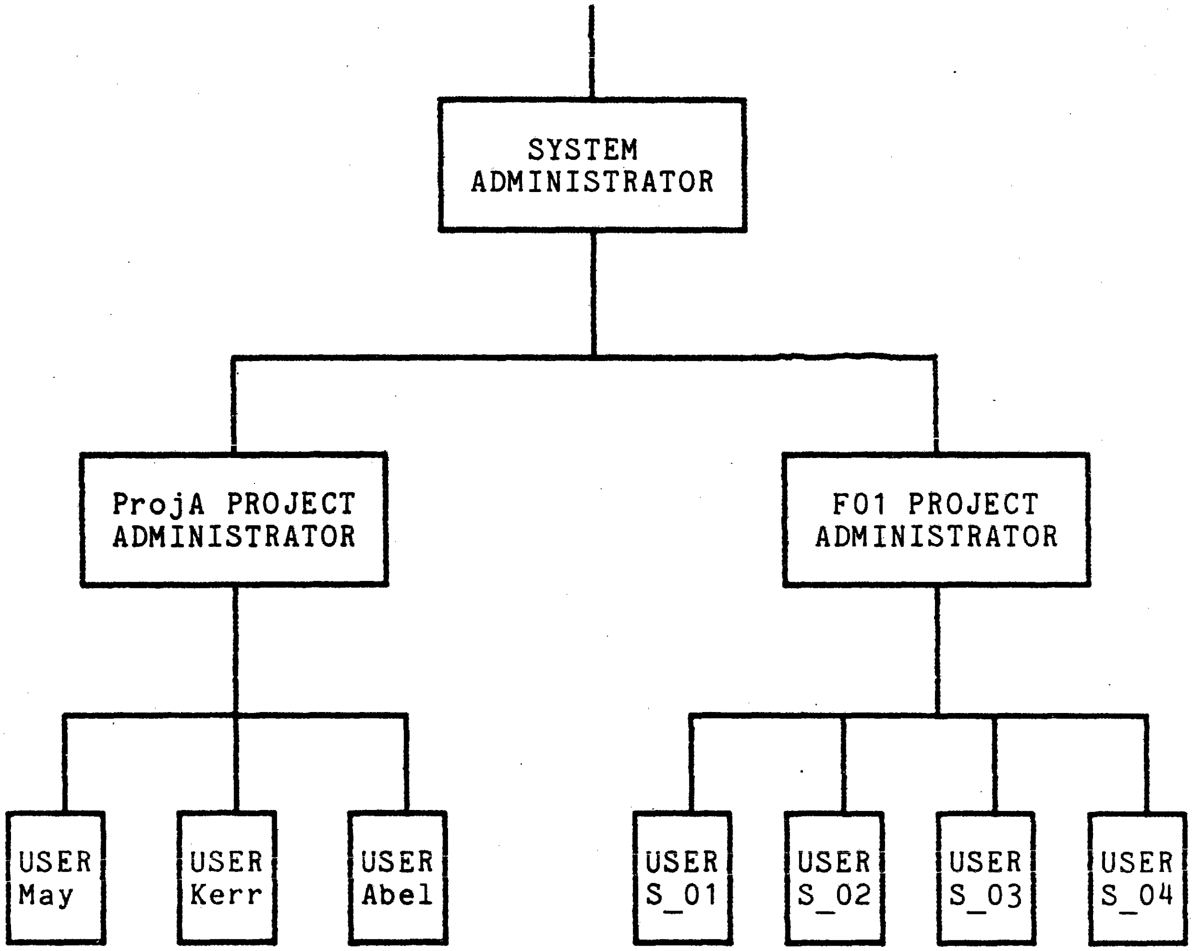
  - USER

    - CONTROL AND USAGE OF RESOURCES ALLOCATED TO HIM/HER

[1] Related Multics Course: System Administration (F60)

[2] Related Multics Course: Project Administration (F61)

ADMINISTRATION



ADMINISTRATION

- THE SYSTEM ADMINISTRATOR

  - ESTABLISHES SYSTEM CONFIGURATION AND OPERATING PARAMETERS (METERING AND TUNING)

  - DEFINES, CREATES, AND ALLOCATES RESOURCES TO THE SYSTEM PROJECTS

  - ADMINISTERS THE SYSTEM'S SECURITY NEEDS

  - REGISTERS USERS AND ASSIGNS INITIAL (AND NEW) PASSWORDS

  - MAINTAINS RECORDS OF SYSTEM USAGE

  - SETS RESOURCE USAGE PRICES AND DETERMINES BILLING CYCLE

  - GENERATES STATEMENT OF CHARGES TO RECOVER THE COST OF SYSTEM RESOURCES USED

  - PERFORMS ALL OF THE ABOVE TASKS FROM A TERMINAL AND WITHOUT INTERRUPTION OF SERVICE

ADMINISTRATION

- THE PROJECT ADMINISTRATOR
  - DISTRIBUTES RESOURCES AND ATTRIBUTES ALLOCATED TO PROJECT
  - HAS ACCESS TO THE PROJECT'S RESOURCE USAGE AND ACCOUNTING DATA
  - ADDS AND DELETES REGISTERED (AND ANONYMOUS) USERS TO PROJECT
  - DEFINES THE ENVIRONMENT AND SETS RESOURCE LIMITS FOR USERS
  - SETS ACCESS ON USERS' HOME DIRECTORY

- THE USER
  - HAS CONTROL OVER RESOURCES ALLOCATED TO HIM
  - HAS ACCESS TO HIS OWN RESOURCE USAGE AND ACCOUNTING DATA
  - MAY SELECTIVELY SHARE HIS PROGRAMS AND DATA WITH OTHER USERS
  - MAY CHANGE HIS PASSWORD

TOPIC II

HOW TO ACCESS MULTICS

This page has intentionally
been left blank.

REMOTE TERMINAL ACCESS

- REMOTE TERMINAL ACCESS IS THE NORMAL MODE OF ACCESS

- ALL SYSTEM RESOURCES ARE ACCESSIBLE VIA REMOTE TERMINAL

- USES STANDARD TELEPHONE LINES

- "LOGGING IN" REFERS TO THE PROCESS OF:

  - TELEPHONING THE MULTICS SYSTEM

  - CONNECTING THE TELEPHONE TO THE TERMINAL

  - WAITING FOR MULTICS TO SEND YOU A GREETING

  - IDENTIFYING YOURSELF TO THE SYSTEM

USER REGISTRATION AND IDENTIFICATION

- NEW USERS ARE REGISTERED BY THE SITE SYSTEM ADMINISTRATOR

- ADMINISTRATOR ASSIGNS A Project_id, A UNIQUE Person_id, AND A PASSWORD

Person id's	Project id's	PASSWORD'S
TSmith	ProjA	ts
Greenberg	ProjA	$$$!
Student_04	F01	atlanta

- A USER MAY BELONG TO MORE THAN ONE PROJECT - HOWEVER, ONE PROJECT IS ALWAYS DESIGNATED AS THE USER'S "DEFAULT PROJECT"

- THE USER'S PASSWORD IS ASSOCIATED WITH HIS Person_id ONLY

- User_id

  ▯ REFERS TO THE Person_id.Project_id PAIR

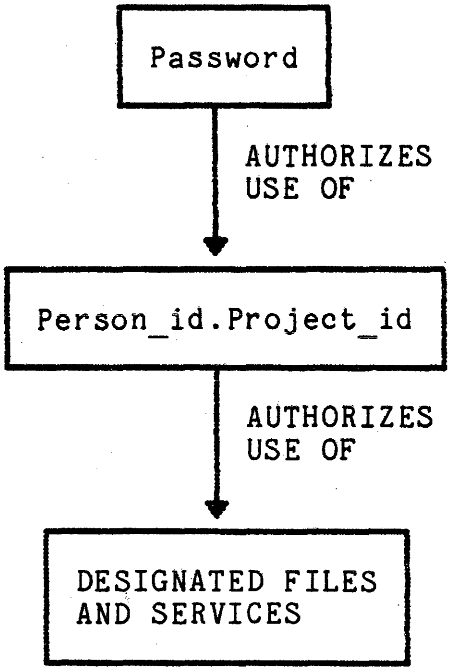
  TSmith.ProjA

  SWebber.Doc_66

USER REGISTRATION AND IDENTIFICATION

- THE USER'S PASSWORD IS REQUIRED IN ORDER TO AUTHENTICATE THE USE OF THE USER'S Person_id

- THE USER'S Person_id AND Project_id ARE THE KEYS THAT DETERMINE WHAT INFORMATION AND SERVICES HE IS AUTHORIZED TO ACCESS AND CONTROL



CHARACTERISTICS OF TERMINALS

- TERMINALS BEHAVE LIKE ELECTRIC TYPEWRITERS

- MANY HAVE SEPARATE "CARRIAGE RETURN" AND "LINE FEED" KEYS

- CONCEPTUALLY, THE "LINE FEED" KEY SENDS THE TYPED LINE

- TERMINAL SWITCHES:

DUPLEX:	HALF OR FULL	MODE:	LINE
CASE:	LOWER	RATE:	30
PARITY:	EVEN		

- TERMINAL TO COMPUTER COUPLINGS:
  - HARDWIRED (A PERMANENT WIRE CONNECTS TERMINAL TO COMPUTER)
  - TELEPHONE-MODEM (TELEPHONE LINE CONNECTS TERMINAL TO COMPUTER)
    - ACOUSTIC (PROXIMITY)
    - DIRECT (PUSH A BUTTON)

ACCESS SEQUENCE

- PLUG-IN AND TURN ON THE TERMINAL

- SET THE TERMINAL SWITCHES

- IF HARDWIRED TERMINAL

    ▯ IDENTIFY YOURSELF VIA THE login COMMAND

- IF TELEPHONE-MODEM TERMINAL

    ▯ DIAL THE APPROPRIATE NUMBER AND WAIT FOR HIGH-PITCHED TONE

    ▯ CONNECT THE PHONE TO THE COUPLER DEVICE

    ▯ WAIT FOR MULTICS TO RESPOND WITH A MESSAGE SIMILAR TO:

    ```
    Multics MR6.0: Honeywell LISD Phoenix, System M
    Load = 51.0 out of 95.0 units: users = 51
    ```

    ▯ IDENTIFY YOURSELF VIA THE login COMMAND

LOGIN AND LOGOUT COMMANDS

- login, l

  - A COMMAND USED TO GAIN ACCESS TO MULTICS

  - INITIATES A PROGRAM CALLED THE USER'S PROCESS

  - USAGE:

    ```
    login Person_id {Project_id}

    l TSmith
    l TSmith FED
    l Student_07 F01
    ```

    IF A Project_id IS NOT SPECIFIED THE USER'S DEFAULT Project_id IS ASSUMED

  - RESULTS: THE USER WILL BE ASKED TO SUPPLY THE PASSWORD ASSOCIATED WITH HIS Person_id

    - A PASSWORD MASK WILL BE GENERATED -OR- PRINTING OF THE PASSWORD WILL BE INHIBITED

    - SUPPLYING THE CORRECT PASSWORD COMPLETES THE USER'S LOG IN SEQUENCE

  - THE Project_id USED AT LOG IN DETERMINES WHO RECEIVES THE BILL FOR THE CURRENT TERMINAL SESSION

  - THE Person_id AND Project_id USED AT LOG IN DETERMINES WHERE IN THE MULTICS VIRTUAL MEMORY THE USER "FINDS" HIMSELF

LOGIN AND LOGOUT COMMANDS

```
   <DIAL TELEPHONE NUMBER>
   <CONNECT TERMINAL/TELEPHONE>
   <WAIT FOR LOGIN HERALD>


   Multics MR6.0: Honeywell LISD Phoenix, System M
   Load = 51.0 out of 95.0 units: users = 51

!  1 TSmith
   Password
!  ████████████
   TSmith ProjA logged in 06/28/77  1553.2 mst Tue from terminal "243".
   Last login 06/28/77  1425.8 mst Tue from terminal "013"
   A new PL/1 compiler was installed; type help new_pl1.
   Type help sked for hours of operation FW31.
   r 1553.5 1.314 1.332 30
```

typed only in ! not on computer

- READY MESSAGE

  - A MESSAGE THAT IS PRINTED EACH TIME THE USER IS AT "COMMAND LEVEL"

  - THE READY MESSAGE REPORTS

    - THE TIME OF DAY

    - THE NUMBER OF CPU SECONDS USED SINCE THE LAST READY MESSAGE

    - THE NUMBER OF MEMORY UNITS USED SINCE THE LAST READY MESSAGE

    - THE NUMBER OF PAGES (1024 WORDS) BROUGHT INTO MEMORY FOR THE USER SINCE THE LAST READY MESSAGE

LOGIN AND LOGOUT COMMANDS

- logout

  - INFORMS MULTICS THAT THE USER IS THROUGH WITH THE CURRENT TERMINAL SESSION

  - USAGE: logout

  - RESULTS: THE USER WILL BE DISCONNECTED FROM MULTICS

```
!       logout
        TSmith ProjA logged out 06/28/77 1749.4 mst Tue
        CPU usage 17 sec, memory usage 103.1 units
        hangup
```

- new_proc

  - DESTROYS THE USER'S CURRENT PROCESS AND CREATES A NEW ONE

  - EFFECTIVELY THE SAME AS LOGGING OUT AND LOGGING IN AGAIN

  - OFTEN USED "WHEN ALL ELSE FAILS" OR TO RESET THE USER'S ENVIRONMENT

  - USAGE: new_proc

TYPING CONVENTIONS

● # (NUMBER SIGN)

▯ USED TO "ERASE" THE PREVIOUS CHARACTER

```
login TSM#mith

login TSMith####mith

logen##in TSme#ith

logim   T###n TSmith
```

▯ NOTE: WHITE SPACE IS COUNTED AS ONE CHARACTER WHEN USING THE ERASE CHARACTER

● @ (AT SIGN)

▯ USED TO "KILL" THE CURRENT LINE

```
login TSMith@login TSmith

logen@login TSme#ith

logen##im#n Tsm##Sme@login TSmith
```

TYPING CONVENTIONS

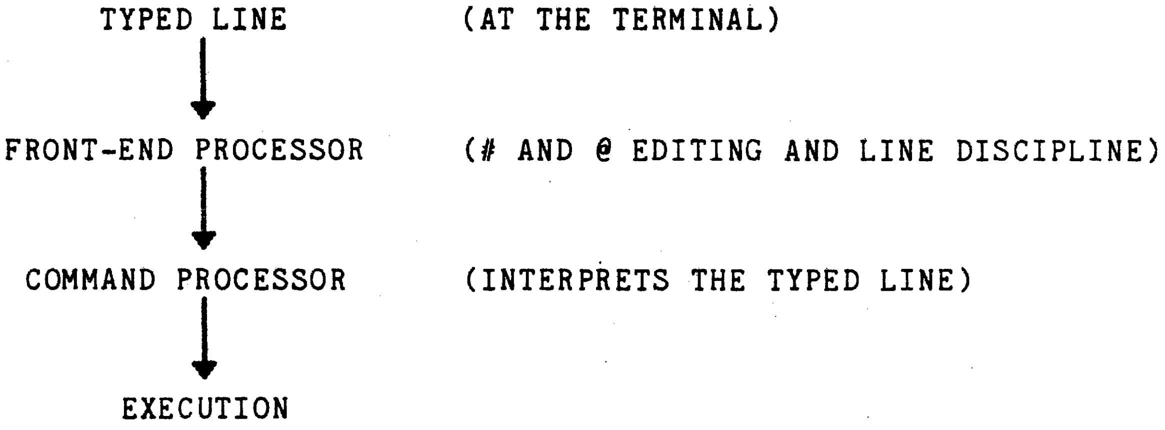
- \ (BACKSLASH)

  - THE CHARACTER \ IS A FRONT-END PROCESSOR ESCAPE SEQUENCE THAT CAUSES CERTAIN CHARACTER THAT FOLLOWS TO BE INTERPRETED AS A LITERAL

  - OFTEN USED TO SUPPRESS (ESCAPE) THE SPECIAL MEANING OF #, @, linefeed AND OTHER SPECIAL CHARACTERS

```
sm TSmith.ProjA I need a \#8 tin can.

sm TSmith.ProjA He's selling 3 \@ $4.50 each.
```

- COMMAND LINE FLOW



TYPED LINE (AT THE TERMINAL)

FRONT-END PROCESSOR (# AND @ EDITING AND LINE DISCIPLINE)

COMMAND PROCESSOR (INTERPRETS THE TYPED LINE)

EXECUTION

SOME SIMPLE COMMANDS

- who

    ▯ LISTS THE NAMES AND PROJECTS OF ALL USERS CURRENTLY LOGGED IN

    ▯ USAGE:

    ```
    who {args} {-control_args}
    who
    who -name
    who -long
    who TSmith
    who .ProjA
    who May .F01 .ProjA
    ```

* beside name means absentee program running

– will list specific options

- how_many_users, hmu

    ▯ TELLS HOW MANY USERS ARE CURRENTLY LOGGED IN

    ▯ USAGE:

    ```
    how_many_users {args} {-control_args}
    hmu
    hmu -long
    hmu TSmith
    hmu .ProjA
    hmu May .F01 .ProjA
    ```

SOME SIMPLE COMMANDS

- help (YA WANT IT, WE GOT IT)

  - PROVIDES INFORMATION ABOUT THE MULTICS SYSTEM AND ITS COMMANDS

  - USAGE:

    ```
    help {name}
    help who
    help sked
    help help
    ```

  - ANSWER: yes, no, rest, skip, section str, search sA sB ...

- list, ls (ROLL CALL)

  - RETURNS INFORMATION ABOUT THE USERS SEGMENTS (FILES)

  - USAGE:

    ```
    list
    ls
    ```

SOME SIMPLE COMMANDS

- accept_messages, am (I'M LISTENING)

  - ENABLES THE USER TO RECEIVE MESSAGES AT HIS TERMINAL

  - OTHERWISE, MESSAGES WILL GO TO THE USER'S MAILBOX (A SEGMENT HAVING THE NAME Person_id.mbx)

  - CREATES A PERMANENT MAILBOX FOR THE USER IF NONE EXISTS

  - USAGE:

    ```
    USAGE:  accept_messages

            am
    ```

- send_message, sm

  - SENDS MESSAGES TO A GIVEN USER ON A GIVEN PROJECT

  - MESSAGES ARE EITHER:

    - PRINTED ON THE RECIPIENT'S TERMINAL, OR

    - PLACED IN THE RECIPIENT'S MAILBOX

  - USAGE:

    ```
    USAGE:    send_message Person_id.Project_id message

              sm TSmith.ProjA When are you going to lunch?

              sm Greenberg.FED May I have access to your file?
    ```

SOME SIMPLE COMMANDS

- print_messages, pm

  - PRINTS ALL MESSAGES STORED IN THE USER'S MAILBOX

  - MESSAGES ARE DELETED FROM THE MAILBOX WHEN PRINTED

  - USAGE:
    ```
    print_messages
    pm
    ```

- defer_messages, dm (I'M BUSY...NO DISTRACTIONS WANTED)

  - REDIRECTS ANY AND ALL INCOMING MESSAGES TO THE USER'S MAILBOX

  - ELIMINATES UNWANTED INTERRUPTIONS

  - THIS IS THE DEFAULT UPON LOGIN

  - "UNDONE" BY THE accept_message COMMAND

  - USAGE:
    ```
    defer_messages
    dm
    ```

YOU ARE NOW READY FOR WORKSHOP #1

This page has intentionally
been left blank.

TOPIC III

QEDX BASICS

This page has intentionally
been left blank.

WHAT IS QEDX

- qedx, qx

  - ONE OF SEVERAL TEXT EDITORS AVAILABLE ON MULTICS USED TO

    - MODIFY THE CONTENTS OF EXISTING ASCII SEGMENTS

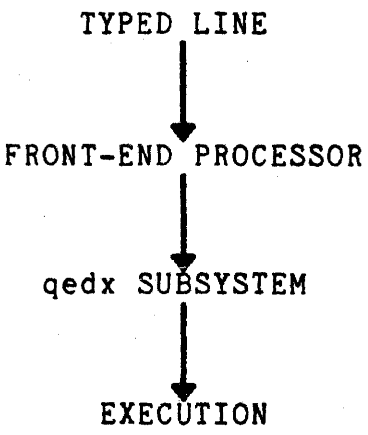
    - CREATE (INPUT) THE CONTENTS OF DESIRED ASCII SEGMENTS

  - A SUBSYSTEM WHICH CAN ONLY BE ENTERED BY COMMAND, AND EXITED BY REQUEST

  - USAGE: qedx

    qx

COMMAND LINE FLOW



TYPED LINE

FRONT-END PROCESSOR

qedx SUBSYSTEM

EXECUTION

QEDX CONCEPTS

- USER INVOKES THE EDITOR BY TYPING qedx OR qx

- THERE ARE TWO PRINCIPAL MODES OF OPERATION WITHIN qedx

  - EDIT MODE

    - THE INITIAL (DEFAULT) MODE WHEN ENTERING qedx

    - THE USER READS THE CONTENTS OF AN EXISTING SEGMENT INTO A BUFFER (A SCRATCH PAD)

    - THE USER THEN PERFORMS EDITING FUNCTIONS ON THE CONTENTS OF THE BUFFER BY TYPING EDIT REQUESTS:

      - LOCATING

      - SUBSTITUTING

      - DELETING

      - PRINTING

    - THE USER THEN WRITES (SAVES) THE EDITED VERSION OF THE SEGMENT BACK TO THE SAME (OR A NEW) SEGMENT

  - INPUT MODE

    - THE USER ENTERS THE INPUT MODE (FROM THE EDIT MODE) BY TYPING ONE OF THREE INPUT REQUESTS

    - ALL SUBSEQUENT TEXT FROM THE TERMINAL (EXCEPT ESCAPE SEQUENCES) IS APPENDED TO THE USER'S BUFFER (A SCRATCH PAD)

    - "\f" IS AN ESCAPE SEQUENCE THAT RETURNS THE USER TO THE EDIT MODE

QEDX CONCEPTS

- ALL LINES IN A qedx BUFFER ARE GIVEN IMAGINARY LINE NUMBERS STARTING WITH 1 (ONE)

- THERE EXISTS A CONCEPTUAL POINTER TO INDICATE THE "CURRENT LINE"

- qedx REQUESTS MAY DO ONE OR MORE OF THE FOLLOWING

  - MOVE THE CONCEPTUAL POINTER

  - PERFORM OPERATIONS ON THE CURRENT LINE

  - PERFORMS OPERATIONS ON A SET OF LINES WITH 1 (ONE)

- ALL LINES CAN BE ADDRESSED BY SUPPLYING THEIR LINE NUMBER. NO LINE NUMBER IMPLIES "CURRENT LINE"

p	(PRINT CURRENT LINE)
5p	(PRINT THE FIFTH LINE)
3,9d	(DELETE LINES 3 THROUGH 9)

QEDX CONCEPTS

- THE ADDITION AND DELETION OF LINES AFFECT THE IMAGINARY LINE NUMBERS IMMEDIATELY

- AFTER EACH qedx REQUEST, THE "CURRENT LINE" GENERALLY BECOMES THE LAST LINE ADDRESSED

- FOR CONVENIENCE, THE LAST LINE CAN ALSO BE ADDRESSED BY USING THE SYMBOL $ (DOLLAR SIGN)

BASIC QEDX REQUESTS

EDIT REQUESTS	DESCRIPTION
r path	READ: READ CONTENTS OF THE SEGMENT SPECIFIED BY path AND APPEND AFTER THE SPECIFIED LINE ($ ASSUMED)
p	PRINT: PRINT THE SPECIFIED LINE(S) ON THE TERMINAL
=	LINE NUMBER: PRINT LINE NUMBER OF SPECIFIED LINE
d	DELETE: DELETE SPECIFIED LINE(S) FROM THE BUFFER
/xxx/	LOCATE: LOCATE AND PRINT THE NEXT LINE CONTAINING THE xxx CHARACTER STRING. WRAP AROUND IF NECESSARY
s/existing/new/	SUBSTITUTE: SUBSTITUTE EVERY OCCURRENCE OF THE EXISTING STRING WITH THE NEW CHARACTER STRING IN SPECIFIED LINE(S)
w path	WRITE: WRITE THE SPECIFIED LINE(S) OF THE BUFFER INTO SEGMENT HAVING THE NAME path (ENTIRE BUFFER IS THE DEFAULT)
e command_line	EXECUTE: PASS THE REMAINDER OF THE REQUEST LINE TO THE MULTICS COMMAND PROCESSOR FOR EXECUTION
q	QUIT: EXIT FROM THE EDITOR

BASIC QEDX REQUESTS

<u>INPUT REQUESTS</u> <u>DESCRIPTION</u>

a APPEND: ENTER INPUT MODE AND APPEND THE LINE(S) TYPED AT THE TERMINAL <u>AFTER</u> THE SPECIFIED LINE

i INSERT: ENTER INPUT MODE AND INSERT THE LINE(S) TYPED AT THE TERMINAL <u>BEFORE</u> THE SPECIFIED LINE

c CHANGE: ENTER INPUT MODE AND <u>REPLACE</u> THE SPECIFIED LINE(S) WITH THE LINE(S) TYPED AT THE TERMINAL

\f terminates these modes. (not part of text

BASIC QEDX REQUESTS

. ⇒ current line

- ADDRESSING SYNTAX ⇒ can use these in any combinations

  - qedx REQUEST MAY TAKE ONE OF THREE GENERAL FORMS:

    - <request> - GENERALLY APPLIED TO THE CURRENT LINE

```
r temp.pl1          (APPENDS TO END OF BUFFER)

p

s/old/new/

d

w add.pl1           (WRITES THE ENTIRE BUFFER)
```

    - ADR<request> - APPLIED TO THE LINE ADDRESSED

```
6r >udd>FED>Kerr>temp.pl1

5p

7s/old/new/

9d

3w add.pl1
```

    - ADR,ADR<request> - APPLIED TO THE RANGE OF LINES ADDRESSED

including numbers shown

```
1,$p

5,15s/old/new/

9,12d

1,20w add.pl1
```

// . ± ADD <REQUEST>   RELATIVE

// /XXX/ <REQUEST>   CONTEXT

BASIC QEDX EXAMPLES

```
!         qedx
!         a
!         "Now is hte time
!         for al good
!         their county
!         \f
```

get out of input mode

RESULTING BUFFER

```
   ┌──────────────────────┐
   │ "Now is hte time     │ 1
   │ for al good          │ 2
 → │ their county         │ 3
   └──────────────────────┘
```

```
!         1,$p
          "Now is hte time
          for al good
          their county
!         2p
          for al good
!         s/al/all/
!         $s/ty/try."/
!         p
          their country."
!         w Henry.quote
```

RESULTING BUFFER

```
   ┌──────────────────────┐
   │ "Now is hte time     │ 1
   │ for all good         │ 2
 → │ their country."      │ 3
   └──────────────────────┘
```

```
!         /good/
          for all good
!         a
!         men to come
!         to hte aid of
!         1,$p
!         \f
```

RESULTING BUFFER

```
   ┌──────────────────────┐
   │ "Now is hte time     │ 1
   │ for all good         │ 2
   │ men to come          │ 3
   │ to hte aid of        │ 4
 → │ 1,$p                 │ 5
   │ their country."      │ 6
   └──────────────────────┘
```

BASIC QEDX EXAMPLES

```
                                         RESULTING BUFFER

                                     "Now is the time          1
!        p                           for all good              2
         1,$p                        men to come               3
!        d                           to the aid of             4
!        1,$s/hte/the/          →    their country."           5
!        w Henry.quote
```

```
                                         RESULTING BUFFER
!        1,$d
!        a
!                -Patrick Henry  →            -Patrick Henry   1
!        \f
```

```
                                         RESULTING BUFFER

!        0r Henry.quote
!        p                            FAMOUS QUOTE:            1
         their country."                                       2
!        a                      →                              3
!                                     "Now is the time         4
!        \f                           for all good             5
!        1i                           men to come              6
!        FAMOUS QUOTE:                to the aid of            7
!                                     their country."          8
!                                                              9
!        \f                                 -Patrick Henry    10
!        w Henry.quote
```

YOU ARE NOW READY FOR WORKSHOP
#2

This page has intentionally
been left blank.

TOPIC IV

MORE QEDX

This page has intentionally
been left blank.

QEDX TERMINOLOGY

- REGULAR EXPRESSION:

  - ANY NUMBER OF CHARACTERS (INCLUDING NONE) DELIMITED BY A RIGHT SLANT AND OBEYING A CERTAIN SYNTAX

```
/abc/               /a.c/

/old/               /old.*$/

/calendar/          /^calendar/
```

- SEARCH EXPRESSION:

  - A REGULAR EXPRESSION IMMEDIATELY FOLLOWING A qedx REQUEST. (/abc/ IN THESE EXAMPLES)

```
s/abc/xyz/

1,25s/abc//

/abc/
```

- REPLACE EXPRESSION:

  - THE SECOND CHARACTER STRING IN A SUBSTITUTE REQUEST. (/abc/ IN THESE EXAMPLES)

```
s/old/abc/

1,5s/old/abc/
```

SPECIAL SYMBOLS

● / (RIGHT SLANT -OR- SLASH)

- DELIMITS A REGULAR EXPRESSION

```
/abc/

/^character/

/ters.*$/
```

● . (PERIOD)

- USAGE 1: AS PART OF AN ADDRESS IN A qedx REQUEST

  - ADDRESSES THE CURRENT LINE

```
1,.d

.,49p
```

- USAGE 2: AS A CHARACTER IN A SEARCH EXPRESSION

  - MATCHES ANY CHARACTER

```
/a.c/       MATCHES:  aac
                      abc
                      azc
                      a c
                      ...etc
```

SPECIAL SYMBOLS

● * (ASTERISK)

▯ HAS SPECIAL MEANING ONLY IN A SEARCH EXPRESSION

▯ MEANS ANY NUMBER (INCLUDING NONE) OF THE PRECEDING CHARACTER

```
/ab*c/    MATCHES:  ac
                    abc
                    abbc
                    abbbc
                    ...etc

/a.*c/    MATCHES:  ac
                    axc
                    axyzc
                    ...etc
```

● $ (DOLLAR)

▯ USAGE 1: AS PART OF AN ADDRESS IN A qedx REQUEST

▯ ADDRESSES THE LAST LINE OF THE USER'S BUFFER

```
1,$p
$d
```

▯ USAGE 2: AS THE LAST CHARACTER OF A SEARCH EXPRESSION

▯ MATCHES THE END OF A LINE (I.E., THE IMAGINARY CHARACTER FOLLOWING THE LAST CHARACTER OF A LINE)

```
/calendar$/    MATCHES:  calendar
```

<u>IF</u> "calendar" ENDS A LINE

SPECIAL SYMBOLS

● ^ (CIRCUMFLEX)

▯ HAS SPECIAL MEANING ONLY AS THE FIRST CHARACTER OF A SEARCH EXPRESSION

▯ MATCHES THE BEGINNING OF A LINE (I.E. AN IMAGINARY CHARACTER PRECEDING THE FIRST CHARACTER ON A LINE)

```
/^calendar/    MATCHES:  calendar
```

<u>IF</u> "calendar" BEGINS A LINE

● & (AMPERSAND)

▯ HAS SPECIAL MEANING ONLY IN THE REPLACE EXPRESSION OF A SUBSTITUTE REQUEST

▯ EACH & IS REPLACED BY THE STRING WHICH <u>MATCHED</u> THE REGULAR EXPRESSION IN THE SEARCH EXPRESSION

```
s/camp/&ing/   SAME AS:   s/camp/camping/

s/junk/"&"/    SAME AS:   s/junk/"junk"/

s/ab/&&&/      SAME AS:   s/ab/ababab/

s/a.c/&def/    SAME AS:   s/aac/aacdef/
                          s/abc/abcdef/
                          s/azc/azcdef/
                          s/a c/a cdef/
                          ...etc
```

SPECIAL SYMBOLS

- // (DOUBLE RIGHT SLANT - OR - SLASH SLASH)

  - USAGE 1: AS THE REPLACE EXPRESSION OF A SUBSTITUTE REQUEST

    - THE REPLACE EXPRESSION IS THE NULL STRING (A MEANS OF DELETING EXISTING CHARACTER STRINGS)

      ```
      s/abc//
      ```

  - USAGE 2: AS A SEARCH EXPRESSION

    - qedx REMEMBERS THE LAST SEARCH EXPRESSION DEFINED BY THE USER

    - // STANDS FOR THE LAST SEARCH EXPRESSION DEFINED

      ```
      /a.c/                   /a.c/
      s//xyz/    SAME AS:     s/a.c/xyz/
      //                      /a.c/
      ```

- \c (LEFT SLANT C -OR- BACKSLASH C)

  - THE SEQUENCE \c IS A qedx ESCAPE SEQUENCE THAT CAUSES THE CHARACTER THAT FOLLOWS TO BE INTERPRETED AS A LITERAL

  - SUPPRESSES (ESCAPES) THE SPECIAL MEANING OF qedx SPECIAL SYMBOLS

    ```
    /a\c.b/

    s/\c^echpplex/^echoplex/

    s/abc/\c&def/
    ```

EXAMPLES

- INTERPRET THE FOLLOWING qedx REQUESTS:

```
1,$s/a...b//

1,$s/a...b/&&/

1,$s/a.*b/xyz/

1,$s/a..*b/xyz/

1,$s/a.\c.*b/xyz/

1,$s/a..\c*b/xyz/

1,$s/^a...b//

1,$s/^a...b$//

1,$s/./a/

1,$s/./&&/

1,$s/.*/&&/

1,$s/^a*b*c*d/abcd/

1,$s/^a.*b.*c.*d/abcd/

1,$s/^ *//

1,$s/^  */     ■&/

1,$s/^.$/&&&/
```

ADDRESSING

- qedx REQUEST MAY TAKE ONE OF THREE GENERAL FORMS

  - <request>

  - ADR<request>

  - ADR,ADR<request>

- ADDRESSES MAY TAKE ONE OF THREE GENERAL FORMS

  - ABSOLUTE LINE NUMBERS

```
5d
$p
10,12s/abc/xyz/
3
```

  - RELATIVE LINE NUMBERS (RELATIVE TO "CURRENT LINE")

```
.+4d
+4d
-2,.p
-2,+5s/abc/xyz/
$-4,$p
-2
```

ADDRESSING

- CONTEXTUAL ADDRESSING
  - AN ADDRESS MAY BE A REGULAR EXPRESSION
  - A LINE CAN BE ADDRESSED BY MATCHING REGULAR EXPRESSIONS TO STRINGS IN THE LINE

```
/abc/d

/abc/,25p

/abc/+2,25p

/abc/+2,+25p

/^abc/,/xyz/s/ab/acc/

-4,/x.z/p

/^ab*c/+2,/x.z/-3s/boat /boating /
```

ADDITIONAL BUFFERS

- THE USER'S INITIAL BUFFER (SCRATCH PAD) HAS A RESERVED NAME OF b(0)

- USERS MAY DEFINE AN ARBITRARY NUMBER OF ADDITIONAL BUFFERS BY SIMPLY REFERRING TO THEM BY SOME CHOSEN NAME

BUFFER REQUESTS	DESCRIPTION
m(name)	MOVE: MOVE THE SPECIFIED LINE OR LINES IN THE CURRENT BUFFER TO A BUFFER HAVING THE SPECIFIED NAME
\b(name)	REPRESENTS THE CONTENTS OF THE SPECIFIED BUFFER
b(name)	BUFFER: CHANGE BUFFERS. MAKE THE SPECIFIED BUFFER THE "CURRENT" BUFFER
x	STATUS: PRINT A SUMMARY OF THE STATUS OF ALL BUFFERS

- CUT AND PASTE EXAMPLE. MOVE LINES 14 THROUGH 17 IMMEDIATELY BELOW LINE 10

```
14,17m(1)
10a
\b(1)\f
```

AREAS FOR ADDITIONAL STUDY

- ADDITIONAL DOCUMENTATION OF qedx

  - MPM COMMANDS AND ACTIVE FUNCTIONS (AG92)

  - NEW USER'S GUIDE (AL40)

  - help qedx

- STUDY TOPICS

  - MULTIPLE REQUEST ON A LINE

```
dp

s/abc/xyz/p

s/abc/xyz/w file_13

\fw
```

AREAS FOR ADDITIONAL STUDY

- ADDITIONAL qedx EDIT REQUESTS

g (global) ⇒ all

```
gp/xyz/

1,10gd/xyz/

10,20g=/xyz/
```

v (exclude)

```
vp/xyz/

1,10vd/xyz/

1,.v=/xyz/
```

n (nothing)

```
5n
```

" (comment)

```
"This is a comment in a macro
```

AREAS FOR ADDITIONAL STUDY

ADDRESSING USING ";" INSTEAD OF ","

```
ADR;ADR<request>

/abc/;+5d
```

- DELIMITERS OTHER THAN "/"

- RESERVED BUFFERS: b(0), b(exec), b(args)

- qedx MACRO FACILITY

  - BY TYPING THE FOLLOWING COMMAND LINE:

    ```
    qedx  my_macro.qedx  add
    ```

    THE FOLLOWING MACRO IS INVOKED WITH BUFFER b(args) CONTAINING THE ARGUMENT add

my_macro.qedx

```
r \b(args).fortran
1,$s/write.*)/print/
w
q
```

TOPIC V

THE STORAGE SYSTEM

This page has intentionally
been left blank.

- SYSTEM GOALS:

  - DECENTRALIZATION OF THE SYSTEM'S LOGICAL FILE SPACE

  - EASE OF USE

- EFFECT ON STORAGE SYSTEM

  - HIERARCHY OF DIRECTORIES

  - PATHNAME CONVENTION

    ```
    >udd>FED>LJones>tools>my_editor
    ```

  - WORKING DIRECTORY CONCEPT

    ```
    change_wdir >udd>FED>LJones>tools
    ```

SEGMENTS

(FILE - outside peripheral device)
different from segment

- SEGMENT
  - THE BASIC UNIT OF INFORMATION STORAGE
  - SOMETIMES REFERRED TO AS A FILE
  - SEGMENTS RESIDE ON DISK PACKS (SECONDARY STORAGE)
  - SIZE IS INTEGER NUMBER OF RECORDS (1024 WORDS). 0,1,2,3...
  - MAXIMUM SIZE IS 256 RECORDS (256K WORDS)

- ALL SEGMENTS HAVE AT LEAST ONE GIVEN NAME (ENTRYNAME)

```
home_work_3
add.pl1
start_up.ec
```

SEGMENTS

- MUST DISTINGUISH BETWEEN THE CONTENTS OF A SEGMENT AND THE ATTRIBUTES OF A SEGMENT

  - THE CONTENTS OF A SEGMENT MAY BE:

    - DATA (EITHER RAW OR FORMATTED)
    - TEXT (USUALLY ASCII)
    - SOURCE OR OBJECT PROGRAM
    - EMPTY (ZERO LENGTH)

  - SOME ATTRIBUTES OF A SEGMENT ARE:

    - THE NAME(S) OF THE SEGMENT (CALLED ENTRYNAME(S))
    - THE SEGMENTS UNIQUE IDENTIFIER (A UNIQUE, 36 BIT, INTERNALLY USED NAME)
    - THE AUTHOR (I.E., THE user_id OF THE CREATOR)
    - THE LENGTH (IN BITS) OF THE SEGMENTS CONTENTS
    - THE BIT COUNT AUTHOR (I.E. THE user_id OF THE LAST PERSON TO MODIFY THE CONTENTS)
    - THE ACCESS CONTROL LIST SPECIFYING WHO MAY ACCESS THE SEGMENT AND HOW THEY MAY ACCESS IT
    - THE AMOUNT OF DISK SPACE (IN RECORDS) OCCUPIED BY THE SEGMENT

SEGMENTS

- DATE AND TIME SEGMENT'S CONTENTS WERE LAST MODIFIED

- DATE AND TIME THE SEGMENT'S ATTRIBUTES WERE LAST MODIFIED

- DATE AND TIME SEGMENT'S CONTENTS WERE LAST DUMPED (I.E., COPIED TO TAPE BY THE MULTICS BACKUP PROCEDURES)

- DATE AND TIME SEGMENT'S CONTENTS WERE LAST REFERENCED

- THE STATE OF THE SEGMENT'S COPY SWITCH

- THE STATE OF THE SEGMENT'S SAFETY SWITCH

● THE CONTENTS OF A SEGMENT MAY BE READ BY USING THE print COMMAND OR qedx's "r" AND "p" REQUESTS

● THE ATTRIBUTES OF A SEGMENT MAY BE OBTAINED BY USING THE list OR THE status COMMAND

● THE ACCESSING OF A SEGMENT'S CONTENTS AND A SEGMENT'S ATTRIBUTES ARE INDEPENDENTLY CONTROLLED BY THE MULTICS ACCESS CONTROL MECHANISM

SEGMENTS

CONTENTS

```
                              0
 The Multics System i
 s a general purpose
 computer system deve
 loped by the\012Mass
 achusetts  Institute
  of Technology and H
 oneywell  Infor-\012
 mation Systems.  Int
 roduced to commercia  1
 l markets\012in Janu
 ary 1973,  Multics w
 as then the result o
 f more than 7 years
 of research.\012



                              2
```

ATTRIBUTES	ATTRIBUTE VALUE
names:	home_work_3, hw3
author:	May.FED.a
access:	r w May.FED.* r w *.SysDaemon.*
bit count:	49698
records used:	2
date modified:	03/21/77 1034.4 mst
date dumped:	03/21/77 1051.0 mst
date used:	09/07/77 0818.9 mst

- MAY HAVE UP TO 19 MILLION SEGMENTS IN A SYSTEM (512 MSU * 38,000 SEGMENTS)

- ~~FOR ORGANIZATION REASONS,~~ SOME SEGMENTS CONTAIN THE NAMES, ADDRESSES AND ATTRIBUTES OF OTHER SEGMENTS

  - THESE SEGMENTS SERVE AS A CATALOG OF THE OTHER SEGMENTS

  - THESE "CATALOG" SEGMENTS ARE CALLED DIRECTORIES

DIRECTORIES

- DIRECTORY

  - A SEGMENT CONTAINING THE NAMES, ADDRESSES AND ATTRIBUTES OF OTHER SEGMENTS AND/OR OTHER STORAGE SYSTEM ENTITIES

  - A MEANS OF ORGANIZING (CATALOGING) SEGMENTS AND/OR OTHER STORAGE SYSTEM ENTITIES

  - SPECIFICALLY, A DIRECTORY MAY BE A CATALOG OF THE FOLLOWING STORAGE SYSTEM ENTITIES:

    - FILES

      - SEGMENTS (SINGLE SEGMENT FILES)

      - MULTISEGMENT FILES

    - OTHER DIRECTORIES

    - LINKS

    - MAY BE EMPTY

- ALL DIRECTORIES HAVE AT LEAST ONE GIVEN NAME (ENTRYNAME)

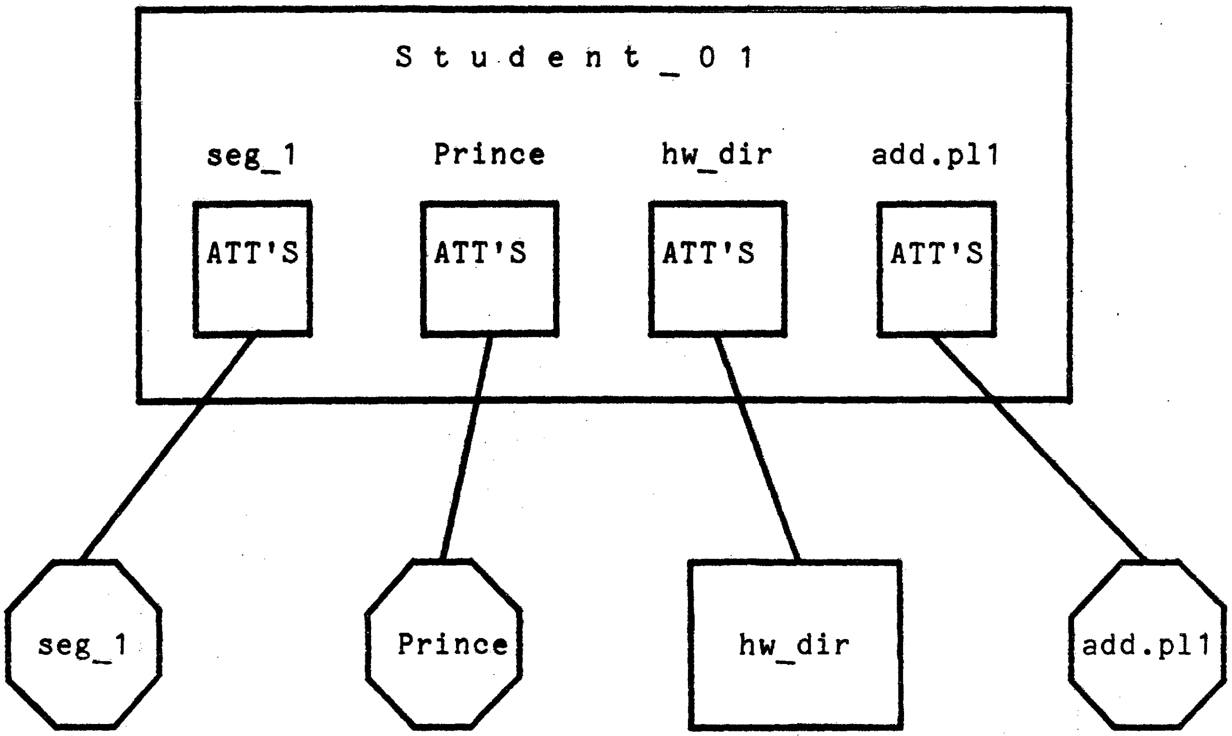
  udd user directory directories

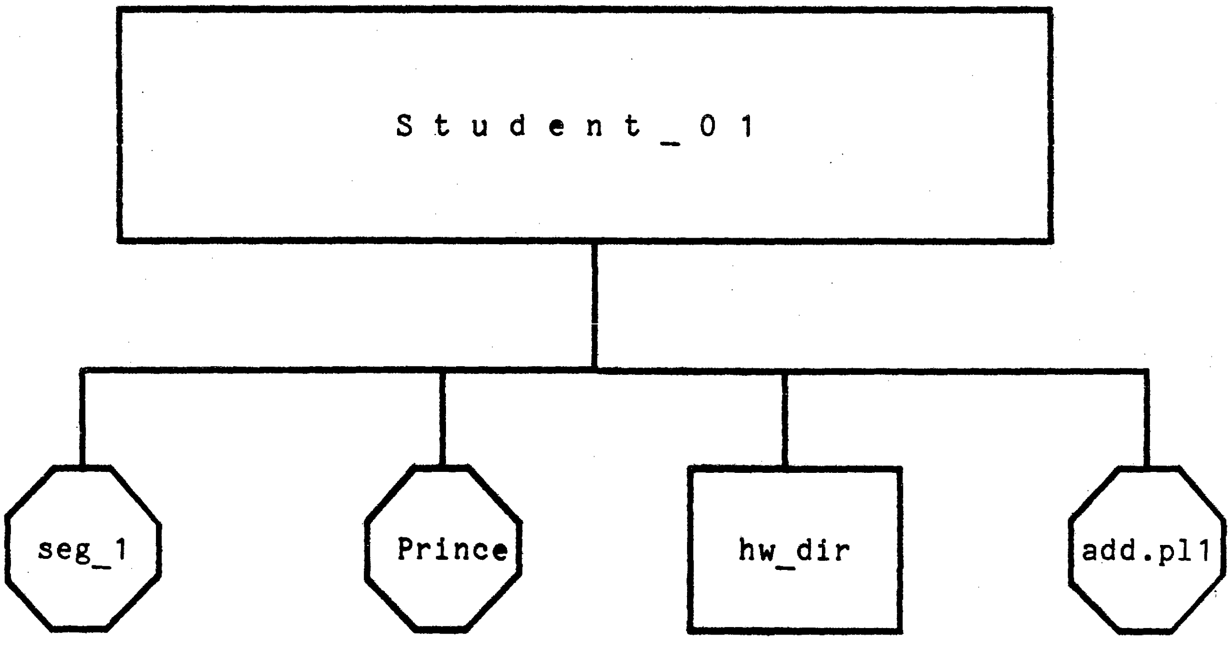
  F01

  Dir_A

  system_library_unbundled

<u>DIRECTORIES</u>





STORAGE SYSTEM CONTROL

- MAY HAVE UP TO 19 MILLION SEGMENTS IN A SYSTEM, SOMEONE MUST CONTROL:

  - THE ALLOCATION OF DISK SPACE TO USERS (WHO GETS WHAT?)

  - THE CREATION AND USE OF SEGMENTS (I'VE GOT A SECRET!)

  - THE SHARING OF SEGMENTS (YOU MAY USE MINE TOO!)

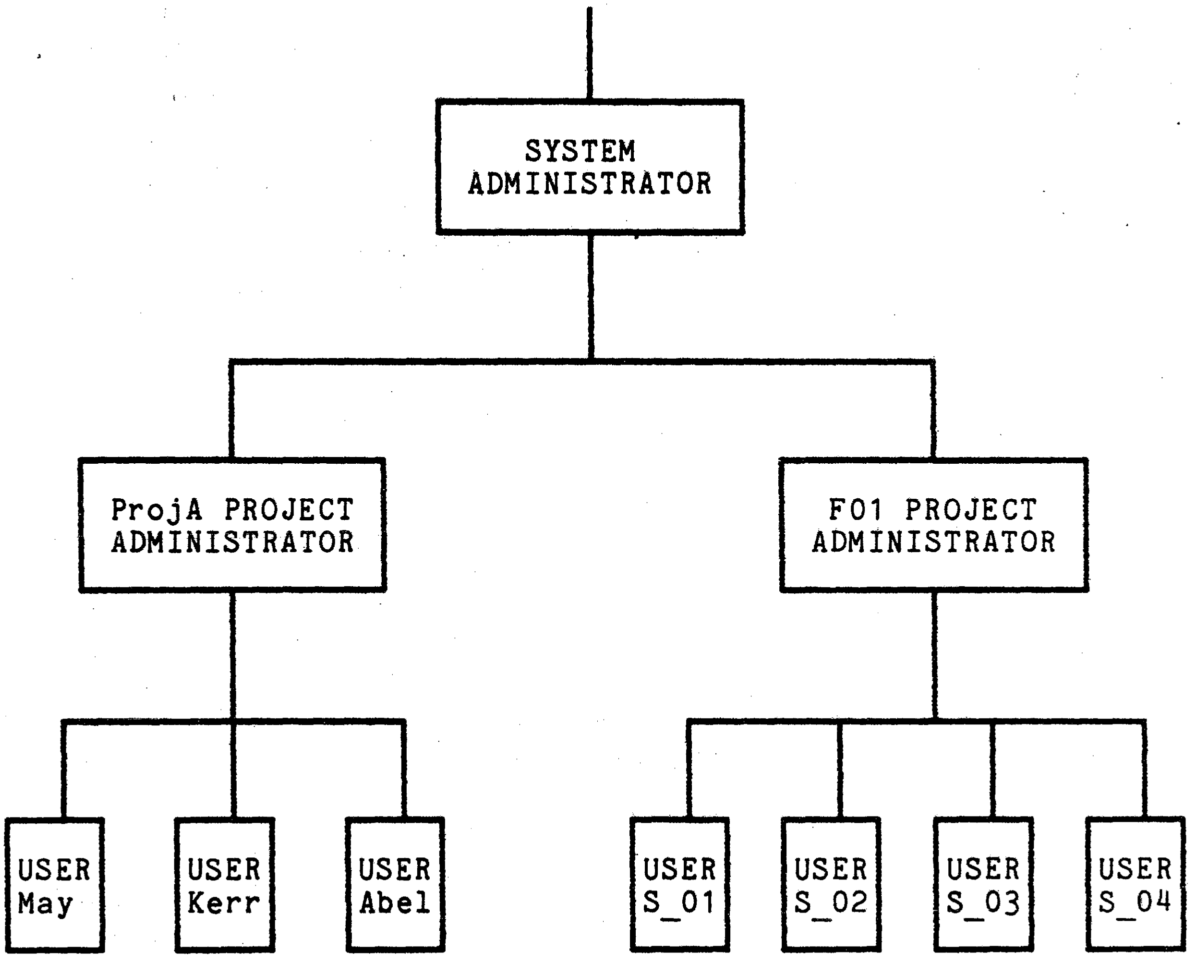
- HIERARCHY OF ADMINISTRATORS

  - THE SYSTEM ADMINISTRATOR

  - PROJECT ADMINISTRATORS

  - USERS

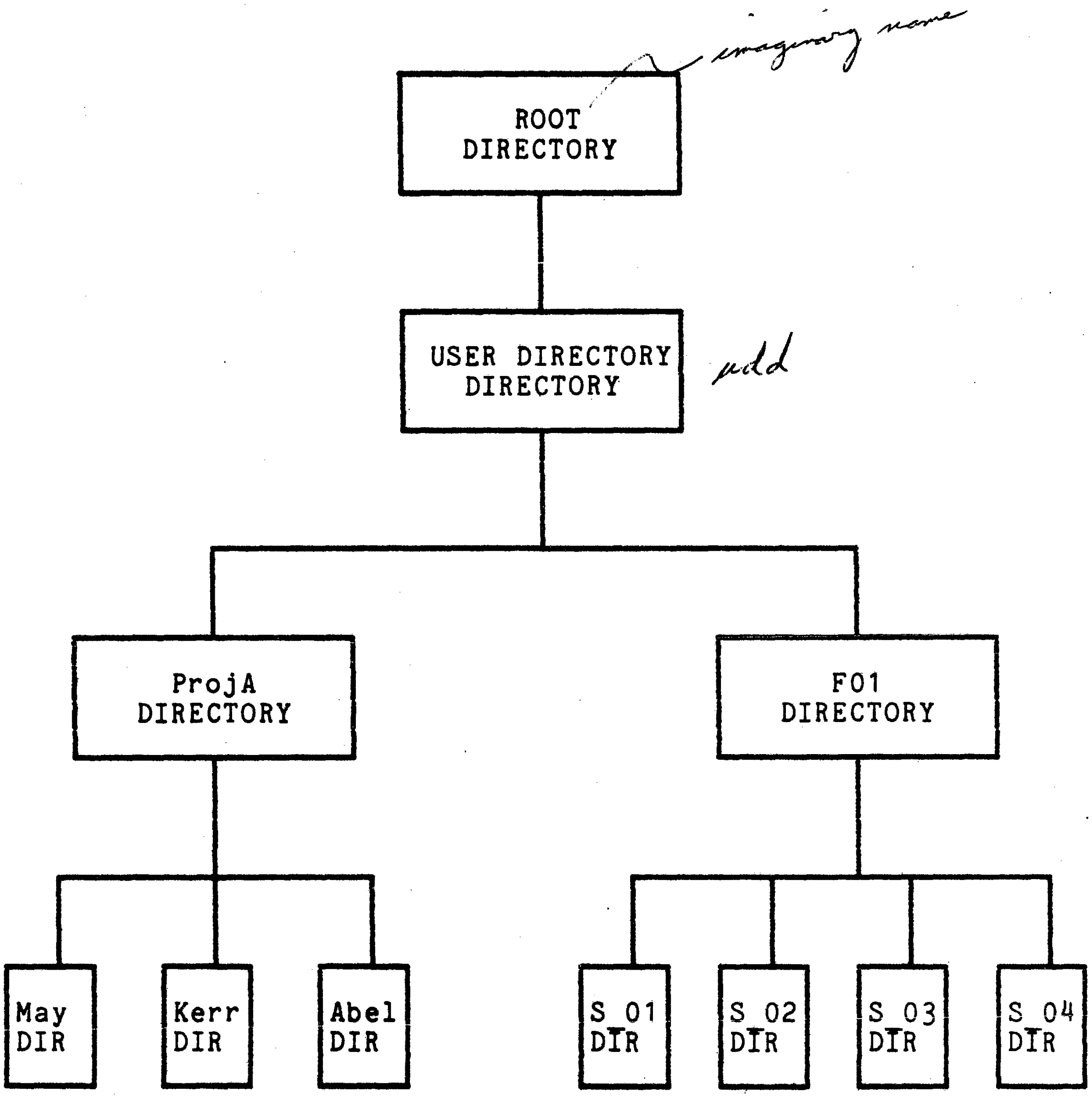
STORAGE SYSTEM CONTROL



STORAGE SYSTEM HIERARCHY

- NEED TO PARTITION THE FAMILIES OF DIRECTORIES AND SEGMENTS INTO MANAGEABLE COMMUNITIES

- STORAGE SYSTEM IS MAPPED ONTO ADMINISTRATION HIERARCHY



STORAGE SYSTEM HIERARCHY

- THE STORAGE HIERARCHY IS ANALOGOUS IN FORM TO AN INVERTED TREE

- THE USER DIRECTORY DIRECTORY (udd) EMANATES FROM THE ROOT DIRECTORY

- ALL PROJECT DIRECTORIES EMANATE FROM THE USER DIRECTORY DIRECTORY (udd)

- ALL USER DIRECTORIES EMANATE FROM THEIR RESPECTIVE PROJECT DIRECTORIES

- USERS MAY ARBITRARILY CREATE DIRECTORIES SUBORDINATE TO THEIR OWN USER DIRECTORY - UP TO A MAXIMUM DIRECTORY DEPTH OF 16 (TOP TO BOTTOM)

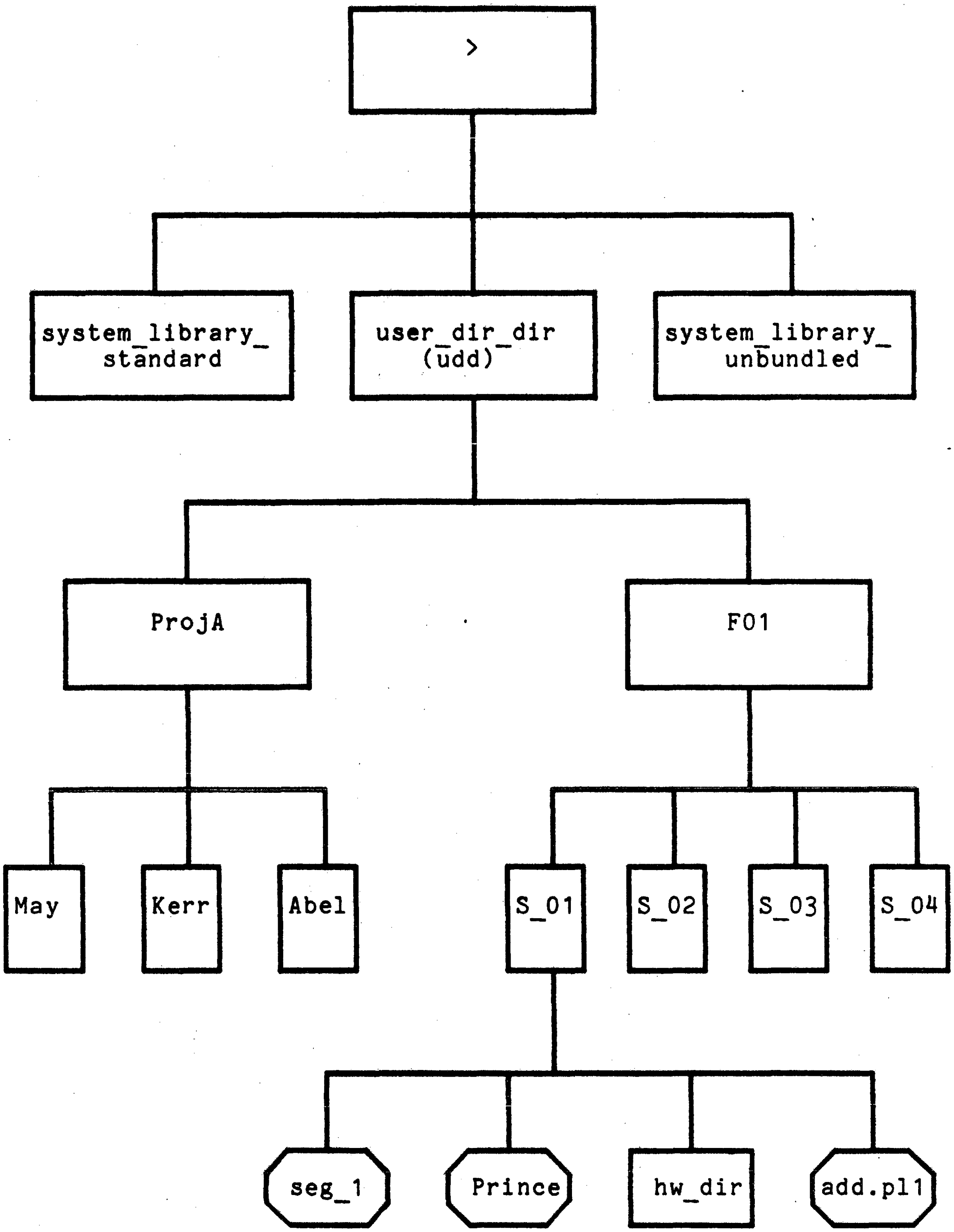
- A SEGMENT WHOSE POSITION IS IMMEDIATELY BELOW A GIVEN DIRECTORY IS SAID TO BE:

  "In the directory"

  - OR -

  "Under the directory"

STORAGE SYSTEM HIERARCHY



PATHNAME CONVENTIONS

- ABSOLUTE PATHNAME

  - A PATHNAME THAT UNIQUELY IDENTIFIES A SEGMENT (OR DIRECTORY) BY ITS ABSOLUTE POSITION IN THE DIRECTORY HIERARCHY

  - FORMED BY CONCATENATING A SEGMENT'S (OR DIRECTORY'S) ENTRYNAME WITH ALL SUPERIOR DIRECTORIES LEADING BACK TO THE ROOT

    ```
    >udd>F01>Student_01>add.pl1
    ```

  - THE > (GREATER-THAN) CHARACTER IS USED TO SEPARATE THE ENTRYNAMES IN A PATHNAME

  - AN ABSOLUTE PATHNAME ALWAYS BEGINS WITH > (GREATER-THAN)

    - DIRECTORY ABSOLUTE PATHNAMES

      ```
      >udd>ProjA

      >udd>F01>Student_01

      >udd>F01>Student_01>hw_dir
      ```

    - SEGMENT ABSOLUTE PATHNAMES

      ```
      >udd>F01>Student_01>add.pl1

      >udd>F01>Student_01>hw_dir>lesson_2

      >udd>ProjA>Kerr>start_up.ec
      ```

PATHNAME CONVENTIONS

- NOTICE THAT AN ABSOLUTE PATHNAME SUCH AS

  ```
  >udd>ProjA>Kerr>start_up.ec
  ```

  IDENTIFIES:

  - THE ENTRYNAME OF THE SEGMENT (start_up.ec)

  - THE Person_id OF THE "OWNER" (Kerr)

  - THE Project_id OF THE "OWNER" (ProjA)

PATHNAME CONVENTIONS

- HOME DIRECTORY (WHERE THE HEART IS)

  - THE DIRECTORY IN WHICH THE USER "FINDS" HIMSELF IMMEDIATELY AFTER LOG IN

  - THE INITIAL WORKING DIRECTORY

  - THE Person_id AND Project_id GIVEN AT LOG IN DETERMINE THE HOME DIRECTORY

  - IS GENERALLY: >udd>Project_id>Person_id

    >udd>F01>Student_02

    >udd>ProjA>Abel

- WORKING DIRECTORY (WHERE THE ACTION IS)

  - THE DIRECTORY IN WHICH THE USER IS CURRENTLY WORKING (THE INITIAL WORKING DIRECTORY IS THE HOME DIRECTORY)

  - THE USER MAY CHANGE HIS WORKING DIRECTORY, AS DESIRED, TO ANY OTHER DIRECTORY IN THE STORAGE SYSTEM

  - COMMANDS SUCH AS list AND qedx's "r" and "w" WILL OPERATE ON THE SEGMENTS IN THE USER'S WORKING DIRECTORY

PATHNAME CONVENTIONS

- RELATIVE PATHNAME

  - A PATHNAME THAT UNIQUELY IDENTIFIES A SEGMENT (OR DIRECTORY) BY ITS POSITION RELATIVE TO THE USER'S WORKING DIRECTORY

  - NEVER BEGINS WITH > (GREATER-THAN)

    - DIRECTORY RELATIVE PATHNAME

```
ProjA
udd>ProjA

Student_01
F01>Student_01

hw_dir
Student_01>hw_dir
F01>Student_01>hw_dir
```

    - SEGMENT RELATIVE PATHNAMES

```
add.pl1
Student_01>add.pl1

lesson_2
hw_dir>lesson_2

start_up.ec
Kerr>start_up.ec
ProjA>Kerr>start_up.ec
```

- THE WORKING DIRECTORY CONCEPT IS SIMPLY A CONVENIENCE THAT ALLOWS THE USER TO TYPE RELATIVE PATHNAMES INSTEAD OF THE LONGER ABSOLUTE PATHNAMES

< ⇒ move up one segment

PATHNAME CONVENTIONS

- ENTRYNAMES (PATHNAME MEMBER)

  - 1 TO 32 CHARACTERS LONG

  - SHOULD NOT INCLUDE > < * = ? % $ " ( ) SPACE OR TAB

  - _ (UNDERSCORE)

    - SIMULATES A SPACE FOR READABILITY

      Bobs_orig_editor.pl1

  - . (PERIOD)

    - SEPARATES COMPONENTS OF AN ENTRYNAME

      ms_tester.old.fortran

  - LAST COMPONENT OF AN ENTRYNAME IS CALLED THE SUFFIX

  - ENTRYNAMES MUST BE UNIQUE WITHIN A DIRECTORY

  - SEGMENTS AND DIRECTORIES MAY HAVE MORE THAN ONE ENTRYNAME

test.14_may.new_compiler	homework_dir
t.14m.nc	hw_dir
may_comp	hw

PATHNAME CONVENTIONS

- NOTICE THAT IN THE SIMPLEST CASE (WHERE A USER LOGS IN, CREATES AND EDITS HIS HOME DIRECTORY FILES) NO KNOWLEDGE OF THE HIERARCHY IS REQUIRED

TOPIC VI

STORAGE SYSTEM COMMANDS

This page has intentionally
been left blank.

DIRECTORY MANIPULATION COMMANDS

- print_wdir, pwd (WHERE AM I?)

  ▯ PRINTS THE PATHNAME OF THE CURRENT WORKING DIRECTORY

  ▯ USAGE:

  ```
  print_wdir
  pwd
  ```

- change_wdir, cwd (MOVE OUT!)

  ▯ CHANGES THE USER'S WORKING DIRECTORY

  ▯ USAGE:

  ```
  change_wdir  {path}
  cwd  dir_A
  cwd  >udd>F01>Student_08>dir_A
  cwd
  ```

DIRECTORY MANIPULATION COMMANDS

- create_dir, cd (UP TO 16 DEEP)

  - CREATES AN EMPTY DIRECTORY

  - DOES NOT CHANGE THE USER'S WORKING DIRECTORY

  - USAGE:

    ```
    create_dir paths {-control_args}

    cd dir_A

    cd >udd>F01>Student_09>myd
    ```

- delete_dir, dd (DO YOU REALLY...?)

  - DELETES (DESTROYS) SPECIFIED DIRECTORIES (AND ALL SUBORDINATE DIRECTORIES AND SEGMENTS)

  - USAGE:

    ```
    delete_dir paths

    dd programs

    dd >udd>F01>Student_01>programs
    ```

SEGMENT MANIPULATION COMMANDS

- create, cr (SELDOM NEEDED)

  ▯ CREATES AN EMPTY SEGMENT

  ▯ USAGE:

  ```
  create paths
  cr seg_1
  cr seg_1 seg_2 A B
  cr >udd>F01>Student_07>add.pl1
  ```

- delete, dl

  ▯ DELETES (DESTROYS) SPECIFIED SEGMENTS

  ▯ USAGE:

  ```
  delete paths
  dl seg_1 add.pl1
  ```

SEGMENT MANIPULATION COMMANDS

- copy, cp

does not need an existing segment

  - COPIES A SPECIFIED SEGMENT TO A NEW POSITION IN THE HIERARCHY

  - DOES NOT COPY A SEGMENT'S "ADD NAMES" UNLESS REQUESTED

  - USAGE:

```
copy path1 {path2} {-control_args}

cp >udd>FED>LJones>add  >udd>FED>LJones>exp>add

cp >udd>FED>LJones>add  add.old

cp >udd>FED>LJones>add

cp >udd>FED>LJones>add  -name
```

- move

- segment is destroyed
- again, does not need an existing segment

  - MOVES A SPECIFIED SEGMENT (TO INCLUDE ACL AND ADD NAMES) TO A NEW POSITION IN THE HIERARCHY

  - USAGE:

```
move path1 {path2} {-control_args}

move >udd>FED>Kerr>dev>x_sort >udd>FED>Kerr>tools>sort

move >udd>FED>Kerr>dev>x_sort sort

move >udd>FED>Kerr>dev>x_sort
```

SEGMENT MANIPULATION COMMANDS

must include original segment name first then other alias's

- add_name, an (ALIAS)

  ▯ ADDS ALTERNATE NAME(S) TO A SEGMENT OR DIRECTORY

  ▯ SUCH NAMES ARE CALLED "ADD NAMES"

```
▯ USAGE:    add_name path names
            an seg_1.new s_1.n s1n
            an >udd>F01>Student_01>seg_1.new s1n
```

same as an for procedure

- delete_name, dn

  ▯ DELETES NAME(S) FROM SEGMENTS AND DIRECTORIES

```
▯ USAGE:    delete_name paths
            dn seg_1.new s1n
```

SEGMENT MANIPULATION COMMANDS

- rename, rn

  - REPLACES A SEGMENT OR DIRECTORY NAME WITH ANOTHER

  - USAGE:

    ```
    rename path1 name1...pathn namen

    rn s_1.n seg_1.new
    ```

- list, ls (ROLL CALL)

  - RETURNS ATTRIBUTE INFORMATION ABOUT STORAGE SYSTEM ENTITIES
    (permission, page length, all alias's

  - ★ BY **DEFAULT** ONLY **SEGMENTS** ARE LISTED ~~[illegible]~~

    see manual

  - USAGE:

    ```
    list {entrynames} {-control_args}

    ls

    ls add.pl1 seg_1

    ls -dr
    ```

    directories only - how many, names, permissions

    ```
    ls -all -sort name

    list -date_time_contents_modified
    ```

SEGMENT MANIPULATION COMMANDS

- status, st (WHO, WHAT, WHERE, AND WHEN)

  - RETURNS STATUS INFORMATION ABOUT SEGMENTS AND DIRECTORIES, INCLUDING

    - DATE AND TIME MODIFIED, USED AND DUMPED

    - User_id OF AUTHOR AND User_id OF LAST MODIFIER

    - SIZE, ACCESS CLASS, ACCESS MODES, RING BRACKETS

  - USAGE:

    ```
    status paths {-control_args}

    st seg_1

    st seg_1 -length

    st seg_2 -author -date
    ```

- print, pr (...LET'S SEE WHAT YOU LOOK LIKE)

  - PRINTS THE CONTENTS OF A SEGMENT

  - USAGE:

    ```
    print path {begin} {end}

    pr seg_1

    pr add.pl1 150

    pr prince   40   120

    pr >udd>F01>Student_08>add.pl1
    ```

SEGMENT MANIPULATION COMMANDS

- compose, comp (PLASTIC SURGERY)

  - FORMATS TEXT SEGMENTS INTO MANUSCRIPT FORM

  - SEGMENTS MUST HAVE SUFFIX OF compin AND NORMALLY CONTAIN CONTROL STATEMENTS WHICH DRIVE THE FORMATTING

  - IF OUTPUT IS DIRECTED TO A SEGMENT, THE ENTRYNAME IS GIVEN A SUFFIX OF compout

  - THIS COMMAND REPLACES THE runoff COMMAND, AND PROVIDES A SUPERSET OF THE runoff CAPABILITIES TO INCLUDE INLINE ARTWORK

  - USAGE:

```
compose paths {-control_args}

comp thesis.compin

comp thesis.compin -in 10 -of -dv dtc300s -pass 2
```

SEGMENT MANIPULATION COMMANDS

- dprint, dp (START THE PRESSES!)

  - QUEUES A REQUEST TO PRINT THE CONTENTS OF A SPECIFIED SEGMENT ON THE LINE PRINTER

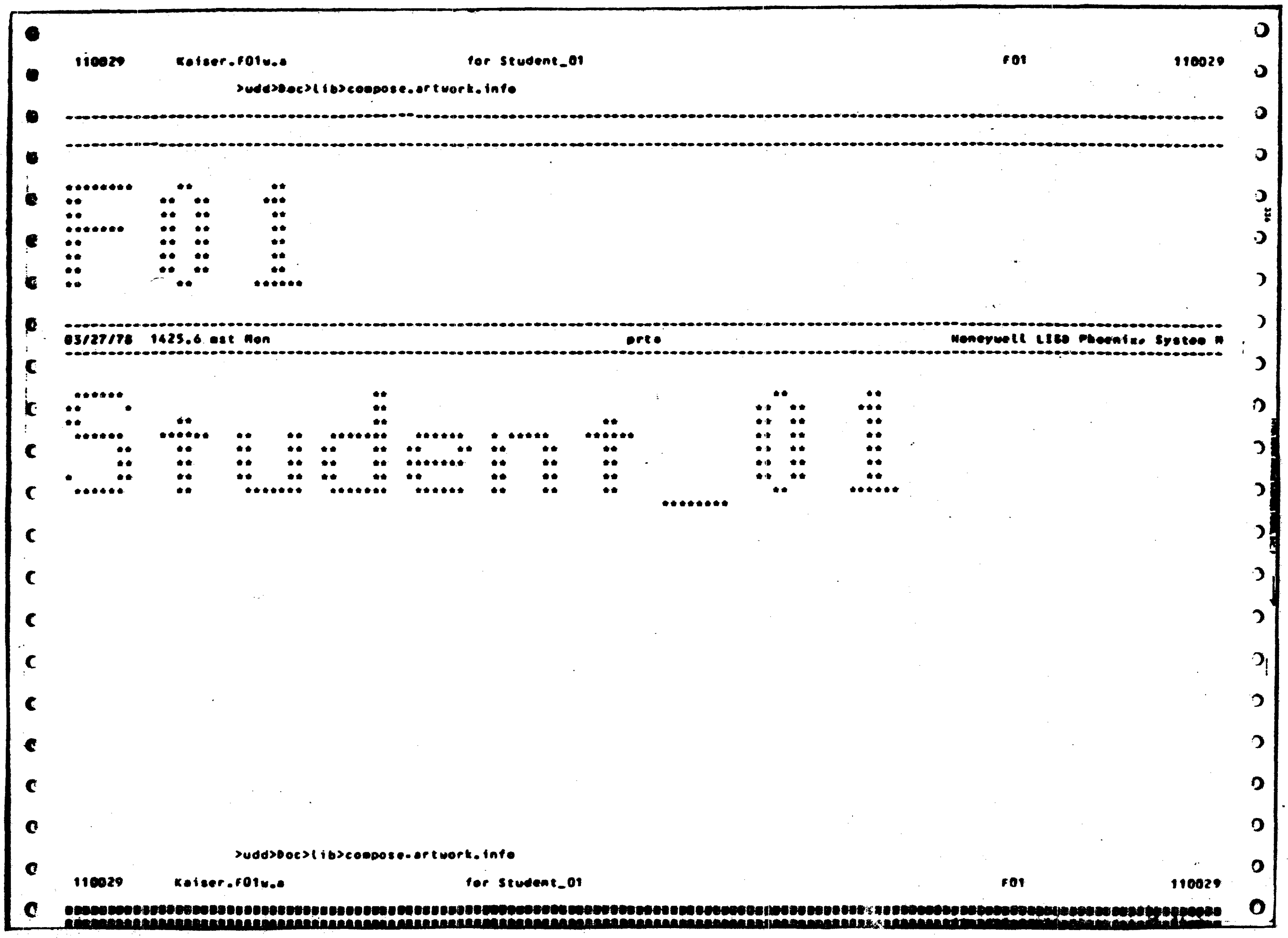
  - THE USER MAY SPECIFY ONE OF THREE PRIORITY QUEUES (QUEUE "3" IS ASSUMED - LOWEST PRIORITY, LOWEST COST)

  - THE PRINTING IS DONE BY ONE OF THE SYSTEM DAEMONS (A SERVICE PROCESS)

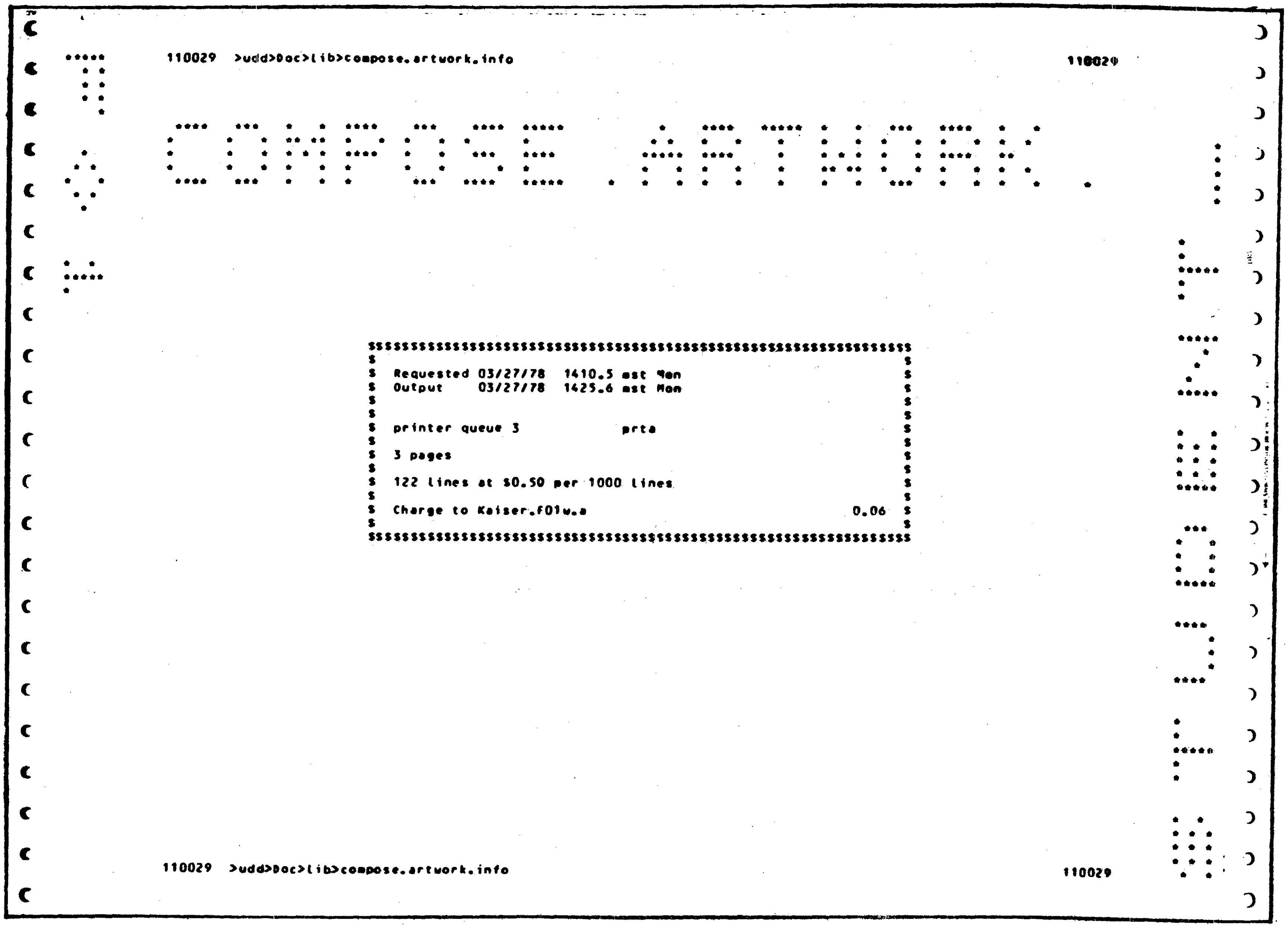
  - USAGE:

```
USAGE:     dprint {-control_args} {paths}

           dp seg_1

           dp -cp 4 -ds MD_104 seg_1

           dp seg_1 -cp 3 seg_2 -ds Bldg_4 seg_3

           dp -he "Tom Smith" -notify seg_1 add.pl1

           dp -delete -q 1 Prince
```

  - DEFAULTS: -cp 1, -ds Project_id, -he Person_id, -q 3



```
110029      Kaiser.F01w.a                       for Student_01                                            F01            110029
                     >udd>Doc>lib>compose.artwork.info
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------

F01

--------------------------------------------------------------------------------------------------------------------------------
03/27/78  1425.6 mst Mon                                   prta                                 Honeywell LISD Phoenix, System M
--------------------------------------------------------------------------------------------------------------------------------

Student_01

                     >udd>Doc>lib>compose.artwork.info
110029      Kaiser.F01w.a                       for Student_01                                            F01            110029
```


COMPOSE.ARTWORK.

```
110029  >udd>Doc>lib>compose.artwork.info                                   110029

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$                                                                          $
$  Requested 03/27/78  1410.5 mst Mon                                      $
$  Output    03/27/78  1425.6 mst Mon                                      $
$                                                                          $
$                                                                          $
$  printer queue 3           prta                                          $
$                                                                          $
$  3 pages                                                                 $
$                                                                          $
$  122 lines at $0.50 per 1000 lines                                       $
$                                                                          $
$  Charge to Kaiser.F01w.a                                          0.06   $
$                                                                          $
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

110029  >udd>Doc>lib>compose.artwork.info                                   110029
```

- dpunch, dpn (START THE PUNCHES!)

  - QUEUES A REQUEST TO PUNCH THE CONTENTS OF A SPECIFIED SEGMENT ON THE CARD PUNCH

  - CONTROL ARGUMENTS AND DEFAULTS ARE THE SAME AS THE dprint COMMAND'S

  - USAGE:

    ```
    dpunch {-control_args} {paths}

    dpn -he Larry -ds "Room 21" prince
    ```

- list_daemon_requests, ldr

  - PRINTS INFORMATION ABOUT OUTSTANDING dprint AND dpunch REQUESTS IN A SPECIFIED QUEUE (DEFAULT IS -queue 3)

  - USAGE:

    ```
    list_daemon_requests {-control_args}

    ldr

    ldr -queue 1
    ```

SEGMENT MANIPULATION COMMANDS

- cancel_daemon_request, cdr

  - CANCELS A dprint OR dpunch REQUEST IN THE QUEUE SPECIFIED (DEFAULT IS -queue 3)

  - USAGE:

```
USAGE:  cancel_daemon_request request_id {-control_args}

        cdr prince

        cdr -id 202008

        cdr -entry prince
```

- walk_subtree, ws (AND DON'T COME BACK UNTIL YOU ARE THROUGH!)

  - EXECUTES ANY SUPPLIED COMMAND LINE IN A SPECIFIED DIRECTORY, AND ALL INFERIOR DIRECTORIES

  - A LIBRARY MANAGEMENT TOOL

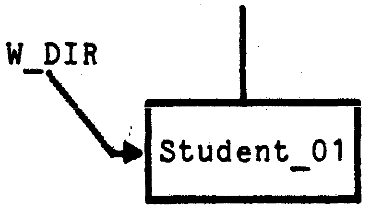
  - USAGE:

```
USAGE:  walk_subtree path "command_line" {-control_args}

        ws >udd>F01 "list -all"

        ws -wd "sa ** r *.*.*"

        ws -wd "da LJones.*.*" -bottom_up
```

starting at WORKING DIRECTORY

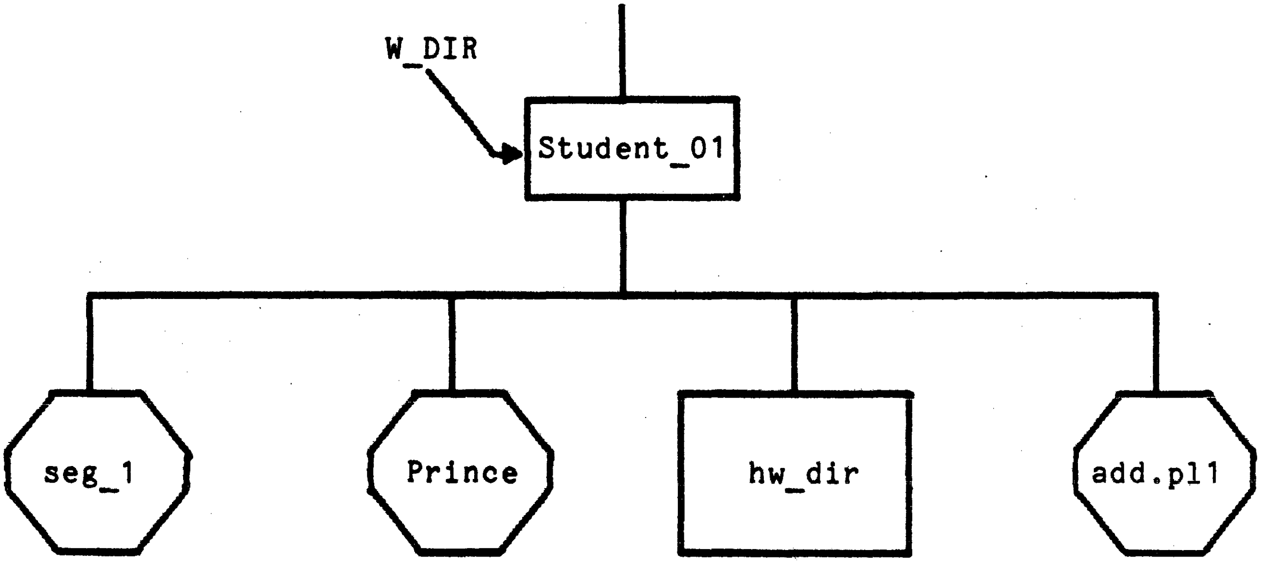
STORAGE SYSTEM EXAMPLES

NOTE: THE READY MESSAGE IS NOT SHOWN IN THE FOLLOWING TERMINAL SESSION



segments only

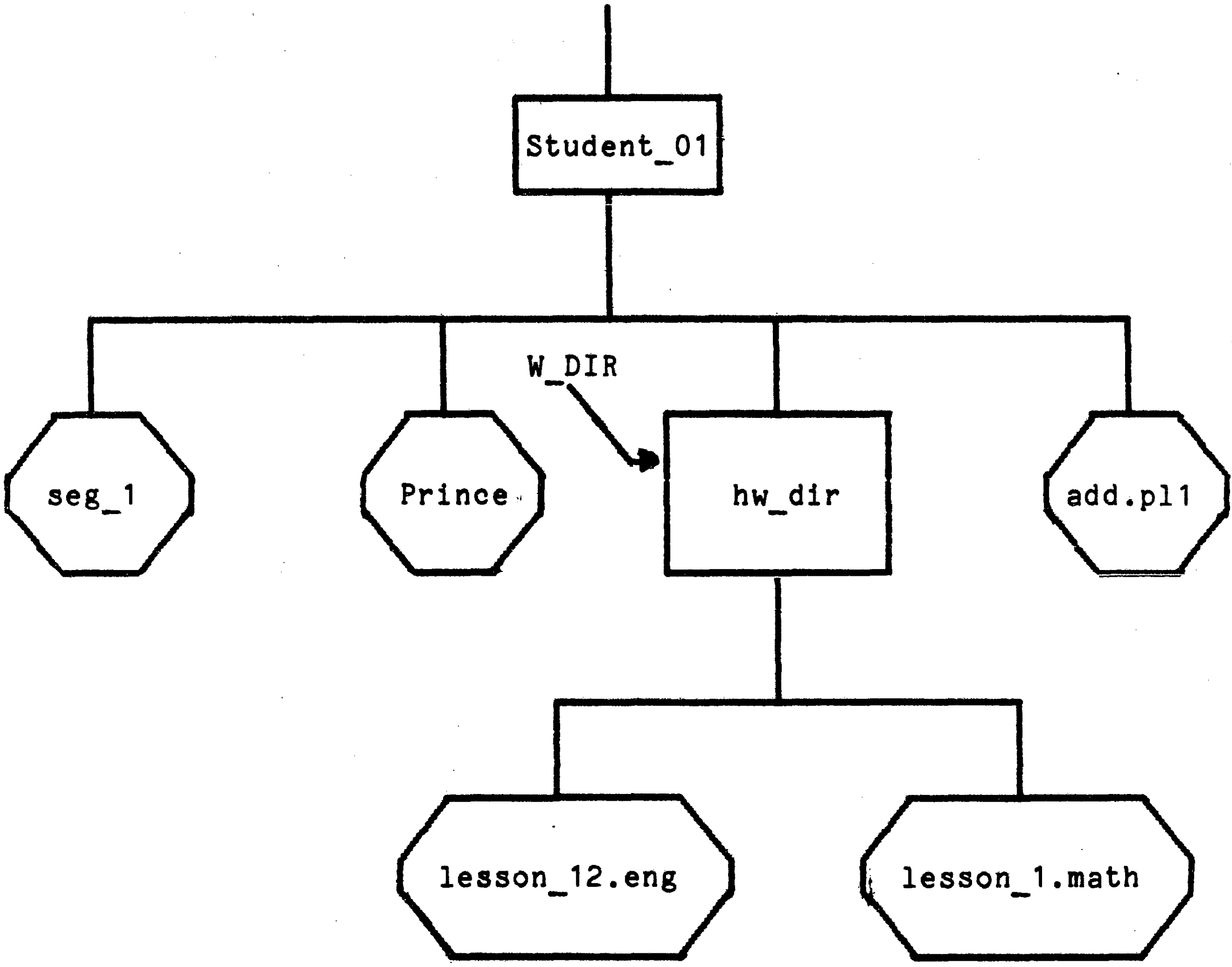
```
!    pwd                                                  (print_wdir)
     >udd>F01>Student_01                                  (output)

!    ls                                                   (list)
     Directory empty.                                     (output)

!    qx                                                   (qedx)
!    a                                                    (append mode)
!    add: proc                                            (text)
!    end add                                              (text)
!    \f                                                   (edit mode)
!    w add.pl1                                            (write)
!    q                                                    (quit)

!    cr seg_1 Prince                                      (create)

!    cd hw_dir                                            (create_dir)
```



STORAGE SYSTEM EXAMPLES

```
!    pwd                                        (print_wdir)
     >udd>F01>Student_01                        (output)

!    ls                                         (list)
                                                (output)
     Segments = 3, Lengths = 1.                 (output)
                                                (output)
     r w     0  Prince                          (output)
     r w     0  seg_1                           (output)
     r w     1  add.pl1                         (output)

!    ls -sort name -all                         (list)
                                                (output)
     Segments = 3, Lengths = 1.                 (output)
                                                (output)
     r w     1  add.pl1                         (output)
     r w     0  seg_1                           (output)
     r w     0  Prince                          (output)
                                                (output)
     Directories = 1.                           (output)
                                                (output)
     sma   hw_dir                               (output)

!    cr hw_dir>lesson_1.math                    (create)

!    cwd hw_dir                                 (change_wdir)

!    pwd                                        (print_wdir)
     >udd>F01>Student_01>hw_dir                 (output)

!    cr lesson_12.eng                           (create)
```

STORAGE SYSTEM EXAMPLES



STORAGE SYSTEM EXAMPLES

```
!     ls lesson_12.eng                                   (list)
                                                         (output)
      Segments = 1, Lengths = 0.                         (output)
                                                         (output)
      r w     0  lesson_12.eng                           (output)

!     ls -sort name -reverse -dtu  date time used last   (list)
                                                         (output)
      Segments = 2, Lengths = 0.                         (output)
                                                         (output)
      09/13/77  0849.2  r w    0  lesson_12.eng          (output)
      09/13/77  0849.1  r w    0  lesson_1.math          (output)

!     cwd                                                (change_wdir)

!     pwd                                                (print_wdir)
      >udd>F01>Student_01                                (output)


!     dp add.pl1                                         (dprint)
      1 request signalled, 22 already in queue 3         (output)

!     ws -wd "ls -brief"  single names of segments       (walk_subtree)
                                                         (output)
                 >udd>F01>Student_01                     (output)
                                                         (output)
      Segments = 3, Lengths = 1.                         (output)
                                                         (output)
      Prince                                             (output)
      seg_1                                              (output)
      add.pl1                                            (output)
                                                         (output)
                 >udd>F01>Student_01>hw_dir              (output)
                                                         (output)
      Segments = 2, Lengths = 0.                         (output)
                                                         (output)
      lesson_12.eng                                      (output)
      lesson_1.math                                      (output)
```

YOU ARE NOW READY FOR WORKSHOP #3

This page has intentionally
been left blank.

TOPIC VII

THE COMMAND LANGUAGE

This page has intentionally
been left blank.

WHAT IS A COMMAND

- COMMAND PROCEDURE
  - A PROGRAM - USUALLY WRITTEN BY A SYSTEMS PROGRAMMER
  - RESIDES IN ONE OF THE SYSTEM'S LIBRARIES
  - EXECUTED BY TYPING ITS NAME
  - DESIGNED TO
    - PERFORM EXPECTED TASK
    - ACCEPT AN ARBITRARY NUMBER OF ARGUMENTS
    - REPORT TYPING ERRORS
    - HANDLE OTHER USER ERRORS

WHAT IS A COMMAND

- COMMAND

  - THE CHARACTER STRING TYPED AT THE TERMINAL TO INVOKE A SPECIFIC COMMAND PROCEDURE

  - EXAMPLES:

    - THE COMMAND PROCEDURE accept_messages IS INVOKED BY TYPING THE COMMANDS:

      ```
      accept_messages

      am
      ```

    - THE COMMAND PROCEDURE print IS INVOKED BY TYPING THE COMMANDS:

      ```
      print seg_1

      pr treasure_hunt
      ```

WHAT IS A COMMAND

- COMMAND LINE

  - A LINE TYPED AT THE TERMINAL WHEN AT MULTICS COMMAND LEVEL

  - MAY BE ONE COMMAND

    ```
    accept_messages

    am

    print  seg_1
    ```

  - MAY BE A MISTYPED COMMAND

    ```
    primt  seg_1
    ```

  - MAY BE MORE THAN ONE COMMAND SEPARATED BY SEMI-COLONS

    ```
    accept_messages; pr seg_1

    am; print start_up.ec; who
    ```

  - MAY BE NULL (I.E., JUST A LINEFEED) (two semicolons side by side)

  - FREE FORMAT ACCEPTED (EXTRA SPACES ARE IGNORED)

    ```
    list   -all;    sm   LJones.FED    TECO is ready!

      print    add.pl1

    am;pm;who

    am  ;pm;  who  ;;
    ```

COMMAND ARGUMENTS

- ARGUMENTS

  - A SERIES OF CHARACTER STRINGS THAT FOLLOW A COMMAND

- PATHNAME ARGUMENTS

  - THE NAMES OF SEGMENTS OR DIRECTORIES UPON WHICH THE COMMAND IS TO ACT

```
cwd >udd>F01>Student_01

print seg_1
```

- CONTROL ARGUMENTS

  - ARGUMENTS THAT MODIFY THE MANNER IN WHICH THE COMMAND PERFORMS ITS TASK

  - ALWAYS START WITH A - (MINUS)

```
list -segment

list -directory

list -all

list seg_1 -date_time_used
```

COMMAND ARGUMENTS

- ARGUMENTS OF CONTROL ARGUMENTS

```
dprint -copy 2 -ds MS_102 add.pl1
dp -cp 2 -ds "MS 102" add.pl1
set_tty -modes crecho
memo -time 8am WAKE UP!
```

- OTHER ARGUMENTS

```
sm Student_04.F01 Going to lunch?
print seg_2  15  40
login TSmith
```

- NO ARGUMENTS

```
print_wdir
defer_messages
```

- DEFAULT ARGUMENTS (HURRAH FOR DEFAULTS!)

```
change_wdir  {Home Directory}
print seg_2  {first  last}
```

COMMON CONTROL ARGUMENTS

LONG FORM	SHORT FORM	USUAL MEANING
-all	-a	OPERATE ON ALL TYPES OR ALL ENTRIES
-brief	-bf	SHORTEN THE VERBOSITY AND/OR CONTENTS OF RESPONSE
-copy n	-cp n	CREATE n COPIES
-directory	-dr	OPERATE ON DIRECTORIES
-long	-lg	INCREASE THE VERBOSITY AND/OR CONTENTS OF RESPONSE
-optimize	-ot	OPTIMIZE GENERATED CODE (FOR COMPILERS)
-print	-pr	PRINT A STATUS OR SUMMARY REPORT
-queue n	-q n	USE PRIORITY QUEUE n
-segment	-sm	OPERATE ON SEGMENTS, OR SEND OUTPUT TO A SEGMENT
-table	-tb	GENERATE A SYMBOL TABLE (FOR COMPILERS)
-time {dt}	-tm {dt}	DELAY UNTIL THE SPECIFIED TIME, OR GENERATE TIMING STATISTICS
-totals	-tt	PRINT TOTALS

STAR CONVENTION

- STAR CONVENTION

  - A SHORTHAND NOTATION ACCEPTED BY MANY COMMANDS USED TO SPECIFY A GROUP OF SEGMENTS OR DIRECTORIES

  - * MATCHES ANY SINGLE COMPONENT OF AN ENTRYNAME

  - ** MATCHES ANY #in GROUP OF COMPONENTS IN AN ENTRYNAME

  - ? MATCHES ANY CHARACTER IN A COMPONENT OF AN ENTRYNAME

- ASSUME SOME DIRECTORY CONTAINS THE FOLLOWING SEGMENTS:

```
a.fortran          seg_1.pl1
ad.fortran         seg_1.new
add                seg_1.new.cobol
add.pl1            seg_1.old
new                seg_1.old.pl1
new.a.fortran      seg_1.old.test.pl1
```

- EXAMPLES USING THE list COMMAND (THE list HEADER IS NOT SHOWN)

```
list seg_1.pl1 seg_1.new seg_1.old

     rw     1  seg_1.pl1
     rw     2  seg_1.new
     rw     1  seg_1.old
```

STAR CONVENTION

```
list seg_1.*

     rw      1  seg_1.pl1
     rw      2  seg_1.new
     rw      1  seg_1.old
```

```
list seg_1.*.*

     rw      1  seg_1.new.cobol
     rw      2  seg_1.old.pl1
```

```
list new.**
```

includes 'ϕ'

```
     rw      1  new
     rw      1  new.a.fortran
```

```
list *.pl1

     rw      1  add.pl1
     rw      1  seg_1.pl1
```

```
list **.pl1

     rw      1  add.pl1
     rw      1  seg_1.pl1
     rw      2  seg_1.old.pl1
     rw      1  seg_1.old.test.pl1
```

```
list *.*.*.*

     rw      1  seg_1.old.test.pl1
```

```
list a*
```

start with a & 1 compone
any # of characters

```
     rw      1  add
```

STAR CONVENTION

```
list s*.*.pl1

     rw     2  seg_1.old.pl1
```

```
list s*.**.p*

     rw     1  seg_1.pl1
     rw     2  seg_1.old.pl1
     rw     1  seg_1.old.test.pl1
```

```
list ???

     rw     1  add
     rw     1  new
```

```
list a?.*

     rw     1  ad.fortran
```

```
list a?*.*

     rw     1  ad.fortran
     rw     1  add.pl1
```

```
list ad*.**

     rw     1  add
     rw     1  add.pl1
     rw     1  ad.fortran
```

```
list **

     <all segments>
```

SUBSYSTEMS

- SUBSYSTEM

  - A COLLECTION OF PROGRAMS THAT PROVIDE A SPECIAL ENVIRONMENT FOR SOME PARTICULAR PURPOSE

    - EDITING

      edm

      qedx

    - CALCULATION

      calc

    - DEBUGGING

      probe

      debug

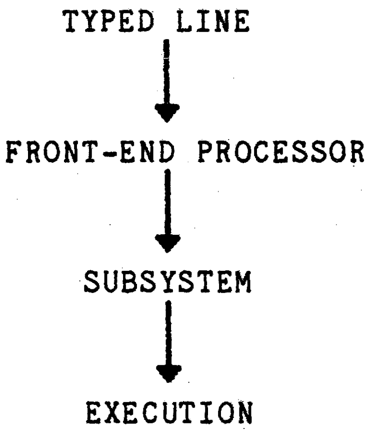
  - A USER ENTERS A SUBSYSTEM BY COMMAND AND EXITS THE SUBSYSTEM BY REQUEST

  - TYPED LINES ARE INTERPRETED BY THE SUBSYSTEM, NOT BY THE MULTICS COMMAND PROCESSOR; THEREFORE, THEY ARE NOT COMMAND LINES

  - THE SUBSYSTEM MAY PERFORM ITS OWN REQUEST PROCESSING, FILE HANDLING, AND ACCOUNTING

SUBSYSTEMS

- REQUEST LINE FLOW



TYPED LINE

FRONT-END PROCESSOR

SUBSYSTEM

EXECUTION

- COMMAND LEVEL

  - THE PROCESS STATE IN WHICH TYPED LINES ARE INTERPRETED BY THE MULTICS COMMAND PROCESSOR

  - TYPED LINES ARE REFERRED TO AS COMMAND LINES

- SUBSYSTEM LEVEL

  - THE PROCESS STATE IN WHICH TYPED LINES ARE INTERPRETED BY THE SUBSYSTEM

  - TYPED LINES ARE REFERRED TO AS REQUEST LINES

This page has intentionally

been left blank.

TOPIC VIII

EXEC_COM BASICS

This page has intentionally been left blank.

WHAT IS AN EXEC COM

- EXEC_COM

  ▯ A SEGMENT THAT CONTAINS A SERIES OF COMMAND LINES

  ▯ MAY BE CREATED USING A TEXT EDITOR

  ▯ SEGMENT NAME MUST HAVE A SUFFIX OF ec

      A.ec

      print.ec

      start_up.ec

  ▯ THE COMMAND LINES ARE EXECUTED SEQUENTIALLY, AS A SET, BY USING THE exec_com COMMAND

EXEC COM MECHANISM

to execute a command segment

- exec_com, ec

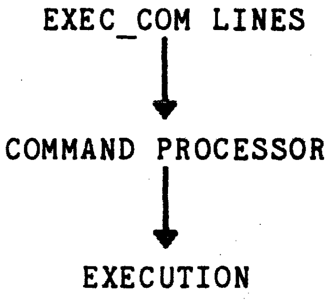
  - EXECUTES THE COMMAND LINES CONTAINED IN AN EXEC_COM SEGMENT

  - COMMAND LINES ARE PRINTED ON THE USER'S TERMINAL AS THEY ARE EXECUTED

  - USAGE:

    ```
    exec_com path

    ec A.ec

    ec print.ec

    ec start_up.ec
    ```

- EXEC_COM LINE FLOW



EXEC_COM LINES

COMMAND PROCESSOR

EXECUTION

EXEC COM MECHANISM

- EXAMPLE: LET THE SEGMENT print.ec CONTAIN THE FOLLOWING TEXT:

  print.ec

```
cwd >udd>F01>Student_07
ls
pr  seg_1
logout
```

  - TYPING THE ONE COMMAND

    ec print.ec

    HAS THE SAME EFFECT AS TYPING THE FOUR COMMANDS ABOVE

START UP EXEC COMS

- START_UP EXEC_COM

  - HAS THE NAME start_up.ec

  - LOCATED IN USER'S HOME DIRECTORY

  - THE COMMAND LINES CONTAINED WITHIN ARE AUTOMATICALLY EXECUTED FOR THE USER AT LOGIN

  - CAN ALSO BE INVOKED MANUALLY

    ```
    ec start_up.ec
    ```

- COMMON USAGE

  - SET-UP USER'S ENVIRONMENT

  - CHECK ON EVENTS SINCE LAST LOGIN

  - PERFORM ANY DESIRED TASKS AT LOGIN

START UP EXEC COMS

- EXAMPLE: LET THE SEGMENT start_up.ec CONTAIN THE FOLLOWING TEXT:

start_up.ec

```
am
pm
```

  - THESE TWO COMMANDS ARE AUTOMATICALLY EXECUTED FOR THE USER AT LOGIN

- NOTE: THE PRESENCE OF A start_up.ec WILL SUPPRESS THE PRINTING OF THE SYSTEM'S MESSAGE OF THE DAY AT LOG IN

START UP EXEC COMS

- print_motd, pmotd

  - PRINTS OUT THE MESSAGE OF THE DAY IF THE USER HASN'T SEEN IT

  - CREATES AND USES A HOME DIRECTORY SEGMENT NAMED person_id.motd

  - THIS COMMAND IS NORMALLY FOUND IN USER'S start_up.ec

  - USAGE:
    ```
    print_motd
    pmotd
    ```

TOPIC IX

THE ABBREV PROCESSOR

This page has intentionally
been left blank.

WHAT IS THE ABBREV PROCESSOR

- ABBREV PROCESSOR
  - A FACILITY ALLOWING USERS TO ABBREVIATE PARTS OF (OR WHOLE) COMMAND LINES
  - A SUBSYSTEM WHICH MUST BE EXPLICITLY INVOKED BY THE USER (ABBREV MODE)
  - FUNCTIONS
    - RESPOND TO ABBREV REQUEST LINES
    - EXPAND ABBREVIATIONS IMBEDDED IN COMMAND LINES

ABBREV MECHANISM

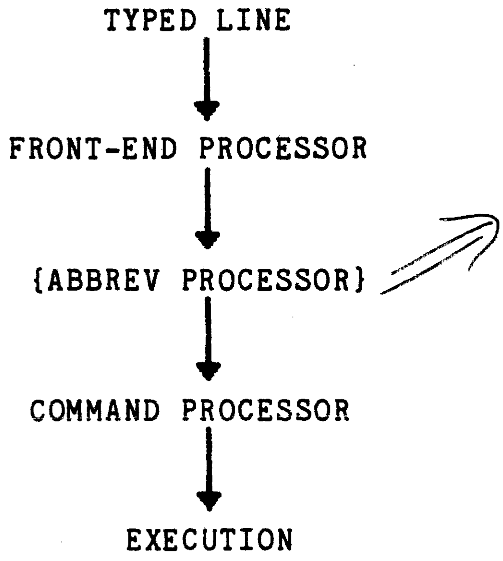
- abbrev, ab — must have

  ▯ PLACES THE USER IN ABBREV MODE. (INVOKES THE ABBREV PROCESSOR)

  ▯ USAGE: abbrev

  ab

- COMMAND LINE FLOW



TYPED LINE

FRONT-END PROCESSOR

{ABBREV PROCESSOR}

COMMAND PROCESSOR

EXECUTION

when implemented

ABBREV MECHANISM

- ABBREV MECHANISM
  - ABBREV PROCESSOR EXAMINES TYPED COMMAND LINES
    - ABBREV REQUEST LINES
      - BEGIN WITH "."
      - DEFINE, DELETE, LIST ABBREVIATIONS
      - CONTROL OTHER ABBREV OPERATIONS
    - COMMAND LINES
      - DO NOT BEGIN WITH "."
      - ANY AND ALL PREVIOUSLY DEFINED ABBREVIATIONS FOUND ARE EXPANDED
      - COMMAND LINE IS THEN PASSED ON TO THE COMMAND PROCESSOR
  - ABBREVIATIONS ARE MAINTAINED IN A HOME DIRECTORY SEGMENT NAMED Person_id.profile

file not in ASCII format so must use .l

ABBREV REQUESTS

- .a

▯ ADD AN ABBREVIATION TO THE USER'S PROFILE SEGMENT

▯ ABBREVIATIONS MUST BE 8 CHARACTERS OR LESS

▯ USAGE:

```
USAGE:   .a abbrev meaning-of-abbreviation
         .a F1 >udd>F01
         .a LARRY LJones.FED
```

- .ab

▯ ADD AN ABBREVIATION WHICH IS VALID ONLY AT THE BEGINNING OF A COMMAND

▯ USAGE:

```
USAGE:   .ab abbrev meaning-of-abbreviation
         .ab dp dprint -he T.Smith -ds Stat_14
         .ab cata list
         .ab GO ec run_it.ec
```

ABBREV REQUESTS

- .d

  - DELETE SPECIFIED ABBREVIATION(S) FROM THE USER'S PROFILE

  - USAGE:
    ```
    .d abbrev1...abbrevn
    .d LARRY
    ```

- .l

  - LIST CURRENTLY DEFINED ABBREVIATIONS AND WHAT THEY STAND FOR

  - USAGE:
    ```
    .l {abbrev1...abbrevn}
    .l
    .l F1 dp
    ```

- .la

  - LIST ABBREVIATIONS THAT BEGIN WITH THE SPECIFIED LETTER(S)

  - USAGE:
    ```
    .la letter1...lettern
    .la m
    .la m d
    ```

ABBREV REQUESTS

- ~~.q~~
  - QUIT USING THE ABBREV PROCESSOR
  - USAGE: .q

- .s
  - EXPAND AND SHOW THE COMMAND LINE WITHOUT EXECUTING IT
  - USAGE:

```
.s text
.s cwd F1; dp add.pl1
```

ABBREV REQUESTS

- EXECUTE THE COMMAND LINE WITHOUT EXPANDING IT

- USAGE 1:
(DON'T EXPAND)

  ```
  . text

  . print A.lunch.cb
  ```

- USAGE 2:
(ACKNOWLEDGE)

  ```
  .
  ```

- ACKNOWLEDGE THAT THE USER IS IN ABBREV MODE (ABBREV RESPONDS WITH "ab")

- OFTEN USED TO AFFIRM COMMAND LEVEL WHEN "MULTING" WITH THE READY MESSAGE OFF

ABBREV EXAMPLES

- LET TSmith's PROFILE SEGMENT CONTAIN THE FOLLOWING ABBREVIATIONS

```
  ft        fortran
  cb        cobol
  LARRY     LJones.FED
  home      >udd>ProjA>TSmith
  F1        >udd>F01
  lunch     sm LJones.FED Lunch time!
b dp        dprint -he TSMITH -ds Stat_14
```

- EXAMPLES OF COMMAND LINE EXPANSION (IF IN ABBREV MODE)

```
print array_dot.ft
print array_dot.fortran


sm LARRY Where's your dims program?
sm LJones.FED Where's your dims program?


cwd F1>Student_01
cwd >udd>F01>Student_01


dp add.pl1
dprint -he TSmith -ds Stat_14 add.pl1


list dp.ft
list dp.fortran


lunch
sm LJones.FED Lunch time!
```

ABBREV EXAMPLES

```
print A.lunch.cb
print A.sm LJones.FED Lunch time!.cobol

. print A.lunch.cb
print A.lunch.cb

. print A.lunch.cobol
print A.lunch.cobol

print clunch.cb
print clunch.cobol

print A_lunch.pl1
print A_lunch.pl1
```

- CANDIDATES FOR EXPANSION MUST BE
  - PART OF COMMAND LINE (MAY BE THE ENTIRE COMMAND LINE)
  - CHARACTER STRING, 8 CHARACTERS OR LESS, THAT ARE
  - BOUNDED BY BREAK CHARACTERS

```
tab        `        <
newline    .        >
space      ;        [
"          |        ]
$          (        {
,          )        }
```

NOTE: _ : AND ? ARE NOT A BREAK CHARACTERS!

ABBREV EXAMPLES

YOU ARE NOW READY FOR WORKSHOP #4

TOPIC X

PROGRAMMING ON MULTICS

This page has intentionally
been left blank.

WHAT IS PROGRAMMING

- PROGRAM
  - A LOGICAL SEQUENCE OF OPERATIONS TO BE PERFORMED BY A COMPUTER
  - SOURCE PROGRAM
    - WRITTEN IN AN ENGLISH-LIKE PROGRAMMING LANGUAGE
    - CREATED BY A USER VIA A TEXT EDITOR
    - KEPT IN A SEGMENT CALLED THE SOURCE SEGMENT
    - CANNOT BE EXECUTED
  - OBJECT PROGRAM
    - WRITTEN IN BINARY MACHINE LANGUAGE
    - CREATED BY A COMPILER (WHICH IS ALSO AN OBJECT PROGRAM) FROM THE SOURCE PROGRAM
    - KEPT IN A SEGMENT CALLED THE OBJECT SEGMENT
    - EXECUTED BY TYPING ITS NAME

WHAT IS PROGRAMMING

- PROGRAMMING LANGUAGES SUPPORTED BY MULTICS(1)

  main language

  - PL/1 (VERSATILE, BLOCK STRUCTURE, DYNAMIC ALLOCATION, RECURSIVE, MANY DATA TYPES)
  - FORTRAN (SCIENTIFIC COMPUTATIONS) '77'
  - COBOL (BUSINESS APPLICATIONS, VERBOSE, WIDELY USED)
  - BASIC (COMPUTATIONS, EASY TO LEARN, LIMITED DATA TYPES)
  - APL (DATA MANIPULATION, CRYPTIC BUT POWERFUL)
  - ALM (MULTICS ASSEMBLER LANGUAGE - YES! WE HAVE AN ASSEMBLER)

- THERE EXISTS A SYSTEM PROGRAM, CALLED A COMPILER, FOR EACH PROGRAMMING LANGUAGE SUPPORTED. COMPILERS ARE DESIGNED TO TRANSLATE A PARTICULAR PROGRAMMING LANGUAGE INTO MACHINE LANGUAGE

(1) RELATED MULTICS COURSES: APL (G11 & F11), BASIC (F127), COBOL-74 (F13), FORTRAN (F14), PL/I (F15, F15C, F15D).

WHAT IS PROGRAMMING

- PROGRAMMING
  - DEVELOPING THE SOURCE PROGRAM IN THE PROGRAMMING LANGUAGE OF CHOICE
  - COMPILING (TRANSLATING) THE SOURCE PROGRAM (SOURCE SEGMENT) INTO AN OBJECT PROGRAM (OBJECT SEGMENT)
  - EXECUTING THE OBJECT PROGRAM USING TEST DATA
  - DEBUGGING THE SOURCE PROGRAM TO CORRECT ALL OBSERVED PROBLEMS
  - MAKING THE OBJECT PROGRAM AVAILABLE
    - SYSTEM PROGRAMS MUST BE "INSTALLED" IN ONE OF THE SYSTEM LIBRARIES
    - USER PROGRAMS NEED NO FURTHER ACTION EXCEPT THE SETTING OF ACCESS TO ALLOW USE BY OTHER USERS

DEVELOPING A SOURCE PROGRAM

- DESIGNING (OUTLINING) A SOURCE PROGRAM
  - ▯ TOP-DOWN DESIGN
  - ▯ MODULAR DESIGN
  - ▯ FLOWCHARTING

- DOCUMENTING THE SOURCE PROGRAM
  - ▯ EXTERNAL DOCUMENTATION (PURPOSE, GENERAL DESIGN, HOW TO USE)
  - ▯ INTERNAL DOCUMENTATION (STEP BY STEP DESCRIPTION OF THE PROGRAM)
  - ▯ "SELF-DOCUMENTING" LANGUAGES

- WRITING (CODING) THE SOURCE PROGRAM (USUALLY ON PAPER) IN SOME PROGRAMMING LANGUAGE
  - ▯ "GO-TO-LESS" PROGRAMMING
  - ▯ MNEMONIC VARIABLE NAMES

DEVELOPING A SOURCE PROGRAM

- ERROR DETECTION AND HANDLING

- INPUTTING THE SOURCE PROGRAM, VIA A TEXT EDITOR, TO A SOURCE SEGMENT

  - A SOURCE SEGMENT MAY BE GIVEN ANY DESIRED NAME, HOWEVER, THE SUFFIX MUST BE THE ENTRYNAME OF THE PROGRAMMING LANGUAGE USED

```
add.pl1                ran_num_gen.basic

A_alpha.cobol          page_fault.alm

array_dot.fortran
```

- OPTIONALLY FORMATTING THE SOURCE PROGRAM (COSMETICS)

  - SEVERAL COMMANDS EXIST FOR THE PURPOSE OF FORMATTING SOURCE PROGRAMS

  - DONE TO IMPROVE THE READABILITY OF A SOURCE PROGRAM

  - THE COMMANDS DETECT AND REPORT CERTAIN TYPES OF SYNTAX ERRORS, AND ARE OFTEN USED AS A PRE-COMPILE EXAMINATION

DEVELOPING A SOURCE PROGRAM

- indent, ind

  - IMPROVES THE READABILITY OF A PL/1 SOURCE SEGMENT

  - USAGE: indent path1 {path2} {control_arg}

    ind add.pl1 add.ind.pl1

    ind add.pl1

    ind >udd>F01>Student_09>add.pl1 -indent 3

- format_cobol_source, fcs

  - CONVERTS FREE-FORM COBOL SOURCE PROGRAMS TO FIXED-FORMAT

  - USAGE: format_cobol_source path1 path2

    fcs A_alpha.cobol A_alpha.fcs.cobol

COMPILING A SOURCE PROGRAM

- COMPILER:

  - A SYSTEM PROGRAM DESIGNED TO TRANSLATE A PARTICULAR PROGRAMMING LANGUAGE (ENGLISH-LIKE) INTO MACHINE LANGUAGE (BINARY)

- COMPILE COMMANDS (CREATE OBJECT PROGRAM AND OBJECT SEGMENT!)

  - USAGE:

```
USAGE:     language_name path {-control_args}

           pl1 add.pl1

           pl1 >udd>F01>Student_09>add.pl1

           cobol A_alpha.cobol

           fortran array_dot.fortran

           basic ran_num_gen.basic

           alm page_fault.alm
```

- OBJECT PROGRAMS

  - ALL OBJECT PROGRAMS PRODUCED BY MULTICS COMPILERS ARE:

    - PURE (DO NOT MODIFY THEIR OWN CODE)

    - RE-ENTRANT (MORE THAN ONE USER MAY EXECUTE THE SAME CODE) at same time.

    - RECURSIVE (A PROGRAM CAN CALL ITSELF)

    - IN STANDARD FORMAT (OBJECTS GENERATED FROM DIFFERENT LANGUAGES MAY CALL EACH OTHER EASILY)

COMPILING A SOURCE PROGRAM

- THE OBJECT PROGRAM IS PLACED IN A SEGMENT (CALLED THE OBJECT SEGMENT) IN THE USER'S WORKING DIRECTORY

- THE OBJECT SEGMENT IS GIVEN THE CORRESPONDING SOURCE SEGMENT'S ENTRYNAME WITH THE SUFFIX REMOVED

    add

    A_alpha

    array_dot

    ran_num_gen

    page_fault

● COMPILER LISTINGS

- SOME CONTROL ARGUMENTS WILL CAUSE A COMPILER TO PRODUCE A COMPILER LISTING OPTIONALLY CONSISTING OF A LINE-NUMBERED SOURCE LISTING, A SYMBOL TABLE, AN OBJECT CODE MAP, ERROR MESSAGES, ETC.

- THE COMPILER LISTING IS PLACED IN A SEGMENT (CALLED THE LIST SEGMENT) IN THE USER'S WORKING DIRECTORY

- THE LIST SEGMENT IS GIVEN THE CORRESPONDING SOURCE SEGMENT'S NAME WITH THE ORIGINAL SUFFIX REPLACED BY THE SUFFIX list

    add.list

    A_alpha.list

    array_dot.list

    ran_num_gen.list

    page_fault.list

COMPILING A SOURCE PROGRAM

- DIAGNOSTICS

  - COMPILERS WILL COMPLAIN ABOUT:

    - SYNTAX ERRORS

    - MISSPELLINGS

    - UNDEFINED REFERENCES

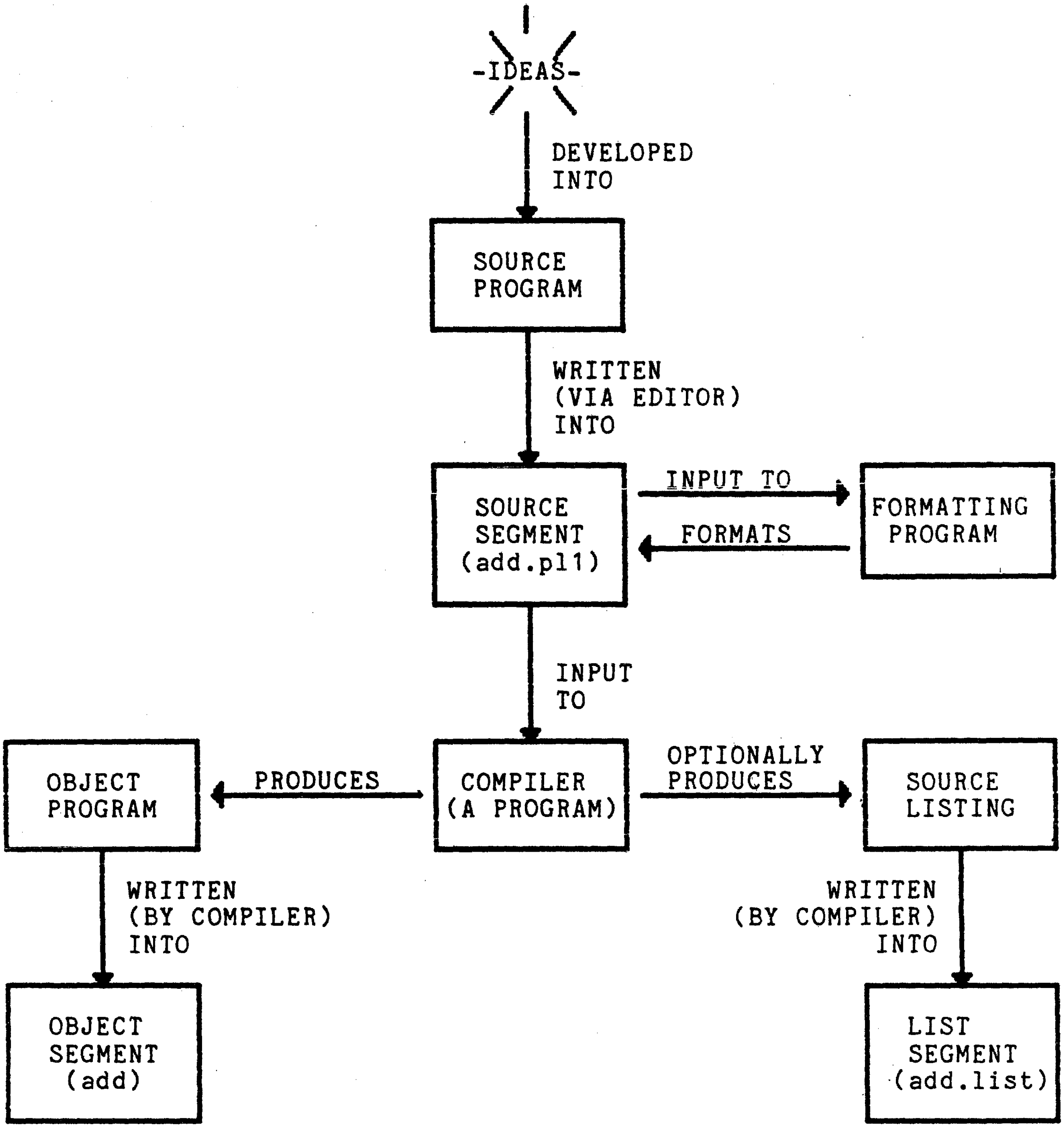
  - ERROR MESSAGES ARE PRINTED AT THE USER'S TERMINAL

  - SEVERE ERRORS WILL SUPPRESS THE FORMATION OF THE OBJECT PROGRAM AND OBJECT SEGMENT

  - THE FORMAT OF ERROR MESSAGES IS COMPILER-DEPENDENT. THE FOLLOWING IS A PL/I ERROR MESSAGE:

```
ERROR 158, SEVERITY 2 ON LINE 30
A constant immediately follows the identifier "zilch"
SOURCE:  a = zilch 4;
```

COMPILING A SOURCE PROGRAM



ENTRYNAMES AND ENTRY POINT NAMES

- ENTRYNAME (

  - A NAME GIVEN TO AN ITEM CONTAINED IN A DIRECTORY (segment program resides on)

- ENTRY POINT NAME

  - THE NAME ASSOCIATED WITH AN ENTRY POINT IN AN OBJECT SEGMENT

- UNLESS OTHERWISE SPECIFIED, MULTICS ASSUMES THE ENTRY POINT NAME IS THE SAME AS THE ENTRYNAME

main program

(MAIN_)

SUBROUTINE (SUB

entry sub1

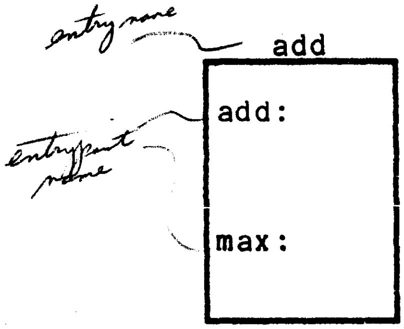
entry point only in subroutines - done in main program.

ENTRYNAMES AND ENTRY POINT NAMES

$ (DOLLAR SIGN)

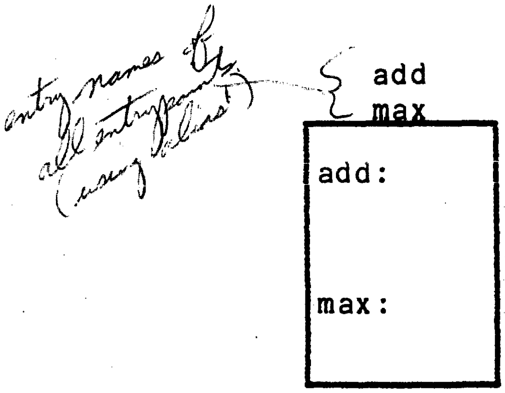
- SEPARATES THE ENTRYNAME OF AN OBJECT SEGMENT FROM THE ENTRY POINT NAME WITHIN THE OBJECT SEGMENT

- EXAMPLE USING AN OBJECT SEGMENT add HAVING ENTRY POINTS add AND max



INVOKED AS	INTERPRETED AS	ENTRY POINT
add	add$add	add
add$max	add$max	max
max	max$max	?

- EXAMPLE AS ABOVE WITH ALIAS NAME max ADDED TO THE SEGMENT



INVOKED AS	INTERPRETED AS	ENTRY POINT
add	add$add	add
add$max	add$max	max
max	max$max	max

EXECUTING AN OBJECT PROGRAM

- AN OBJECT PROGRAM (OBJECT SEGMENT) IS EXECUTED BY TYPING ITS NAME AT A TERMINAL

```
          add                         ran_num_gen

          A_alpha                     page_fault

          array_dot                   >udd>F01>Student_09>add
```

- POSSIBLE RESULTS OF EXECUTING A PROGRAM

  - PROGRAM RUNS TO NORMAL TERMINATION & USER RECEIVES READY MESSAGE

  - PROGRAM PAUSES FOR INPUT FROM THE USER'S TERMINAL

  - PROGRAM HALTS BECAUSE OF A USER-IMPLANTED BREAKPOINT (A DEBUGGING TOOL)

  - PROGRAM HALTS BECAUSE OF A FATAL EXECUTION ERROR

    - OVERFLOW UNDERFLOW, DATA CONVERSION ERROR, UNDEFINED REFERENCE

    - HALTS (INTERRUPTS) THE EXECUTION OF A PROGRAM OR COMMAND. (FROZEN IN MID-AIR)

    - RESPONDS WITH A READY MESSAGE CONTAINING A level CLAUSE

```
r 1038.5 0.185 0.012 27 level 2,11
```

ready message

how many programs are in stack

EXECUTING AN OBJECT PROGRAM

Break key

- PROGRAM ~~HALTS BECAUSE USER~~ ISSUED A QUIT SIGNAL

  - TERMINAL KEY LABELED ATTN, BRK, INTRPT, INTERRUPT,...

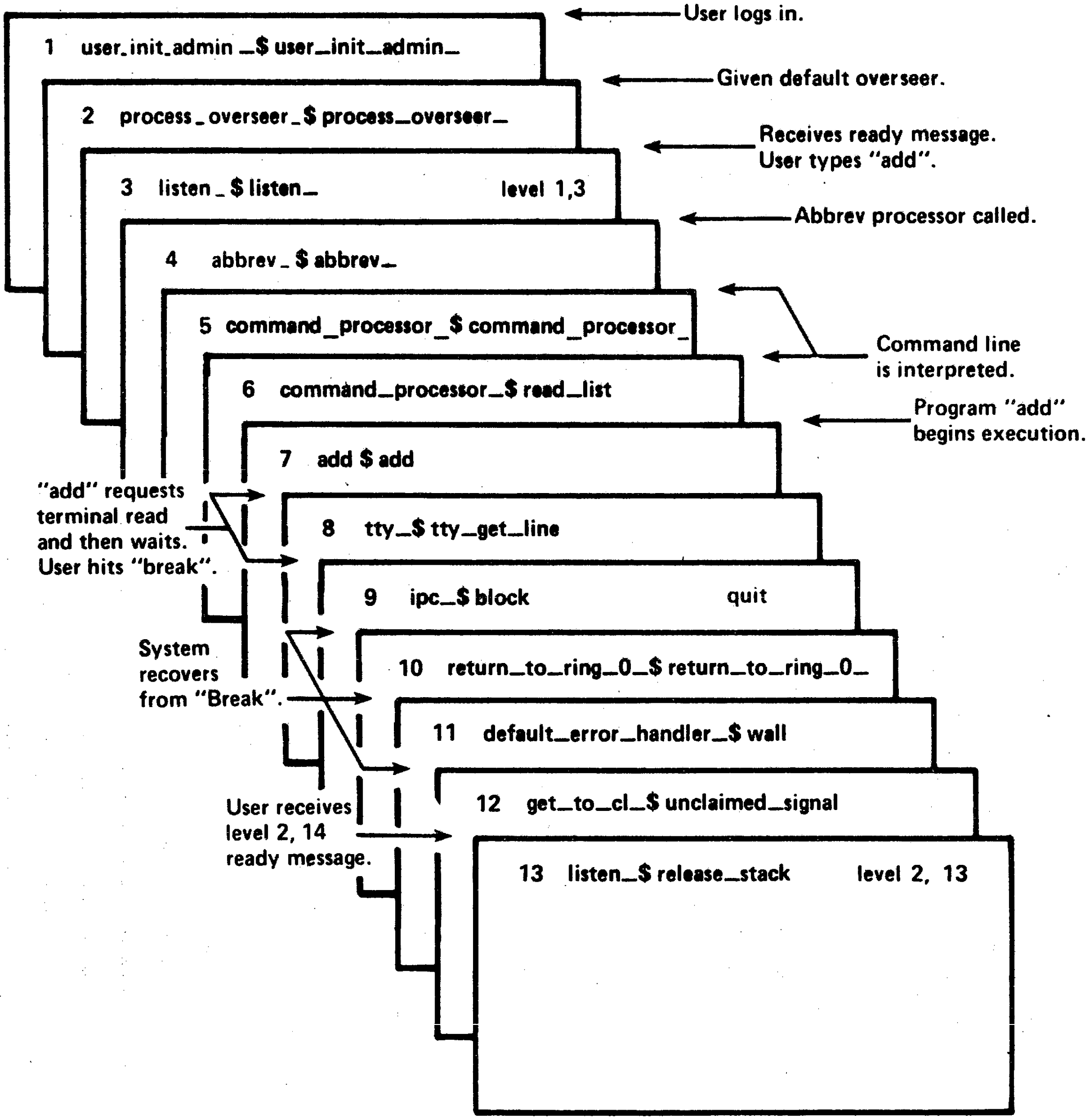
  - HALTS (INTERRUPTS) THE EXECUTION OF A PROGRAM OR COMMAND. (FROZEN IN MID-AIR)

  - RESPONDS WITH A READY MESSAGE CONTAINING A level CLAUSE

```
r 1038.5 0.185 0.012 27 level 2,11
```

EXECUTING AN OBJECT PROGRAM

- THE USER'S STACK (A HISTORY OF CURRENT EVENTS)



EXECUTING AN OBJECT PROGRAM

- POSSIBLE USER ACTIONS AFTER A QUIT SIGNAL OR FATAL ERROR

  - IGNORE THE OLD LEVEL(S) AND PROCEED (EXPENSIVE)

  - OBSERVE WHAT WAS HAPPENING USING DEBUGGING TOOLS

    ```
    value k

    value qty_on_hand
    ```

  - CHANGE VALUES OF VARIABLES USING DEBUGGING TOOLS

    ```
    let k=4

    let qty_on_hand = 0
    ```

  - start, sr

    - RESTARTS THE PROGRAM OF THE IMMEDIATELY PREVIOUS LEVEL AT THE INTERRUPT POINT

    - USAGE:

      ```
      start
      sr
      ```

    - EXAMPLE: WHILE IN qedx, THE USER SIGNALS QUIT WHILE DOING A 1,$p. AFTER RECEIVING A READY MESSAGE THE USER TYPES start, AND THE 1,$p CONTINUES

EXECUTING AN OBJECT PROGRAM

- program_interrupt, pi

  - RESTARTS THE PROGRAM AT A PROGRAMMER DETERMINED POINT

  - PROGRAM MUST HAVE BEEN WRITTEN WITH A program_interrupt HANDLER

  - USAGE: program_interrupt

    pi

  - EXAMPLE: WHILE IN qedx, THE USER SIGNALS QUIT WHILE DOING A 1,$p. AFTER RECEIVING A READY MESSAGE THE USER TYPES pi, AND FINDS HIMSELF BACK IN THE EDITOR AT REQUEST LEVEL

- release, rl

  - RELEASES THE IMMEDIATELY PREVIOUS LEVEL(S) (ONE OR ALL)

  - USAGE: release {-control_arg}

    rl

    rl -all

DEBUGGING TOOLS

- DEBUGGING TOOLS

  - SUBSYSTEMS WHICH ARE USEFUL IN LOCATING, EXAMINING, AND CORRECTING UNEXPECTED OCCURRENCES OR CONDITIONS WITHIN OBJECT PROGRAMS

  - SUCH TOOLS MINIMIZE THE NEED FOR MEMORY DUMPS

- debug, db

  - INVOKES AN INTERACTIVE DEBUGGING AID

  - HARDWARE LEVEL, SYMBOLIC, CRYPTIC

  - PROCEDURES MAY HAVE ORIGINATED FROM ANY LANGUAGE. (SYMBOLIC CAPABILITY ONLY AVAILABLE FOR PL/I AND FORTRAN)

  - IN ORDER TO UTILIZE SYMBOLIC CAPABILITIES, THE SOURCE PROGRAM MUST HAVE BEEN COMPILED WITH THE -table CONTROL ARGUMENT

  - USES A HOME DIRECTORY SEGMENT NAMED Person_id.breaks

  - USAGE:
    ```
    debug

    db
    ```

DEBUGGING TOOLS

- probe, pb

  - INVOKES AN INTERACTIVE DEBUGGING AID

  - SYMBOLIC, SOURCE LEVEL

  - PROCEDURES MUST HAVE ORIGINATED FROM PL/I, FORTRAN, OR COBOL SOURCE

  - IN ORDER TO UTILIZE SYMBOLIC CAPABILITIES, THE SOURCE PROGRAM MUST HAVE BEEN COMPILED WITH THE -table CONTROL ARGUMENT

  - USAGE 1: probe path
(SETTING)

  - USAGE 2: probe
(EXAMINING)

  - A THOROUGH DISCUSSION OF THE probe COMMAND MAY BE FOUND IN THE COMMANDS AND ACTIVE FUNCTIONS MANUAL AG92

DEBUGGING TOOLS

DEBUG CAPABILITIES	PROBE CAPABILITIES
EXAMINE DATA, SOURCE AND OBJECT	EXAMINE DATA AND SOURCE
MODIFY DATA AND OBJECT	MODIFY DATA
EXECUTE COMMANDS	EXECUTE COMMANDS
CONDITIONAL BREAK POINTS	CONDITIONAL BREAK POINTS AND/OR OPERATIONS
EXAMINE MACHINE CONDITIONS	
EXAMINE REGISTERS	
DUMP DATA IN VARIOUS FORMATS	

DYNAMIC SEARCHING

- RECALL:
  - COMMAND PROCEDURE
    - A PROGRAM INVOKED BY TYPING ITS NAME
  - OBJECT PROGRAM
    - EXECUTED BY TYPING ITS NAME

- COMMAND LINE INTERPRETATION
  - THE FIRST WORD OF EVERY COMMAND IS ASSUMED TO BE THE NAME OF AN OBJECT PROGRAM

```
list -all

ls

send_message Student_05.F01 Where are you?

add
```

  - MULTICS SEARCHES THROUGH VARIOUS (PREDETERMINED) DIRECTORIES TO FIND AN OBJECT SEGMENT HAVING THE SPECIFIED NAME
  - IF THE SEARCHING IS SUCCESSFUL, LINKING OCCURS ("THE LINK IS SNAPPED") AND EXECUTION COMMENCES

DYNAMIC SEARCHING

- ALL COMMANDS (SYSTEM PROGRAMS), USER-WRITTEN PROGRAMS, AND SUBROUTINE MUST BE "FOUND" BEFORE THEY CAN BE EXECUTED

- SEARCH RULES (WHERE, O'WHERE) Oh Brother!

    - A LIST SPECIFYING THE NAMES AND THE ORDER OF DIRECTORIES TO BE SEARCHED

    - SEARCH RULES ONLY HAVE SIGNIFICANCE FOR THE EXECUTION OF OBJECT PROGRAMS. COMMANDS SUCH AS list AND print DO NOT SEARCH FOR THEIR "TARGETS"

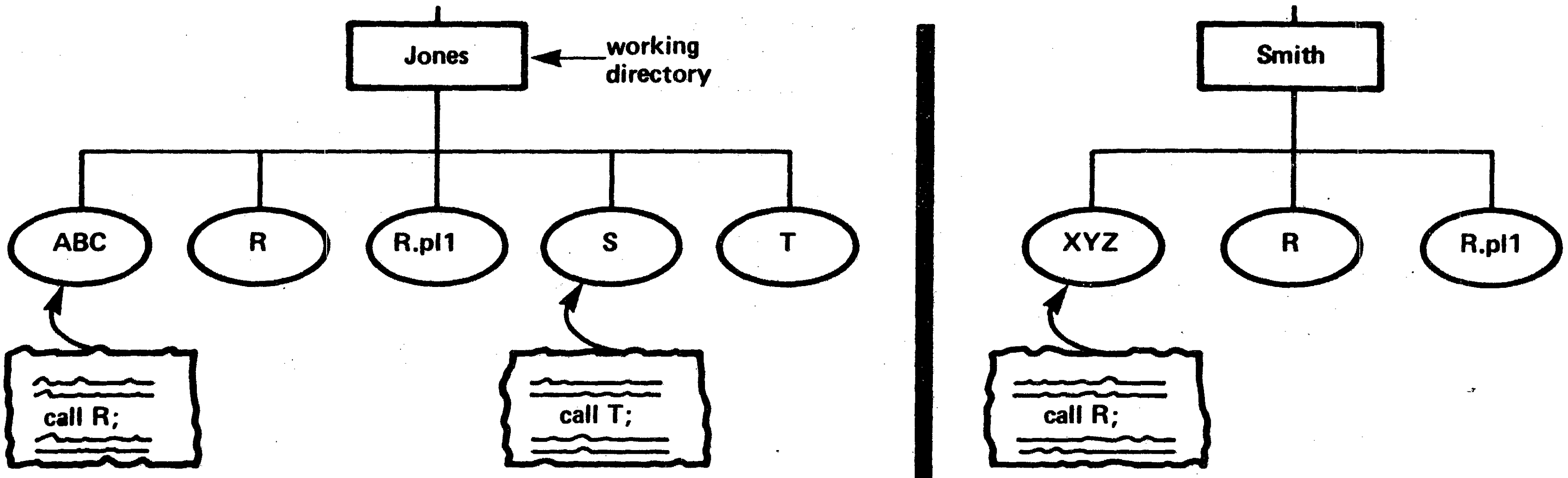
    - DEFAULT SEARCH RULES

called for first time

all under root

```
initiated segments
referencing directory
working directory
>system_library_standard
>system_library_unbundled
>system_library_1
>system_library_tools
>system_library_auth_maint
```

SEARCH RULES EXAMPLE FOR USER JONES



Jones' TERMINAL INPUT

COMMAND LINE TYPED	SEGMENTS FOUND
login Jones password	{ login RELATED SEGMENTS } ≈ 250
pwd	> sss > pwd
S	> udd > . . . > Jones > S > udd > . . . > Jones > T
> udd > . . . > Smith > XYZ	> udd > . . . > Smith > XYZ > udd > . . . > Smith > R
XYZ	> udd > . . . > Smith > XYZ > udd > . . . > Smith > R
ABC	> udd > . . . > Jones > ABC > udd > . . . > Smith > R
pr > udd > . . . > Smith > R.pl1	> sss > pr
pr R.pl1	> sss > pr

Jones' INITIATED SEGMENT LIST

REFERENCE NAME	SEGMENT INITIATED
{ ⋮ }	{ login RELATED SEGMENTS }
pwd	> sss > pwd
S	> udd > . . . > Jones > S
T	> udd > . . . > Jones > T
XYZ	> udd > . . . > Smith > XYZ
R	> udd > . . . > Smith > R
ABC	> udd > . . . > Jones > ABC
pr	> sss > pr

DYNAMIC SEARCHING

- print_search_rules, psr

used to find out what priority search rules are used.

□ PRINTS THE USER'S CURRENT SEARCH RULES

□ USAGE:

```
print_search_rules
psr
```

- add_search_rules, asr

□ ADDS A DIRECTORY TO THE USER'S SEARCH RULES

□ USAGE:

```
add_search_rules path1 {-control_arg path2}
asr >udd>F01>Student_01>tools -after working_dir
```

```
initiated segments
referencing directory
working directory
>udd>F01>Student_01>tools
>system_library_standard
>system_library_unbundled
>system_library_1
>system_library_tools
>system_library_auth_maint
```

DYNAMIC SEARCHING

- delete search rules, dsr (opposite asr)

  - DELETES ONE OR MORE DIRECTORIES FROM THE USER'S SEARCH RULES

  - USAGE:

    ```
    delete_search_rules paths

    dsr >udd>F01>Student_06>tools
    ```

- initiate, in

  - ENABLES USERS TO INITIATE (MAKE KNOWN) SEGMENTS DIRECTLY

  - THE SEGMENTS REFERENCE NAME AND ITS ABSOLUTE PATHNAME ARE PLACED IN THE USER'S LIST OF INITIATED SEGMENTS

  - USAGE:

    ```
    initiate path {ref_names} {-control_args}

    in >udd>FED>Kerr>tools>editor

    in >udd>FED>Kerr>tools>editor qx
    ```

DYNAMIC SEARCHING

- list_ref_names, lrn (Be at least 250 after login)

  - LISTS THE REFERENCE NAME, PATHNAME AND SEGMENT NUMBER OF SEGMENTS KNOWN TO THE USER'S PROCESS (I.E. INITIATED SEGMENTS)

  - USAGE:

    ```
    list_ref_names {paths}{-control_args)

    lrn

    lrn >udd>F01>Student_07>add
    ```

- terminate_ref_name, tmr

  - ALLOWS THE USER TO REMOVE A SEGMENT FROM THE LIST OF SEGMENTS KNOWN TO HIS PROCESS (I.E. INITIATED SEGMENTS)

  - USAGE:

    ```
    terminate_ref_name ref_names

    tmr add who
    ```

DYNAMIC SEARCHING

- where, wh (... HOW I WONDER WHERE YOU ARE!)

  - USES CURRENT SEARCH RULES TO LOCATE AND PRINT THE ABSOLUTE PATHNAME OF A SEGMENT

  - ONLY THE PRIMARY NAME OF THE LOCATED SEGMENT IS PRINTED

  - MAY BE USED TO CHECK IF A NAME IS "SAFE" TO USE FOR A SEGMENT

  - USAGE:

    ```
    where ref_name {-control_arg}

    wh qx

    wh sort

    wh wh -all

    wh >udd>F01>Student_06>add.pl1
    ```

- NOTE THAT IN THE SIMPLEST CASE (WHERE A USER LOGS IN, CREATES AND EXECUTES PROGRAMS IN HIS HOME DIRECTORY), NO KNOWLEDGE OF SEARCH RULES OR INITIATED SEGMENTS IS REQUIRED

YOU ARE NOW READY FOR WORKSHOP #5

This page has intentionally

been left blank.

TOPIC XI

ACCESS CONTROL

This page has intentionally
been left blank.

WHAT IS ACCESS CONTROL

- ACCESS CONTROL

  - A FACILITY FOR CONTROLLING (IN A SELECTIVE MANNER):

    - ACCESS TO THE CONTENTS OF SEGMENTS

    - ACCESS TO THE ATTRIBUTES OF SEGMENTS

    - ABILITY TO CREATE SEGMENTS

    - ABILITY TO DELETE SEGMENTS

  - A FACILITY ALLOWING USERS TO SELECTIVELY SHARE THEIR PROGRAMS AND DATA WITH OTHER USERS

  - USER MUST EXPLICITLY GRANT (SET) ACCESS IF SHARING IS DESIRED

    - set_acl, delete_acl & list_acl COMMANDS

  - SELECTIVE SHARING

    - BY Person_id

    - BY Project_id

    - BY ACCESS MODE (READ, WRITE ...)

ACCESS CONTROL LIST (ACL)

- ACCESS CONTROL LIST (ACL)

  - EVERY SEGMENT AND DIRECTORY HAS ITS OWN ACCESS CONTROL LIST (ACL)

  - AN ACL IS A LIST OF User_ids-LIKE ENTRIES CALLED ACCESS IDENTIFIERS, AND ASSOCIATED ACCESS MODES

```
r    TSmith.ProjA.*
rw   Student_04.F01.*
r    *.FED.*
```

  - IN ORDER FOR A USER TO ACCESS A SEGMENT (OR DIRECTORY):

    - THE USER'S User_id MUST "MATCH" AN ENTRY ON THE ACL

    - FURTHERMORE, THE USER IS RESTRICTED TO THE ACCESS MODE(S) SPECIFIED BY THAT PARTICULAR ACL ENTRY

  - BY DEFAULT, USERS ARE GIVEN COMPLETE ACCESS TO THE SEGMENTS AND DIRECTORIES THEY CREATE

  - USERS MAY ADD AND DELETE ENTRIES FROM THE ACL'S OF THEIR SEGMENTS AND DIRECTORIES VIA THE set_acl AND delete_acl COMMANDS

ACCESS CONTROL LIST (ACL)

- AN ACL IS CONSIDERED AN ATTRIBUTE OF A SEGMENT OR DIRECTORY. IN ORDER TO set_acl OR delete_acl, THE USER MUST HAVE THE APPROPRIATE PERMISSION TO DO SO

- ACCESS VIOLATIONS ARE TRAPPED BY THE SYSTEM AND THE VIOLATOR IS INFORMED

- ACCESS CHANGES OCCUR INSTANTANEOUSLY SINCE ACCESS RIGHTS ARE CHECKED BY HARDWARE WITH EVERY ACCESS

ACCESS CONTROL LIST (ACL)

- ACCESS MODES FOR SEGMENTS (rew n)

Can have any comb of these
r, e
r, w
r, e, w

- ▯ READ (r)
  - ▯ CONTENTS OF THE SEGMENT CAN BE READ BY THE DESIGNATED USER(S)
  - ▯ print, copy, move, qedx's "r" request

- ▯ EXECUTE (e) does not mean can read
  - ▯ CONTENTS OF THE SEGMENT CAN BE EXECUTED BY THE DESIGNATED USER(S). (MEANINGFUL ONLY FOR OBJECT SEGMENTS)
  - ▯ DEPENDING ON THE OBJECT PROGRAM, READ MAY ALSO BE REQUIRED FOR EXECUTION
  - ▯ add, >udd>F01>Student_01>add

- ▯ WRITE (w)
  - ▯ CONTENTS OF THE SEGMENT CAN BE MODIFIED (OVER WRITTEN) BY THE DESIGNATED USER(S)
  - ▯ qedx's "w" request

must be used alone

- ▯ NULL (n) - default access code
  - ▯ ALL ACCESS TO THE CONTENTS OF THE SEGMENT IS EXPLICITLY DENIED FOR THE DESIGNATED USER(S)

ACCESS CONTROL LIST (ACL)

- ACCESS MODES FOR DIRECTORIES (sma n)---DO NOT INFLUENCE ACCESS ON INFERIOR DIRECTORIES

ed in any
be

- STATUS (s)
  - ATTRIBUTES OF EXISTING ENTRIES IN THE DIRECTORY CAN BE OBTAINED BY THE DESIGNATED USER(S)
  - status, list
- MODIFY (m) cannot add or append
  - ATTRIBUTES OF EXISTING ENTRIES IN THE DIRECTORY CAN BE MODIFIED BY THE DESIGNATED USER(S).
  - ENTRIES CAN ALSO BE DELETED BY THE DESIGNATED USER(S)
  - add_name, rename, delete, set_acl, delete_acl
- APPEND (a)
  - NEW SEGMENTS, DIRECTORIES, AND LINKS CAN BE CREATED IN (OR MOVED TO) THE DIRECTORY BY THE DESIGNATED USER(S)
  - create, create_dir, link, copy, move

must be
used alone

- NULL (n)
  - ALL ACCESS TO THE ATTRIBUTES OF EXISTING ENTRIES IN THE DIRECTORY IS EXPLICITLY DENIED FOR THE DESIGNATED USER(S) -AND- THE CREATION OF ENTRIES IN THE DIRECTORY IS EXPLICITLY DENIED FOR THE DESIGNATED USER(S)

ACCESS CONTROL LIST (ACL)

- ACCESS IDENTIFIERS

  - THREE-COMPONENT CHARACTER STRING

    Person_id.Project_id.tag

  - TAG IDENTIFIES THE TYPE OF PROCESS

    - a - AN INTERACTIVE PROCESS (A REAL USER)

    - m - AN ABSENTEE PROCESS (AN "ABSENT" USER)

    - z - A SYSTEM PROCESS (A DAEMON) LOGGED IN BY THE OPERATOR

  - MULTICS ASSIGNS EVERY USER AN ACCESS IDENTIFIER AT LOG IN

  - WHEN USED IN AN ACCESS CONTROL LIST, ACCESS IDENTIFIERS ARE OFTEN CALLED "ACL ENTRIES"

  - IF A COMPONENT OF AN ACL ENTRY IS A STAR (ASTERISK), THE STAR IS INTERPRETED AS MATCHING ANY Person_id, ANY Project_id, OR ANY tag, DEPENDING ON ITS POSITION IN THE ACCESS IDENTIFIER

*.F01.a	MATCHES ALL INTERACTIVE F01 USERS
Student_09.*.*	MATCHES Student_09 REGARDLESS OF HOW HE LOGS IN
..m	MATCHES ALL ABSENTEE USERS
..*	MATCHES EVERYONE

ACCESS CONTROL LIST (ACL)

- ENTRIES IN A SEGMENT'S ACL ARE AUTOMATICALLY ORDERED - MOST SPECIFIC IDENTIFIERS FIRST

    Frommer.F01.a

    Frommer.F01.*

    Frommer.*.a

    Frommer.*.*

    *.F01.a

    .F01.

    ..a

    ..*

- MATCHING User_id WITH ENTRIES ON AN ACL

    - PROCEEDS FROM TOP TO BOTTOM

    - FIRST MATCH DETERMINES ACCESS MODE(S)

    - NO MATCH IMPLIES NO ACCESS

ACCESS CONTROL LIST (ACL)

- EXAMPLE: LET seg_1 HAVE THE FOLLOWING ACCESS CONTROL LIST

seg_1's ACL

```
rew      LJones.FED.a
rw       Student_07.F01.*
r        TSmith.ProjA.*
rew      White.*.*
rw       *.ProjA.*
n        *.FED.*
rw       *.SysDaemon.*
n        *.*.m
r        *.*.*
```

USER	USER'S ACCESS
LJones.FED.a	rew
LJones.FED.m	n
TSmith.ProjA.a	r
TSmith.FED.a	n
Green.ProjA.a	---
White.ProjA.a	---
White.FED.a	---
LJones.F01.m	---
LJones.FED.m	---
Kerr.MAC.a	---

ACCESS MANIPULATION COMMANDS

- set_acl, sa

to change permission use same access code with whatever modes you now want this code to have.

□ MANIPULATES THE ACL'S OF SEGMENTS AND DIRECTORIES

permissions

□ USAGE:
```
set_acl path mode1 User_id1... moden User_idn
sa add.pl1 rw Student_04.F01.*
sa add.pl1 r *.F01.* rw *.FED.*
sa *.pl1 r LJones.*.*
sa dir_A sma LJones.FED.*
sa ** r *.*.*
```

effective immediately

- delete_acl, da

□ REMOVES ENTRIES FROM ACL'S OF SEGMENTS AND DIRECTORIES

□ USAGE:
```
delete_acl {path {User_ids}}
da add.pl1 *.F01.*
da add.pl1 Student_04.F01.* *.FED.*
da dir_A LJones.FED.*
```

- list_acl, la

  ▯ LISTS THE ACL'S OF SEGMENTS AND DIRECTORIES

  ▯ IF path OMITTED THEN NO User_id POSSIBLE

  ▯ USAGE:

```
USAGE:  list_acl {path {User_ids}}
        la add.pl1
        la add.pl1 LJones.FED.*
        la
```

DEFAULT AND INITIAL ACL ENTRIES

- WHEN SEGMENTS AND DIRECTORIES ARE CREATED, AN ACL IS AUTOMATICALLY PROVIDED BY MULTICS CONTAINING DEFAULT ACL ENTRIES

  - FOR MOST SEGMENTS:

    ```
    rw      Person_id.Project_id.*
    rw      *.SysDaemon.*
    ```

  - FOR DIRECTORIES:

    ```
    sma     Person_id.Project_id.*
    sma     *.SysDaemon.*
    ```

- NORMAL ACCESS GIVEN TO SYSTEM DIRECTORIES:

  - A USER IS GIVEN "sma" ON HIS HOME DIRECTORY

  - A USER IS GIVEN "s" ON HIS PROJECT DIRECTORY

  - A PROJECT ADMINISTRATOR IS GIVEN "sma" ON THE PROJECT DIRECTORY

  - THE SYSTEM'S ADMINISTRATOR HAS "sma" ON >udd

DEFAULT AND INITIAL ACL ENTRIES

- INITIAL ACCESS CONTROL LIST

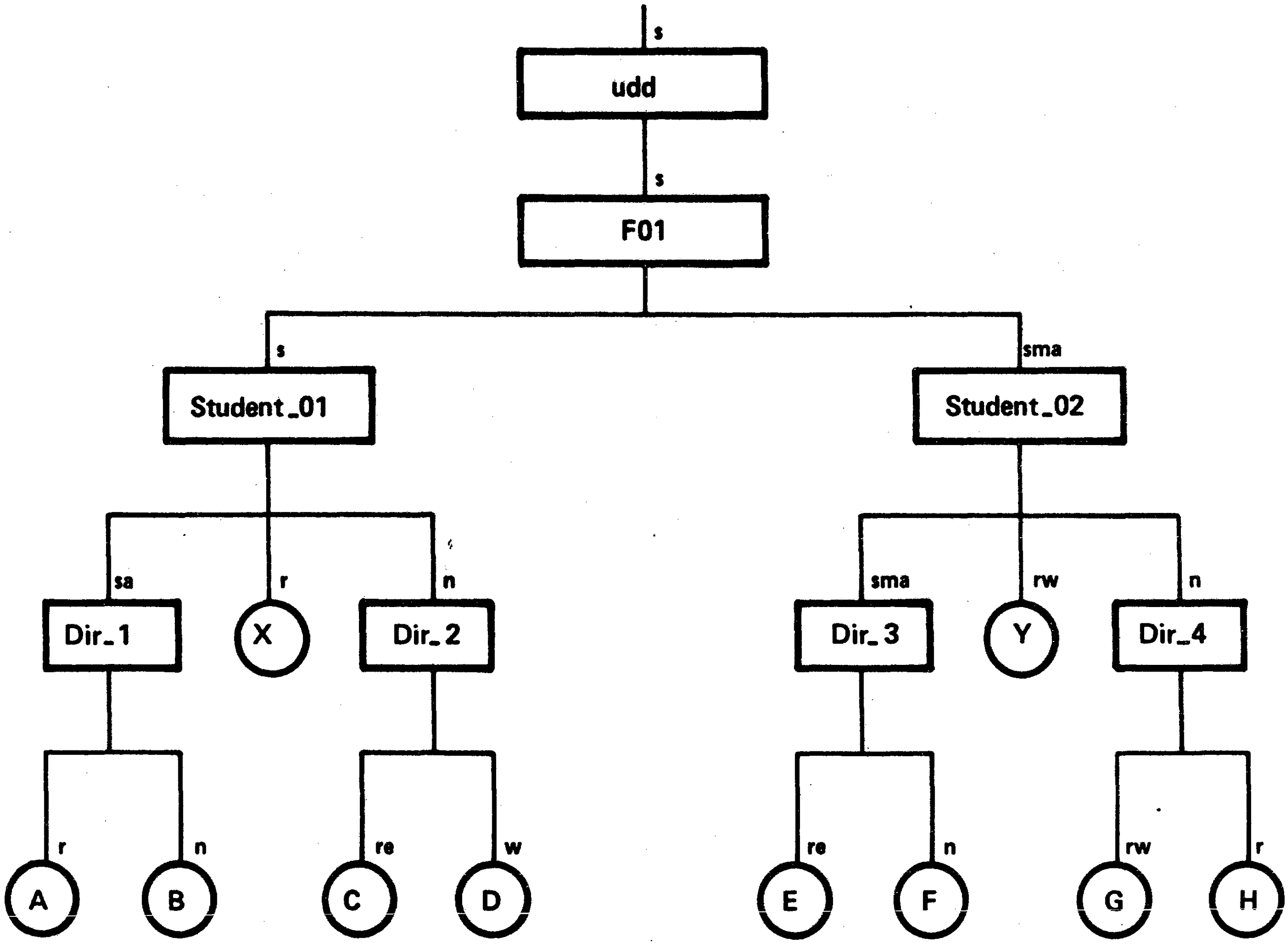
  - A FACILITY FOR DEFINING ADDITIONAL DEFAULT ACL ENTRIES TO BE INCLUDED IN THE ACL OF SEGMENTS AND DIRECTORIES WHEN CREATED

  - DEFINABLE AT THE DIRECTORY LEVEL

  - REFER TO THE DESCRIPTION OF THE FOLLOWING COMMANDS IN THE MULTICS COMMANDS MANUAL

set_iacl_seg,	sis	set_iacl_dir,	sid
list_iacl_seg,	lis	list_iacl_dir,	lid
delete_iacl_seg,	dis	delete_iacl_dir,	did

ACCESS EXAMPLE FOR STUDENT_02



ACCESS EXAMPLES

- IN WHICH DIRECTORIES CAN Student_02 SUCCESSFULLY EXECUTE THE list COMMAND?

- IN WHICH DIRECTORIES CAN Student_02 CREATE A SEGMENT?

- SUPPOSE Student_02 CREATES A SEGMENT IN DIRECTORY Dir_1 BY TYPING

```
     create >udd>F01>Student_01>Dir_1>Z
```

  CAN Student_02 DELETE THIS SEGMENT? no - need m permission on directory.

  CAN Student_02 rename THIS SEGMENT? no "

  CAN Student_02 READ AND WRITE THE CONTENTS OF THIS SEGMENT? yes

- IN WHICH DIRECTORIES CAN Student_02 SUCCESSFULLY EXECUTE THE rename COMMAND? anything under STU 2 & DIR 3

- IN WHICH DIRECTORY CAN Student_02 SUCCESSFULLY EXECUTE THE set_acl COMMAND?

ACCESS EXAMPLES

- TO WHICH SEGMENTS CAN Student_02 WRITE?

  D, Y, G

- TO WHICH SEGMENTS COULD Student_02 EVENTUALLY WRITE BY SETTING THE APPROPRIATE ACCESS?

  anything under STU_02

- SUPPOSE Student_02 CREATES A DIRECTORY UNDER Dir_1 BY TYPING:

```
create_dir >udd>F01>Student_01>Dir_1>Dir_5
```

WHAT PERMISSIONS WILL Student_02 HAVE ON THIS DIRECTORY? S, M, A

WHAT PERMISSIONS WILL Student_01 HAVE ON THIS DIRECTORY? n

CAN Student_01 GIVE HIMSELF PERMISSIONS ON Dir_5? yes

This page has intentionally

been left blank.

TOPIC XII

USER COMMUNICATION

This page has intentionally
been left blank.

MESSAGE FACILITY

- MESSAGE — goes to terminal if logged on if possible,

  - TEXT WHICH IS COMMUNICATED BETWEEN USERS VIA THE send_message COMMAND

go to mailbox

- MAILBOX — goes directly to mailbox

  - A SEGMENT IN THE DIRECTORY >udd>Project_id>Person_id HAVING THE NAME Person_id.mbx

- accept_messages, am (I'M LISTENING)

  - ALLOWS ANY AND ALL INCOMING MESSAGES TO BE PRINTED ON THE USER'S TERMINAL

  - OTHERWISE, MESSAGE WILL GO TO THE USER'S MAILBOX

  - ALSO CREATES A MAILBOX IF NONE EXISTS

  - USAGE:

    ```
    accept_messages {-control_args}

    am

    am -print

    am -brief
    ```

MESSAGE FACILITY

- send_message, sm

  - SENDS A MESSAGE TO A SPECIFIED USER ON A SPECIFIED PROJECT

  - SMALL MESSAGES (ONE LINE)

  - MESSAGES ARE EITHER

    - PRINTED ON THE RECIPIENTS TERMINAL, OR

    - PLACED IN THE RECIPIENTS MAILBOX

  - USAGE 1: send_message Person_id.Project_id message

```
sm TSmith.Project_id When are you going to lunch?

sm Greenberg.FED May I have access to your file?
```

  - USAGE 2: send_message Person_id.Project
    (DIALOGUE MODE)

```
sm TSmith.FED
Input:
When are you going to lunch?
From TSmith.FED 11/10/78 1546.3 mst Fri: 12:00
Mary wants to go with us.
=:Fine, bring her along.
Meet you in the lobby.
.
```

MESSAGE FACILITY

- defer_messages, dm (I'M BUSY...NO DISTRACTIONS WANTED)

  ▯ REDIRECTS ANY AND ALL INCOMING MESSAGES TO THE USER'S MAILBOX

  ▯ ELIMINATES UNWANTED INTERRUPTIONS

  ▯ USAGE:

  ```
  defer_messages
  dm
  ```

- print_messages, pm

  ▯ PRINTS ALL MESSAGES STORED IN THE USER'S MAILBOX

  ▯ MESSAGES ARE DELETED FROM THE MAILBOX WHEN PRINTED

  ▯ USAGE:

  ```
  print_messages  {-control_arg}
  pm
  pm -last
  ```

MAIL FACILITY

- MAIL

  - ~~TEXT WHICH~~ IS COMMUNICATED BETWEEN USERS VIA THE mail COMMAND

Rdm, Read_mail - comes back and asks questions

- mail, ml

  - PRINTS ANY AND ALL MAIL (OR MESSAGES) IN A USER'S MAILBOX, OR

  - SENDS THE CONTENTS OF A SEGMENT TO ANOTHER USER

  - WHEN SENDING, MAIL IS PLACED IN THE RECIPIENT'S MAILBOX

  - USAGE 1: mail {path} {-control_arg}
    (PRINTING)

```
ml

ml -bf

ml >udd>F01>Student_04>Student_04.mbx
```

  - ALSO CREATES A RING-PROTECTED MAILBOX IF NONE EXISTS

  - USAGE 2: mail path User_ids {-control_arg}
    (SENDING)

```
mail fw16.report LJones.FED

mail letter Student_05.F01 -ack

mail S_letter.runout TSmith.ProjA Boyd.ProjA
```

MAIL FACILITY

```
USAGE 3:    mail * User_ids
(SENDING)
```

```
mail * LJones.FED TSmith.ProjA
Input:
The finance committee will begin meeting
on Tuesdays at 3:30 starting
15 June.  Coffee will be provided.
.
```

by itself to get back to command level

- RELATED COMMANDS

  mbx_create, mbcr

  mbx_delete, mbdl

  mbx_add_name, mban

  mbx_delete_name, mbdn

  mbx_rename, mbrn

  mbx_set_acl, mbsa

  mbx_delete_acl, mbda

  mbx_list_acl, mbla

  mbx_set_max_length, mbsml

MEMO FACILITY

- MEMO FACILITY

  - AN INTERACTIVE NOTEBOOK AND REMINDER LIST

  - MAINTAINED BY THE USER IN A HOME DIRECTORY SEGMENT NAMED Person_id.memo

- MEMO

  - A MESSAGE DELIVERED TO THE USER AT A PREDETERMINED DATE AND TIME, OR

  - A COMMAND EXECUTED BY THE SYSTEM AT A PREDETERMINED DATE AND TIME

  - MATURE MEMOS MAY BE ACTIVATED

    - EXPLICITLY (VIA THE memo COMMAND), OR

    - AUTOMATICALLY (WHILE USER IS LOGGED IN)

  - MEMOS OPERATE INDEPENDENT OF THE accept_messages/defer_messages COMMANDS

MEMO FACILITY

- memo

- CREATES AND MAINTAINS AN INTERACTIVE NOTEBOOK AND REMINDER LIST

- USAGE 1: memo
  (EXECUTING)

```
memo
```

- USAGE 2: memo -list {optional_args}
  (LISTING)

```
memo -list

memo -list -match Birthday
```

- USAGE 3: memo {optional_args} memo_text
  (SETTING)

```
memo Good job Tom!!!  Keep up the good work!

memo -time "Friday 8am est" 10am meeting with Olson

memo -date 5/6/78 -repeat 1year Jan's Birthday:May 9

memo -call -tm Friday sm May ProjA Report due today.

memo -alarm -repeat 1day -time noon Lunch time!
```

- USAGE 4: memo -delete optional_args
  (DELETING)

```
memo -delete -match Birthday

memo -delete -call
```

This page has intentionally

been left blank.

TOPIC XIII

MULTICS INPUT/OUTPUT FACILITIES

This page has intentionally
been left blank.

MULTICS INPUT/OUTPUT

- LOGICAL I/O

- DEVICE INDEPENDENT

- SYSTEM I/O MODULES CONTROL THE PHYSICAL DEVICES

- I/O "SWITCHES" CHANNEL THE FLOW OF DATA BETWEEN PROGRAM ACCESSIBLE STORAGE AND DEVICES, FILES, ETC

- SYSTEM INPUT/OUTPUT MODULES

  - THE Multics SYSTEM CONTAINS THE FOLLOWING I/O MODULES:

    - discard_

      IS A SINK FOR UNWANTED OUTPUT

    - rdisk_

      SUPPORTS I/O FROM/TO REMOVABLE DISK PACKS

    - record_stream_

      PROVIDES A MECHANISM FOR DOING RECORD I/O ON AN UNSTRUCTURED FILE, OR VICE VERSA

    - syn_

      ESTABLISHES ONE SWITCH AS A SYNONYM FOR ANOTHER

    - tape_ansi_

      SUPPORTS I/O FROM/TO MAGNETIC TAPE FILES ACCORDING TO STANDARDS PROPOSED BY THE AMERICAN NATIONAL STANDARDS INSTITUTE (ANSI)

    - tape_ibm_

      SUPPORTS I/O FROM/TO MAGNETIC TAPE FILES ACCORDING TO STANDARDS ESTABLISHED BY IBM

    - tape_mult_

      SUPPORTS I/O FROM/TO MAGNETIC TAPE FILES IN Multics STANDARD TAPE FORMAT

SYSTEM INPUT/OUTPUT MODULES

- tty_

  SUPPORTS I/O FROM/TO TERMINALS

- vfile_

  SUPPORTS I/O FROM/TO FILES IN THE STORAGE SYSTEM

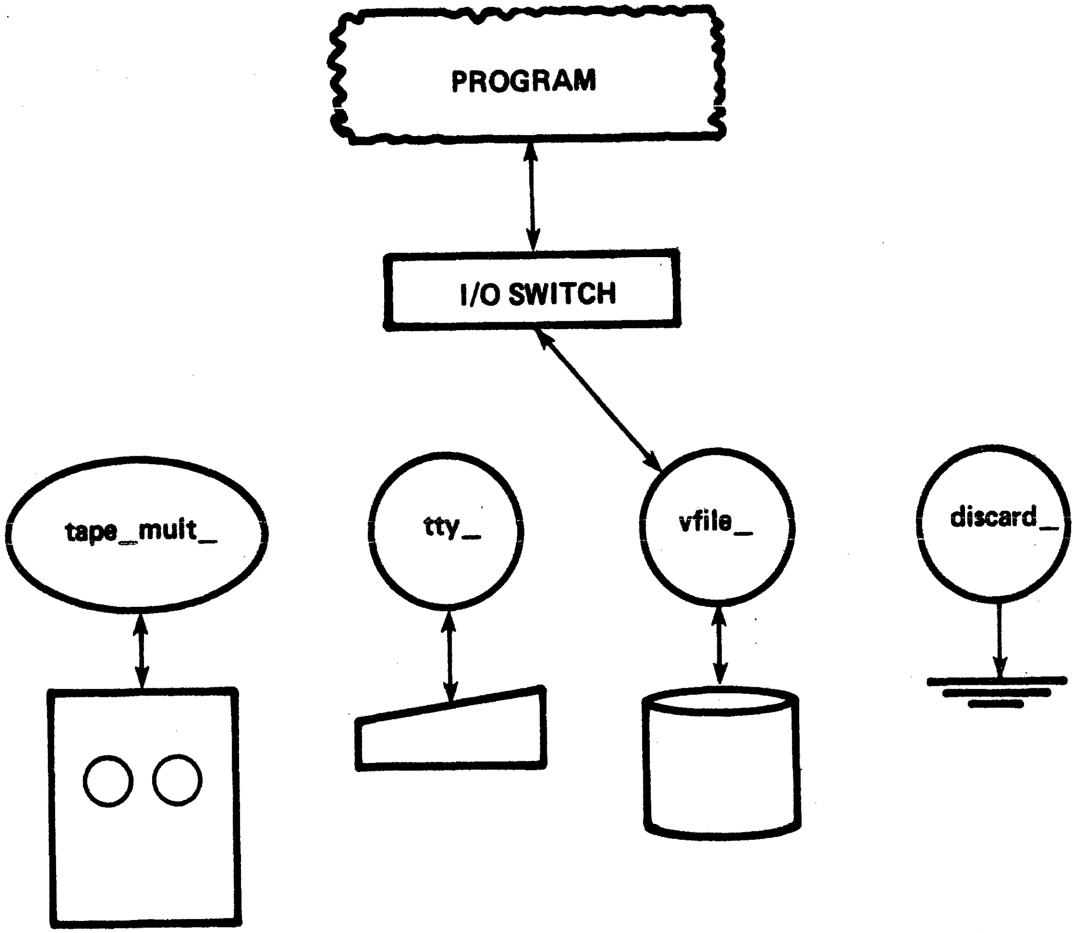
- THESE MODULES ARE DESCRIBED IN SECTION III OF THE MPM SUBROUTINES AND IN THE MPM PERIPHERAL INPUT/OUTPUT

- THE USER MAY CONSTRUCT HIS OWN I/O SYSTEM INTERFACE MODULES. SEE "WRITING AN I/O MODULE" IN SECTION IV OF THE MPM SUBSYSTEM WRITERS' GUIDE

INPUT/OUTPUT SWITCHES

- SOFTWARE CONSTRUCT WHICH MAKES I/O DEVICE INDEPENDENT

- CONNECTS THE SOURCE OF A READ OR WRITE TO THE TARGET (FILE, TAPE, ETC.) THROUGH A SYSTEM I/O MODULE



INPUT/OUTPUT SWITCHES

- TO PERFORM I/O, THE FOLLOWING FIVE STEPS MUST BE CARRIED OUT (EITHER EXPLICITLY OR IMPLICITLY):

  1) ATTACH AN I/O SWITCH. THIS STEP SPECIFIES THE SEGMENT PATHNAME, TAPE VOLUME NAME, ETC. FROM/TO WHICH THE INPUT/OUTPUT OPERATION IS MADE AND THE I/O MODULE WHICH PERFORMS THE OPERATION (vfile_, tape_ansi_, etc.)

  2) OPEN THE I/O SWITCH. THIS STEP PREPARES THE SWITCH FOR A PARTICULAR MODE OF PROCESSING (E.G. READING RECORDS SEQUENTIALLY) USING THE ALREADY ESTABLISHED ATTACHMENT

  3) PERFORM THE REQUIRED DATA TRANSFER WORKING THROUGH THE SWITCH

  4) CLOSE THE I/O SWITCH

  5) DETACH THE I/O SWITCH

- SWITCHES MAY BE ATTACHED BY

  ▯ io_call COMMAND

  ▯ SUBROUTINE CALL TO iox_$attach_ioname

  ▯ LANGUAGE OPEN STATEMENT (IF NOT PREVIOUSLY ATTACHED)

  ▯ DEFAULT WHEN RUNNING FORTRAN, PL/1, AND COBOL

INPUT/OUTPUT SWITCHES

- SWITCHES MAY BE OPENED BY
  - io_call COMMAND
  - SUBROUTINE CALL TO iox_$open LANGUAGE
  - LANGUAGE OPEN STATEMENTS
  - DEFAULT WHEN RUNNING FORTRAN, PL/1, AND COBOL

- DATA TRANSFER MAY BE PERFORMED BY:
  - io_call COMMAND
  - SUBROUTINE CALL TO iox_, ioa_
  - get, put, read, write, etc. in PL/1
  - read, write, etc. in FORTRAN
  - read, write, etc. in COBOL
  - I/O STATEMENTS IN OTHER LANGUAGES

INPUT/OUTPUT SWITCHES

- THE I/O SWITCH MAY BE CLOSED BY:

  - io_call COMMAND

  - LANGUAGE close STATEMENT (IF THE SWITCH WAS OPENED BY A LANGUAGE OPEN STATEMENT

  - close_file COMMAND

- THE I/O SWITCH MAY BE DETACHED BY:

  - io_call COMMAND

  - SUBROUTINE CALL TO iox_$detach_iocb

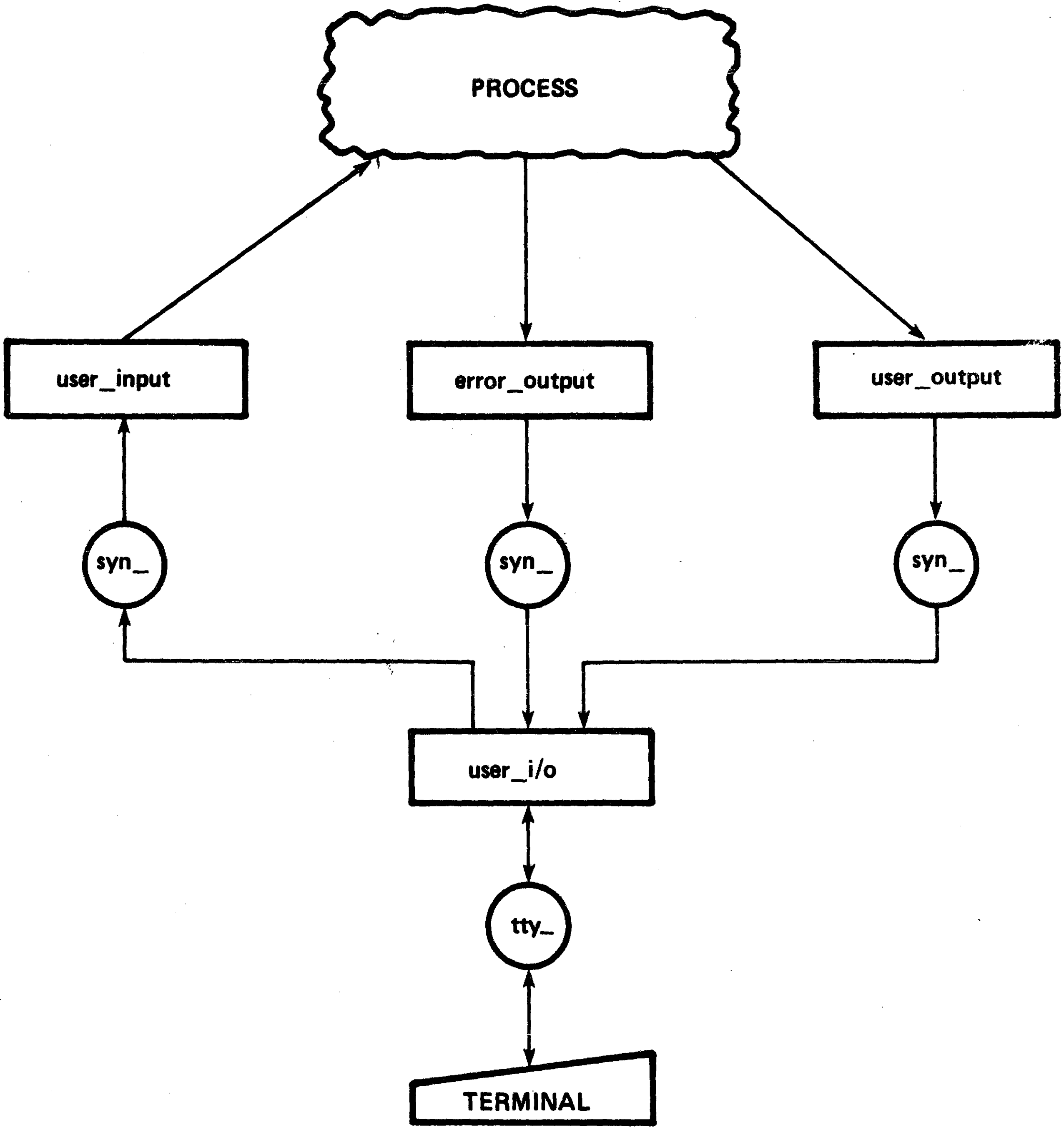
  - LANGUAGE close STATEMENT (IF THE SWITCH WERE ATTACHED BY THE LANGUAGE open STATEMENT)

INPUT/OUTPUT SWITCHES

- FOUR SWITCHES ATTACHED DURING PROCESS CREATION (login, new_proc)
  - user_i/o
  - user_input
  - user_output
  - error_output

- user_i/o IS ATTACHED TO THE USER'S TERMINAL THROUGH tty_ AN OPENED FOR STREAM INPUT AND OUTPUT

- user_input, user_output, and error_output ARE ATTACHED TO user_i/o THROUGH syn_ AND ARE OPENED FOR INPUT, OUTPUT, AND OUTPUT RESPECTIVELY



STANDARD ATTACHMENTS

INPUT/OUTPUT COMMANDS

- io_call, io

  - PERFORMS AN OPERATION ON A DESIGNATED I/O SWITCH

  - USAGE:

```
io_call opname switchname {args}

io attach payroll_tape tape_ansi_ payrol
     -cr -nm employee_rec
     -nb 1 -retain all

io attach poem vfile_ >udd>F01>Student_02>The_Ravin

io open poem stream_input

io get_line poem

io close poem

io detach poem
```

- close_file, cf

  - CLOSES SPECIFIED FORTRAN AND PL/1 FILES

  - USAGE:

```
close_file {-control_arg} filenames

close_file poem file08

close_file -all
```

- print_attach_table, pat

  - PRINTS INFORMATION ON THE USER'S TERMINAL ABOUT I/O SWITCH ATTACHMENTS

  - USAGE:
    ```
    print_attach_table {-control_args} {switch_names}

    pat

    pat poem
    ```

- file_output, fo

  - DIRECTS ALL SUBSEQUENT USER'S OUTPUT (TERMINAL OUTPUT) TO A SEGMENT UNTIL THE revert_output COMMAND IS ENCOUNTERED

  - ATTACHES user_output TO A SPECIFIED OUTPUT FILE

  - ERROR MESSAGES (IF THEY OCCUR) STILL APPEAR ON THE USER'S TERMINAL

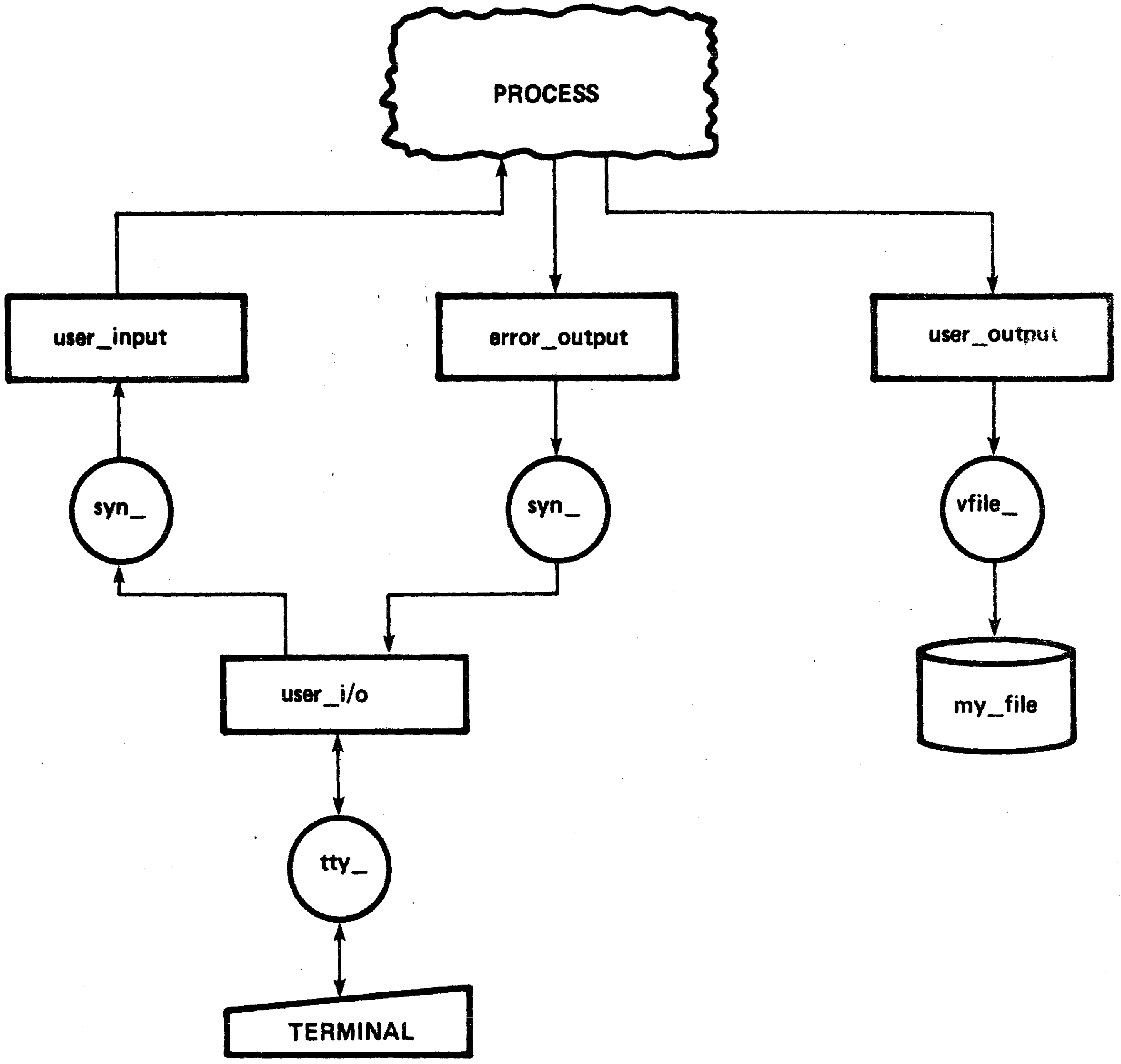
  - USAGE:
    ```
    file_output {path}

    fo who_save

    fo
    ```

- revert_output, ro

  - RESTORES USER'S OUTPUT (TERMINAL OUTPUT) TO THE TERMINAL

  - USAGE:

    ```
    revert_output

    ro
    ```



file_output my_file

INPUT/OUTPUT COMMANDS

- copy_file, cpf

  - COPIES RECORDS FROM A STRUCTURED INPUT FILE TO AN OUTPUT FILE

  - USAGE: copy_file in_ctrl_arg out_ctrl_arg {-control_args}

```
cpf -input_description "vfile_ >udd>F01>Student_01>funky"
      -output_switch funky_sw

cpf -isw funky_sw -ods "tape_ansi foo -nm first_file -nb 1"

cpf -ct 13 -ids "tape_ansi_ 887677 -nm TEST21 -ret all"
      -ods "record_stream_ user_output"

cpf -isw in -osw out -from 43 -to 78
```

- copy_cards

  - COPIES SPECIFIED CARD IMAGE SEGMENTS FROM THE SYSTEM POOL STORAGE INTO A USER'S DIRECTORY

  - THE SEGMENTS TO BE COPIED MUST HAVE BEEN CREATED USING THE Multics CARD IMAGE FACILITY

  - USAGE: copy_cards deck_name {pathname}

```
copy_cards card_deck

copy_cards card_deck my_file
```

EXAMPLES

- EXAMPLE 1

□ READ RECORDS FROM A TAPE WITH A VOLUME NAME OF payrol AND A FILE NAME OF emp

□ READ THESE RECORDS ONE AT A TIME FROM COMMAND LEVEL

```
! io attach pt tape_ansi_ payrol -name emp -nb 1 -ret all
! io open payroll _tape sequential_input
! pat pt
  pt       tape_ansi_ payrol -name emp -nb 1 -ret all sequential_input

! io read pt 85
  io_call: 85 characters returned. 15093Robert Redford        5534 W.Yucca

! io read pt 85
  io_call: End of information reached.  No data returned by pt.
! io close pt
! pat pt
  pt        tape_ansi_ payrol -name emp -number 1 -retain all
                                             (not open)
! io detach pt
```

□ NOTE: THE TAPE RESOURCES WERE ALLOCATED PRIOR TO THE ATTACH. OTHERWISE, A MESSAGE WHICH WOULD INDICATE THE TAPE WAS BEING MOUNTED WOULD HAVE BEEN RETURNED AFTER THE io attach

EXAMPLES

- EXAMPLE 2

  - READ DATA FROM A FILE AND WRITE TO ANOTHER

  - USING THE SAME PROGRAM, READ FROM A FILE AND WRITE TO THE TERMINAL. (USE io_call AND ATTACH THE OUTPUT SWITCH EXTERNAL TO THE PROGRAM)

  - THE PROGRAM:

```
example_2: proc;
dcl  payroll_in file,
     payroll_out file,
     sysprint file,
     1 emp_record,
     2 pay_no char (5),
     2 emp,
     3 name char (20),
     3 address char (20),
     (endfile, record, transmit) condition;

          on endfile (payroll_in) go to fini;
          on record (payroll_in) ;
          on transmit (payroll_in) begin;
               put skip list
               ("TRANSMIT ERROR. LAST RECORD READ WAS:", emp_record);
               go to fini;
          end;

          open file (payroll_in)
          title ("vfile_ payroll_file") input;
          open file (payroll_out)
          title ("vfile_ payroll_file_2")  output;

          do while ("1"b);
               get file (payroll_in) list (emp_record);
               put file (payroll_out) skip list (emp_record);
          end;

 fini:    close file (payroll_in), file (payroll_out);
          put skip list ("done");
     end example_2;
```

EXAMPLES

□ INPUT FILE, payroll_file:

```
"12002", "Barbara Striesand", "4040 N. 30th lane"
"15093", "Robert Redford", "5534 W. Yucca"
"15666", "Julie Christie", "3322 W. Milky Way"
```

□ SEQUENCE OF EXECUTION

```
! example_2

  done
! pr payroll_file_2

                    payroll_file_2      11/09/78 1710.8 mst Thu


  "12002" "Barbara Striesand     " "4040 N. 30th Lane    "
  "15093" "Robert Redford        " "5534 W. Yucca        "
  "15666" "Julie Christie        " "3322 W. Milky Way    "

! pat
  user_i/o            tty_ tty724         stream_input_output
  user_input          syn_ user_i/o
  user_output         syn_ user_i/o
  error_output        syn_ user_i/o
  sysprint            syn_ user_output

! io attach payroll_out syn_ user_output
! pat payroll_out
  payroll_out         syn_ user_output
! example_2

  "12002" "Barbara Striesand     " "4040 N. 30th Lane    "
  "15093" "Robert Redford        " "5534 W. Yucca        "
  "15666" "Julie Christie        " "3322 W. Milky Way    "
  done
! pat payroll_out
  payroll_out         syn_ user_output
```

EXAMPLES

YOU ARE NOW READY FOR WORKSHOP #6

EXAMPLES

- EXAMPLE 3

  - READ A DECK OF CARDS INTO THE SYSTEM AND CREATE A SEGMENT IN YOUR WORKING DIRECTORY CONTAINING THE CARD IMAGES

  - STEP 1: YOU MUST BE REGISTERED BY THE SYSTEM ADMINISTRATOR FOR CARD INPUT. YOU WILL RECEIVE A PASSWORD FOR YOUR CARD DECKS

  - STEP 2: CREATE A SEGMENT IN YOUR HOME DIRECTORY CALLED card_input.acs

  - STEP 3: SET ACL ON THE SEGMENT TC "r" FOR <STATION_ID>.*.* AND FOR Card_Input.Daemon.*

  - STEP 4: PREPARE THE CARD DECK

```
++DATA DECK_NAME PERSON_ID PROJECT_ID
++PASSWORD xxx
++FORMAT MCC LOWERCASE
++INPUT
   .
   .
```

  - STEP 5: SUBMIT THIS DECK TO THE OPERATOR. A MESSAGE WILL BE SENT TO YOU WHEN IT HAS BEEN READ

  - STEP 6: EXECUTE THE COPY CARDS COMMAND

```
copy_cards deck_name
```

TOPIC XIV

MORE ABOUT THE ABBREV PROCESSOR

This page has intentionally
been left blank.

THE DO COMMAND

- MOTIVATION:

  - WOULD LIKE TO BE ABLE TO DO THE FOLLOWING:

    ```
    .ab PL1 ind; pl1
    PL1 add.pl1
    ```

  - WHICH EXPANDS TO:

    ```
    ind; pl1 add.pl1
    ```

  - BUT WANT:

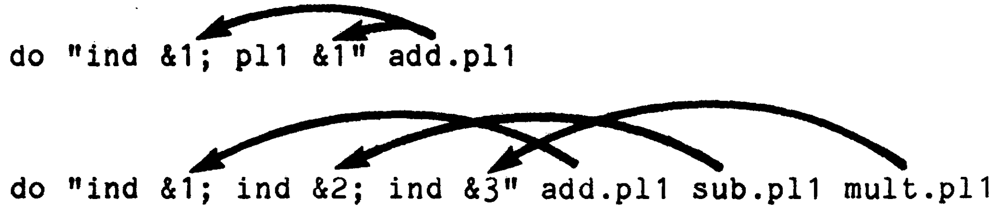
    ```
    ind add.pl1; pl1 add.pl1
    ```

- do

  - SUBSTITUTES SUPPLIED ARGUMENTS INTO A COMMAND LINE

  - PRIMARILY INTENDED FOR USE IN ABBREVIATIONS

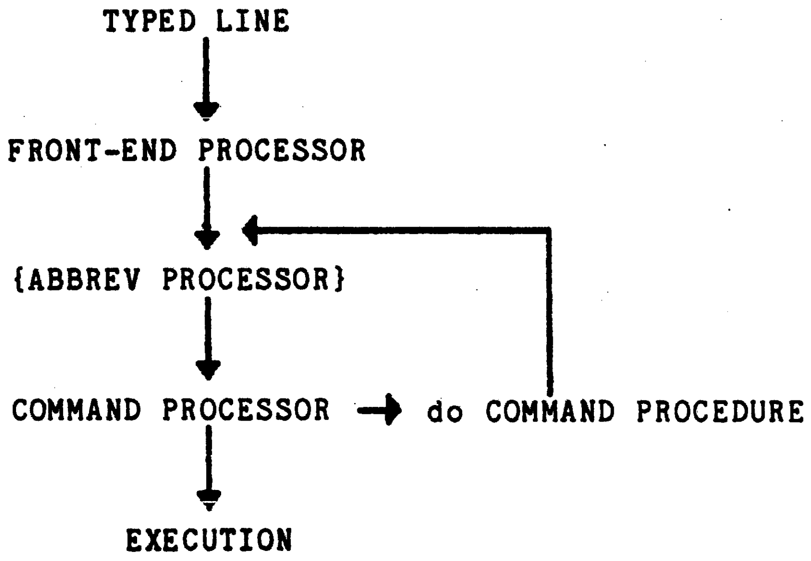
  - USAGE: do "command line" args



```
do "ind &1; pl1 &1" add.pl1

do "ind &1; ind &2; ind &3" add.pl1 sub.pl1 mult.pl1
```

THE DO COMMAND

- COMMAND LINE FLOW



TYPED LINE

FRONT-END PROCESSOR

{ABBREV PROCESSOR}

COMMAND PROCESSOR → do COMMAND PROCEDURE

EXECUTION

THE DO COMMAND

- EXAMPLES USING THE FOLLOWING ABBREVS:

```
PL1          do "ind &1; pl1 &1"
PL1_DP       do "PL1 &1; dp &1"
PL1_EX       do "ind &1.pl1; pl1 &1.pl1; &1"
```

- EXAMPLE 1:

```
PL1 add.pl1                                  (AS TYPED)
do "ind &1; pl1 &1" add.pl1                  (ABBREV PROCESSOR)
ind add.pl1; pl1 add.pl1                     (DO COMMAND)
```

- EXAMPLE 2:

```
PL1_EX add                                   (AS TYPED)
do "ind &1.pl1; pl1 &1.pl1; &1" add          (ABBREV PROCESSOR)
ind add.pl1; pl1 add.pl1; add                (DO COMMAND)
```

THE DO COMMAND

▯ EXAMPLE 3:

```
PL1_DP add.pl1                                (AS TYPED)
do "PL1 &1; dp &1" add.pl1                    (ABBREV PROCESSOR)
PL1 add.pl1; dp add.pl1                       (DO COMMAND)
do "ind &1; pl1 &1" add.pl1; dp add.pl1       (ABBREV PROCESSOR)
ind add.pl1; pl1 add.pl1; dp add.pl1          (DO COMMAND)
```

AREAS FOR ADDITIONAL STUDY

- ADDITIONAL DOCUMENTATION
  - MPM COMMANDS AND ACTIVE FUNCTIONS (AG92)
  - help abbrev
- STUDY TOPICS
  - ADDITIONAL abbrev REQUESTS

.u	(USE ANOTHER PROFILE)
.p	(PRINT THE PATHNAME OF THE PROFILE BEING USED)
.af	(FORCE REDEFINE)
.abf	(FORCE REDEFINE)
.r	(REMEMBER MODE)
.f	(FORGET MODE - THE DEFAULT)
.s	(SHOW LAST LINE)

This page has intentionally

been left blank.

TOPIC XV

ACTIVE FUNCTIONS

This page has intentionally
been left blank.

WHAT IS AN ACTIVE FUNCTION

- ACTIVE STRING

  - A SUB-STRING (A PART) OF A COMMAND LINE DELIMITED (SET OFF) BY SQUARE BRACKETS

  - INTENDED TO BE REPLACED BY A CORRESPONDING VALUE

  - EXAMPLES:

    ```
    sm [last_message_sender] Thank you!
    sm LJones.ProjA Thank you!

    delete [oldest_segment]
    delete seg_1
    ```

  - LIKE A DYNAMIC (OR VARIABLE) ABBREVIATION

  - FREQUENTLY USED WITHIN ABBREVIATIONS

  - THERE ARE MORE THAN 70 ACTIVE STRINGS DEFINED BY MULTICS

WHAT IS AN ACTIVE FUNCTION

- ACTIVE FUNCTION

  - A PROGRAM EXPLICITLY DESIGNED TO EVALUATE AND RETURN THE VALUE OF AN ACTIVE STRING

  - USERS MAY DEFINE THEIR OWN ACTIVE STRINGS AND WRITE THEIR OWN CORRESPONDING ACTIVE FUNCTIONS

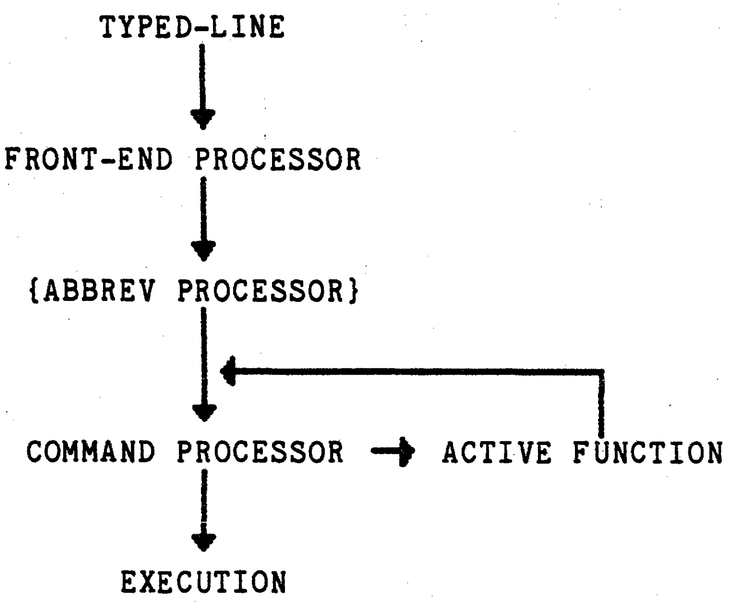
  - MANY ACTIVE FUNCTIONS MAY ALSO BE INVOKED AS COMMANDS

```
last_message_sender

date
```

ACTIVE FUNCTION MECHANISM

- ACTIVE FUNCTION MECHANISM
  - ACTIVE STRINGS ARE IMMEDIATELY EVALUATED (EXECUTED)
  - THE RESULTING VALUE IS SUBSTITUTED FOR THE ACTIVE STRING IN THE COMMAND LINE
  - THE COMMAND LINE IS THEN RETURNED TO THE COMMAND PROCESSOR

- COMMAND LINE FLOW



TYPED-LINE

FRONT-END PROCESSOR

{ABBREV PROCESSOR}

COMMAND PROCESSOR → ACTIVE FUNCTION

EXECUTION

ACTIVE FUNCTION EXAMPLES

- last_message_sender, lms

  - RETURNS THE User_id OF THE SENDER WHO SENT THE LAST MESSAGE RECEIVED

  - USAGE 1: last_message_sender

    ```
    lms
    LJones.FED
    ```

  - USAGE 2: [last_message_sender]

    ```
    sm [lms] THANK YOU!
    sm LJones.FED THANK YOU!

    who [lms]
    who LJones.FED
    ```

- wd

  - RETURNS THE PATHNAME OF THE USERS WORKING DIRECTORY

  - USAGE 1: wd

    ```
    wd
    >udd>FED>May
    ```

  - USAGE 2: [wd]

    ```
    sm Kerr.FED THE pathname is [wd]>add.pl1
    sm Kerr.FED THE pathname is >udd>FED>May>add.pl1
    ```

ACTIVE FUNCTION EXAMPLES

- date

  - RETURNS THE DATE IN mm/dd/yy FORM

  - USAGE 1: date {dt}

    ```
    date
    11/01/77

    date "12 June"
    06/12/78
    ```

  - USAGE 2: [date {dt}]

    ```
    create alpha.version_[date].pl1
    create alpha.version_11/01/77.pl1
    ```

- home_dir

  - RETURNS THE PATHNAME OF THE USERS HOME DIRECTORY

  - USAGE 1: home_dir

    ```
    home_dir
    >udd>F01>Student_07
    ```

  - USAGE 2: [home_dir]

    ```
    pr [home_dir]>add.pl1
    pr >udd>F01>Student_07>add.pl1
    ```

ACTIVE FUNCTION EXAMPLES

- segments, segs

  - RETURNS THE ENTRYNAMES (SEPARATED BY A BLANK) OF ALL SEGMENTS MATCHING A GIVEN STARNAME

  - USAGE: [segments starname]

```
dprint [segs **.pl1]
dprint add.pl1 seg_1.pl1
```

- contents

  - RETURNS THE CONTENTS OF A SPECIFIED ASCII SEGMENT SEPARATED B' BLANKS

  - USAGE: [contents path]

```
sm TSmith.FED Their names are: [contents Names]
sm TSmith.FED Their names are: LJones.FED Kerr.MED

mail letter.2 [contents Names]
mail letter.2 LJones.FED Kerr.MED
```

AREAS FOR ADDITIONAL STUDY

- ADDITIONAL DOCUMENTATION

  - MPM COMMANDS AND ACTIVE FUNCTIONS (AG92)

  - help <active function name>

- STUDY TOPICS

  - ARITHMETIC ACTIVE FUNCTIONS

    - ceil, divide, floor, max, min, minus, mod, plus, quotient, times, trunc

  - CHARACTER STRING ACTIVE FUNCTIONS

    - default, format_line, index, index_set, length, search, string, substr, underline, unique, verify

  - DATE AND TIME ACTIVE FUNCTIONS

    - date, date_time, day, day_name, hour, long_date, minute, month, month_name, time, year

  - LOGICAL ACTIVE FUNCTIONS

    - and, equal, exists, greater, less, nequal, ngreater, nless, not, or

AREAS FOR ADDITIONAL STUDY

- PATHNAME MANIPULATION ACTIVE FUNCTIONS
  - directory, entry, equal_name, path, strip, strip_entry, suffix
- QUESTION ASKING ACTIVE FUNCTIONS
  - query, response
- STORAGE SYSTEM ATTRIBUTES ACTIVE FUNCTIONS
  - lv_attached, status
- STORAGE SYSTEM NAMES ACTIVE FUNCTIONS
  - directories, files, get_pathname, home_dir, links, nondirectories, nonlinks, nonsegments, pd, segments, wd
- USER PARAMETER ACTIVE FUNCTIONS
  - have_mail, last_message_sender, last_message_time, last_message, system, user

TOPIC XVI

MORE ABOUT EXEC_COM'S

This page has intentionally
been left blank.

REVIEW

- EXEC_COM

  - A SEGMENT CONTAINING A SERIES OF COMMAND LINES

  - CREATED USING A TEXT EDITOR

  - NAME MUST HAVE SUFFIX OF ec

  - THE COMMAND LINES ARE EXECUTED SEQUENTIALLY, AS A SET, WHENEVER INVOKED BY THE USER

  - ABBREVIATIONS ARE EXPANDED IN THE NORMAL MANNER

  - MAY BE RECURSIVELY INVOKED

- exec_com, ec

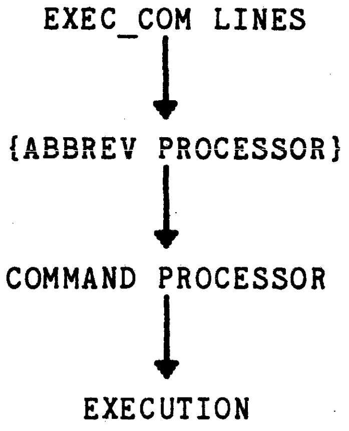
  - EXECUTES THE COMMAND LINES CONTAINED IN AN EXEC_COM SEGMENT

  - USAGE:

    ```
    exec_com path {optional_args}

    ec A.ec

    ec weird.ec flower tree add
    ```

REVIEW

- EXEC_COM LINE FLOW



EXEC_COM LINES

{ABBREV PROCESSOR}

COMMAND PROCESSOR

EXECUTION

ARGUMENT SUBSTITUTION

- ARGUMENT SUBSTITUTION

  - CERTAIN CHARACTER STRINGS ARE REPLACED BY A CORRESPONDING VALUE BEFORE THE EXEC_COM IS EXECUTED

  - EXAMPLES USING EITHER OF THE FOLLOWING COMMAND LINES

```
ec weird.ec flower tree add
ec >udd>F01>Student_05>weird.ec flower tree add
```

    - &<number>

      - REPLACED BY THE CORRESPONDING OPTIONAL ARGUMENT (POSITIONAL)

        &1 ← flower

        &3 ← add

        &5 ← <nothing>

    - &n

      - REPLACED BY THE NUMBER OF ARGUMENTS SUPPLIED

        &n ← 3

    - &ec_name

      - REPLACED BY THE ENTRYNAME PORTION OF THE EXEC_COM'S PATHNAME WITHOUT THE ec SUFFIX

        &ec_name ← weird

    - &ec_dir

      - REPLACED BY THE DIRECTORY NAME PORTION OF THE EXEC_COM PATHNAME

        &ec_dir ← >udd>F01>Student_05

CONTROL STATEMENTS

- CONTROL STATEMENTS
  - ▯ EXEC_COM LINES THAT BEGIN WITH SPECIAL KEY WORDS
  - ▯ ALL KEY WORDS BEGIN WITH & (AMPERSAND)
  - ▯ PROVIDE VARIETY AND CONTROL DURING EXECUTION

- &command_line off
  - ▯ SUPPRESSES THE PRINTING OF SUBSEQUENT COMMAND LINES
  - ▯ USAGE: &command_line off

- &command_line on
  - ▯ CAUSES SUBSEQUENT COMMAND LINES TO BE PRINTED. (THE DEFAULT)
  - ▯ USAGE: &command_line on

CONTROL STATEMENTS

- &print text

  - CAUSES THE TEXT FOLLOWING &print TO BE PRINTED ON THE USER'S TERMINAL

  - USAGE: &print text

    &print BEGINNING COMPILE PHASE

- &label location

  - IDENTIFIES A PLACE TO WHICH AN &goto TRANSFERS CONTROL

  - USAGE: &label label_name

    &label Arg_Check

    &label &1

- &goto location

  - CAUSE CONTROL TO BE TRANSFERRED TO THE PLACE SPECIFIED

  - USAGE: &goto label_name

    &goto Arg_Check

    &goto &1

stty -modes echoplex, polite, replay

CONTROL STATEMENTS

→ used to determine terminal type then characterize terminal
→ wait until final command to accept messages – polite or replay
→ used for echoplex
→ used to determine type of login & do part of start_up.ec

● &if, &then, &else

▯ ALLOWS EXEC_COM LINES TO BE CONDITIONALLY INTERPRETED

▯ USED WITH LOGICAL ACTIVE FUNCTIONS (WHICH RETURN "true" OR "false")

▯ USAGE 1:

```
&if [ACTIVE_FUNCTION {args}]
&then EXEC_COM STATEMENT

&if [exists seg add.pl1]
&then pl1 add.pl1;add
```

▯ USAGE 2:

```
&if [ACTIVE_FUNCTION {args}]
&then EXEC_COM STATEMENT
&else EXEC_COM STATEMENT

&if [equal [wd] [home_dir]]
&then &goto OK
&else &print Assuming working_dir is correct
```

● &

▯ USED TO INDICATE A COMMENT LINE

▯ USAGE: & text

```
& THIS EXEC_COM DETERMINES THE USER'S Project_id
& AND THE TIME OF DAY IN ORDER TO ......
```

CONTROL STATEMENTS

- ● &quit

  - CAUSES EXECUTION OF THE EXEC_COM TO HALT (DEFAULT AT END OF SEGMENT)

  - USAGE: &quit

- ● &attach

  - CAUSES SUBSEQUENT COMMANDS WHICH NORMALLY TAKE THEIR INPUT FROM THE TERMINAL TO TAKE THEIR INPUT FROM THE EXEC_COM SEGMENT

  - USAGE: &attach ⇒ once per exec com segment

  - EXAMPLES:

qedx REQUESTS ARE READ FROM TERMINAL

```
&command_line off
cwd >udd>FED>May
qedx
cwd >udd>FED>Kerr
        .
        .
        .
```

qedx REQUESTS ARE READ FROM EXEC_COM

```
&command_line off
&attach
cwd >udd>FED>May
qedx
r seg_1
1,$s/once/twice/w
q
cwd >udd>FED>Kerr
        .
        .
```

- &detach

  - USED TO REVERT &attach

  - CAUSES SUBSEQUENT COMMAND WHICH NORMALLY TAKE THEIR INPUT FROM THE TERMINAL TO CONTINUE TO TAKE THEIR INPUT FROM THE TERMINAL

  - THE DEFAULT WHEN ENTERING AN EXEC_COM

  - USAGE: &detach

- &input_line off

  - SUPPRESSES THE PRINTING OF SUBSEQUENT INPUT LINES (SUCH AS REQUEST LINES)

  - USAGE: &input_line off

- &input_line on

  - CAUSE SUBSEQUENT INPUT LINES TO BE PRINTED (DEFAULT)

CONTROL STATEMENTS

- &ready on

  - CAUSES THE INVOCATION OF THE USER'S READY PROCEDURE AFTER THE EXECUTION OF EACH COMMAND LINE WITHIN THE EXEC_COM

- &ready off

  - CAUSES THE USER'S READY PROCEDURE NOT TO BE INVOKED (DEFAULT) WITHIN THE EXEC_COM

- exec_com DEFAULTS:

  &command_line on

  &input_line on

  &ready off

  &detach

EXEC COM EXAMPLES

- EXAMPLE 1: LET pl1_pr.ec CONTAIN:

pl1_pr.ec

```
ind &1.pl1
pl1 &1.pl1 -map
dp &1.list

ind &2.pl1
pl1 &2.pl1 -map
dp &2.list
```

- ▯ LET THE COMMAND LINE BE: ec pl1_pr.ec add sub

- ▯ LET THE COMMAND LINE BE: ec pl1_pr.ec sub

- EXAMPLE 2: LET pl1_pr.ec CONTAIN:

pl1_pr.ec

```
&command_line off
&if [nless &n 1] &then &quit

ind &1.pl1
pl1 &1.pl1 -map
dp &1.list
&if [nequal &n 1] &then &quit

ind &2.pl1
pl1 &2.pl1 -map
dp &2.list
&quit
```

numerically less

numerically equal

- EXAMPLE 3: LET pl1_pr.ec HAVE AN ADD NAME OF pl1_pr_.ec AND CONTAIN:

pl1_pr.ec
pl1_pr_.ec

```
&goto &ec_name
&label pl1_pr
&command_line off
&print Beginning &ec_name exec_com
&if [ngreater &n 0] &then &goto pl1_pr_
&print Usage is:  ec &ec_name.ec paths
&quit

&label pl1_pr_
dl &1 -brief
ind &1.pl1
pl1 &1.pl1 -map
& CHECK FOR A SUCCESSFUL COMPILE.  (WAS OBJECT CREATED).
&if [exists segment &1] &then dp &1.list

&if [ngreater &n 1] &then ec &ec_dir>pl1_pr_.ec &2 &3 &4
&quit
```

ADDITIONAL EXAMPLES OF EXEC_COM CONTROL STATEMENTS:

```
&if [equal [day_name] Monday] &then .....

&if [query "Do you really ...?"] &then ........

&if [equal all [response "How many do you want?"] ] &then .....

&if [equal TSmith.FED [last_message_sender] ] &then .......

&if [equal S [substr [user name] 1 1] ] &then ........

&if [equal 0 [min &1 &2 &3 &4] ] &then ........

&if [or  [equal ....   ....]  [less ....   ....] ] &then ......

&if [nless 0 [index "&1 &2 &3 &4" -all] ] &then .......

&if [equal 0 [mod &n 2] ] &then .......
```

AREAS FOR ADDITIONAL STUDY

- ADDITIONAL DOCUMENTATION OF exec_com FACILITIES:
  - MPM COMMANDS AND ACTIVE FUNCTIONS (AG92)
  - help exec_com

- STUDY TOPICS
  - ARGUMENT SUBSTITUTION
    - &qi, &ri (QUOTING AND REQUOTING OF ARGUMENTS)
    - &fi (THE ARGUMENT STRING STARTING WITH THE iTH ARGUMENT)
    - &qfi, &rfi (QUOTING AND REQUOTING OF ARGUMENT STRINGS)
  - CONTROL STATEMENTS
    - &print CONTROL STRINGS: ^/, ^3/, ^-, ^4-, ^¦, ^2¦, ^^

YOU ARE NOW READY FOR WORKSHOP #7

START_UP.CC.

~~&.COMMAND-line off~~

Can be

&if [equal &1 login]
&then
&else &goto label 1

This page has intentionally
been left blank.

TOPIC XVII

ABSENTEE USAGE

This page has intentionally
been left blank.

<u>WHAT IS ABSENTEE USAGE</u>

- ABSENTEE FACILITY

    - A FACILITY FOR RUNNING BACKGROUND JOBS (i.e. BATCH JOBS)

    - GIVES USERS THE ABILITY TO EXECUTE LARGE JOBS WITHOUT WAITING AT THE TERMINAL WHILE THE JOB IS IN PROGRESS

    - CHARGES FOR ABSENTEE USAGE IS USUALLY LOWER THAN CHARGES FOR INTERACTIVE USAGE

    - A USER MAY RUN MANY ABSENTEE JOBS AT ONCE, BUT IS SUBJECT TO THE CURRENT SYSTEM LIMIT ON THE NUMBER OF ABSENTEE JOBS

    - LANGUAGE FOR ABSENTEE USAGE IS <u>IDENTICAL</u> TO THE INTERACTIVE COMMAND LANGUAGE

- PROCESS

    - A PROGRAM CREATED FOR THE USER AT LOG IN, AND DESTROYED AT LOG OUT

    - AN ACTIVE AGENT WHICH DOES WORK FOR THE USER

    - LIKE A PRIVATE COMPUTER, WORKING IN ITS OWN MEMORY UNDER THE CONTROL OF THE USER

WHAT IS ABSENTEE USAGE

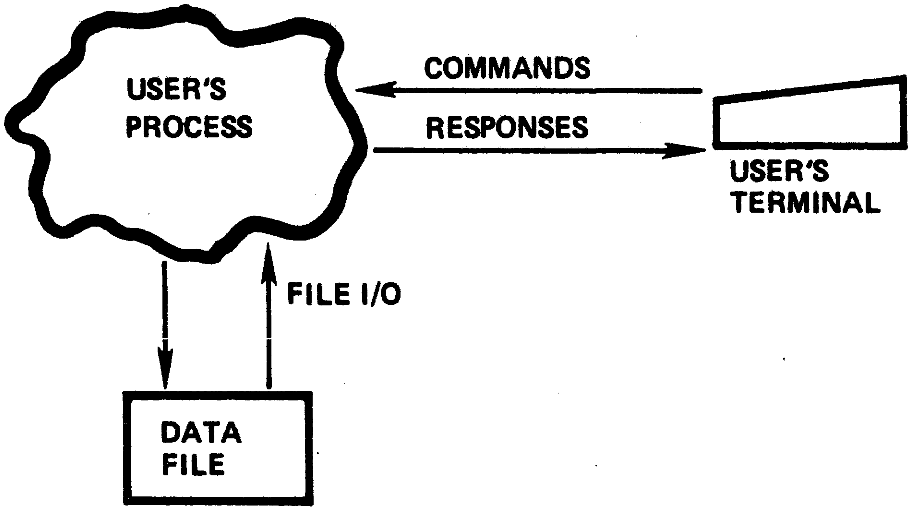
- INTERACTIVE USAGE
  - USING MULTICS INTERACTIVELY VIA A TERMINAL
  - USER'S PROCESS INTERACTS WITH THE USER
- ABSENTEE USAGE
  - USING MULTICS WHILE ABSENT FROM A TERMINAL
  - USER'S PROCESS "INTERACTS" WITH AN INTERACTIVE SCRIPT
  - AN ABSENTEE JOB IS A "PLANNED" INTERACTIVE TERMINAL SESSION

WHAT IS ABSENTEE USAGE

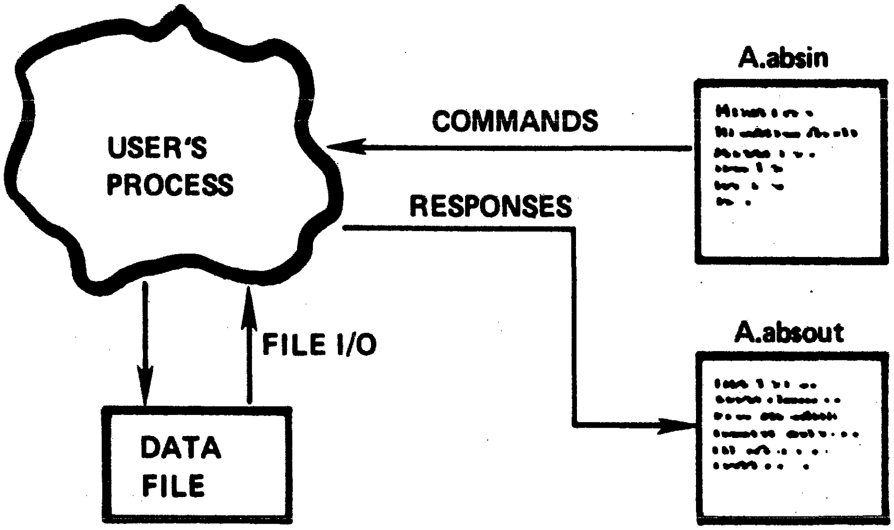
- ABSENTEE MECHANISM
  - USER CREATES AN ABSENTEE INPUT SEGMENT CONTAINING COMMANDS TO BE EXECUTED
  - THE ABSENTEE INPUT SEGMENT
    - MUST HAVE SUFFIX OF absin
    - CONTAINS A PLANNED INTERACTIVE SCRIPT, INCLUDING PRESET ANSWERS TO ANTICIPATED QUESTIONS
    - HAS THE SAME SYNTAX AS EXEC_COM SEGMENTS
  - USER REQUESTS THE EXECUTION OF THE ABSENTEE INPUT SEGMENT VIA THE enter_abs_request COMMAND
  - EXECUTION MAY BE DEFERRED UNTIL A SPECIFIED DATE AND TIME
  - THE ABSENTEE REQUEST IS QUEUED AND RUN AS BACKGROUND TO THE NORMAL INTERACTIVE WORKLOAD
  - ALL OUTPUT NORMALLY DIRECTED TO THE TERMINAL IS REDIRECTED TO A SEGMENT HAVING THE SAME NAME AS THE ABSENTEE INPUT SEGMENT EXCEPT THE SUFFIX IS absout

WHAT IS ABSENTEE USAGE

INTERACTIVE USAGE



ABSENTEE USAGE



WHAT IS ABSENTEE USAGE

- THE ABSENTEE PROCESS WHICH RUNS THE JOB:

  - LOGS INTO THE SYSTEM AS DOES ANY USER (EXCEPT FOR PASSWORD AUTHENTICATION)

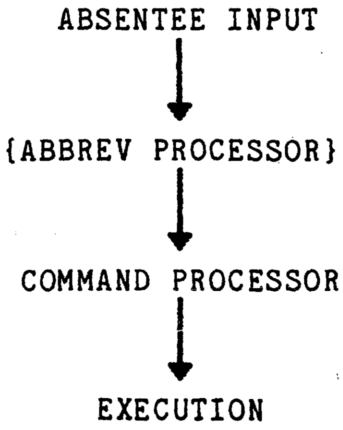
  - EXECUTES THE ABSENT USER'S start_up.ec

  - BEGINS WORKING IN THE ABSENT USER'S HOME DIRECTORY

  - TAKES ITS COMMANDS FROM THE ABSENTEE INPUT SEGMENT

  - APPEARS (AND IS) ANOTHER USER HAVING THE ABSENT USER'S User_id

- COMMAND LINE FLOW



ABSENTEE INPUT

{ABBREV PROCESSOR}

COMMAND PROCESSOR

EXECUTION

- NOTES ON ABSENTEE USAGE

  - start_up.ec SHOULD BE MODIFIED TO RESPOND DIFFERENTLY TO AN ABSENTEE LOG IN

    - THE SYSTEM EXECUTES A USER'S start_up.ec WITH ONE OF THE FOLLOWING COMMAND LINES:

```
ec  start_up.ec  login     interactive
ec  start_up.ec  login     absentee
ec  start_up.ec  new_proc  interactive
ec  start_up.ec  new_proc  absentee
```

    - THE USER MAY QUERY THE ARGUMENTS USING exec_com LINES SIMILAR TO THE FOLLOWING:

```
&if [equal &1 login] &then ...

&if [equal &2 absentee] &then ...
```

  - THE ABSENTEE INPUT SEGMENT MUST CONTAIN change_wdir COMMAND IF WORKING DIRECTORY IS TO BE OTHER THAN HOME DIRECTORY

  - THE ABSENTEE INPUT SEGMENT OFTEN (BUT NEED NOT) HAS logout AS FINAL COMMAND

  - &attach HAS NO SIGNIFICANCE IN AN ABSENTEE INPUT SEGMENT SINCE ALL USER INPUT IS ATTACHED TO absin FILE

  - IF THE ABSENTEE JOB CANNOT BE RUN, OR IF IT TERMINATES ABNORMALLY, THE SUBMITTER WILL RECEIVE A MESSAGE FROM THE SYSTEM

ABSENTEE COMMANDS

- enter_abs_request, ear (BATCH NEVER LOOKED LIKE THIS!)

  - QUEUES A REQUEST FOR THE CREATION OF AN ABSENTEE PROCESS

  - USAGE:

```
enter_abs_request path {-control_args}

ear weird.absin

ear weird.absin -restart

ear trans.absin -time "Friday 9pm"

ear trans.absin -queue 1 -arguments add sum
```

- list_abs_request, lar

  - PRINTS INFORMATION ABOUT ABSENTEE REQUESTS

  - USAGE:

```
list_abs_request {-control_args}

lar

lar -q 1

lar -all
```

ABSENTEE COMMANDS

- cancel_abs_request, car

  - CANCELS AN ABSENTEE REQUEST WHICH IS STILL QUEUED

  - USAGE:

    ```
    cancel_abs_request request_id {-control_args}
    car weird.absin
    car trans.absin -q 1
    car -id 202008
    ```

- answer

  - PROVIDES PRESET ANSWER(S) TO QUESTION(S) ASKED BY A COMMAND

  - PRIMARILY FOR USE IN ABSENTEE INPUT SEGMENTS

  - THE ANSWER IS PROVIDED AN UNLIMITED NUMBER OF TIMES UNLESS LIMITED BY THE -times CONTROL ARGUMENT

  - UNUSED ANSWERS ARE IGNORED

  - USAGE:

    ```
    answer ans {-control_args} command_line
    answer yes delete_dir Dir_A
    answer yes -times 2 help send_message
    answer rest help trace
    answer no answer yes -times 2 help trace
    ```

AREAS FOR ADDITIONAL STUDY

- ADDITIONAL DOCUMENTATION
  - MPM COMMANDS AND ACTIVE FUNCTIONS (AG92)
  - help enter_abs_request
- STUDY TOPICS
  - CONTROL ARGUMENTS
    - -output_file, -limit, -brief
  - ABORTING OF AN ACTIVE ABSENTEE PROCESS (THERE IS NO "GRACEFUL" WAY)
    - DELETE ACCESS TO THE absin SEGMENT
    - DELETE ACCESS TO THE absout SEGMENT
    - CALL OPERATOR
  - SELF PERPETUATING ABSENTEE PROCESSES

This page has intentionally

been left blank

TOPIC XVIII

SOFTWARE CONVENTIONS

This page has intentionally
been left blank.

COMMAND LANGUAGE SPECIAL SYMBOLS

- . (PERIOD)

  - SEPARATES COMPONENTS OF AN ENTRYNAME, STAR NAME, OR ACCESS IDENTIFIER

    ```
    A_test.alpha.cobol

    *.*.pl1

    Student_04.F01.m
    ```

- _ (UNDERSCORE)

  - SIMULATES A SPACE FOR READABILITY IN ENTRYNAMES

    ```
    ran_num_gen.basic
    ```

  - IS NOT AN ABBREV BREAK CHARACTER

  - ALL SYSTEM SUBROUTINES END IN _

- > (GREATER-THAN)

  - DENOTES HIERARCHY LEVEL (TOP DOWN)

  - SEPARATES THE ENTRYNAMES IN A PATHNAME

    ```
    >udd>F01>Student_01>tools>my_editor.pl1
    ```

COMMAND LANGUAGE SPECIAL SYMBOLS

● < (LESS-THAN)

▯ INDICATES ONE LEVEL BACK UP IN THE HIERARCHY

▯ EXAMPLES ASSUMING WORKING DIRECTORY IS >udd>FED>LJones>tools

```
cwd <
cwd >udd>FED>LJones

cwd <<
cwd >udd>FED

pr <start_up.ec
pr >udd>FED>LJones>start_up.ec

pr <<TSmith>tools>random
pr >udd>FED>TSmith>tools>random
```

● * (STAR OR ASTERISK)

▯ MATCHES ANY COMPONENT OF AN ENTRYNAME OR ACCESS IDENTIFIER (STAR CONVENTION)

```
list seg_1.*.*

list seg_1.**

sa *.* r LJones.*.a
```

COMMAND LANGUAGE SPECIAL SYMBOLS

- ? (QUESTION MARK)

  ▯ MATCHES ANY LETTER OF A COMPONENT OF AN ENTRYNAME (STAR CONVENTION)

```
list s???.old.pl1
list s??.**
```

- = (EQUAL)

  ▯ IS REPLACED BY THE CORRESPONDING COMPONENT OF AN ENTRYNAME (EQUAL CONVENTION)

  ▯ * (STAR) AND = (EQUAL) ARE SYMMETRICAL IN MEANING

```
rename  random.gen.pl1  ordered.=.=
rename  random.gen.pl1  ordered.gen.pl1

rename  random.data.base  =.=
rename  random.data.base  random.data

rename  beta  my_=.old
rename  beta  my_beta.old

rename  *.new.pl1     =.old.=
rename  beta.new.pl1  beta.old.pl1

add_name  **.ec      ==.absin
add_name  pl1_pr.ec  pl1_pr.absin
```

COMMAND LANGUAGE SPECIAL SYMBOLS

- % (PERCENT)

  - IS REPLACED BY THE CORRESPONDING CHARACTER OF A COMPONENT NAME (EQUAL CONVENTION)

  - ? (QUESTION MARK) AND % (PERCENT) ARE SYMMETRICAL IN MEANING

```
rename  my_old_ed.pl1  %%%new%%%.=
rename  my_old_ed.pl1  my_new_ed.pl1

rename  ????*.data     %%%.=
rename  alpha_2.data   alp.data
```

- ; (SEMI-COLON)

  - USED TO SEPARATE MULTIPLE COMMANDS IN A COMMAND LINE

```
cwd dir_a; list;   cwd dir_b;ls
```

- - (MINUS SIGN)

  - IDENTIFIES CONTROL ARGUMENTS

```
list -directory

dprint -he TSmith -ds MS106
```

COMMAND LANGUAGE SPECIAL SYMBOLS

- ( ) (PARENTHESES)

  - CAUSES COMMAND ITERATION

```
print (A B add).pl1

print A.pl1
print B.pl1
print add.pl1
```

```
sm Student_0(1 3 5).F01 Return to the classroom.

sm Student_01.F01 Return to the classroom.
sm Student_03.F01 Return to the classroom.
sm Student_05.F01 Return to the classroom.
```

```
rename (A B add).pl1 (a b Add).pl1

rename A.pl1 a.pl1
rename B.pl1 b.pl1
rename add.pl1 Add.pl1
```

```
create_dir (new>(first second) old>third)

create_dir new>first
create_dir new>second
create_dir old>third
```

COMMAND LANGUAGE SPECIAL SYMBOLS

- [ ] (BRACKETS)

  - DELIMITS ACTIVE STRINGS. (EVALUATED BY ACTIVE FUNCTIONS)

    ```
    sm [last_message_sender] Thanks!
    ```

- $ (DOLLAR SIGN)

  - SEPARATES THE ENTRYNAME OF AN OBJECT SEGMENT FROM THE ENTRY POINT NAME WITHIN THE OBJECT SEGMENT

  - DEFAULT ENTRY POINT NAME IS THE ENTRYNAME

    ```
    add
    add$add

    add$max
    add$max
    ```

- " (QUOTES)

  - SUPPRESSES THE SPECIAL MEANING OF COMMAND LANGUAGE SPECIAL SYMBOLS

    ```
    rename ";" foo      (segment name is: ;)

    delete "A B"        (segment name is: A B)

    delete "A""B"       (segment name is: A"B)

    delete A""B         (segment name is: AB)
    ```

SEGMENT NAME SUFFIXES

absin	ABSENTEE INPUT SEGMENT FOR THE enter_abs_request COMMAND
absout	ABSENTEE OUTPUT SEGMENT CREATED VIA THE enter_abs_request COMMAND
acs	ACCESS CONTROL SEGMENT USEFUL TO THE IO DAEMON
alm	ALM SOURCE SEGMENT
apl	APL WORKSPACE SEGMENT
archive	SEGMENT MANIPULATED BY THE archive COMMAND
basic	BASIC SOURCE SEGMENT
bind	BINDFILE FOR THE bind COMMAND
breaks	BREAK SEGMENT USED BY THE debug COMMAND
cmdb	MRDS DATA MODEL SOURCE SEGMENT FOR THE create_mrds_db COMMAND
cmdsm	MRDS DATA SUB-MODEL SOURCE SEGMENT FOR THE create_mrds_dsm COMMAND
cobol	COBOL SOURCE SEGMENT
code	ENCIPHERED SEGMENT CREATED BY THE encode COMMAND
compin	INPUT SEGMENT TO THE compose COMMAND
compout	OUTPUT SEGMENT CREATED BY THE compose COMMAND
dict	WORDPRO DICTIONARY SEGMENT CREATED BY THE add_dict_words COMMAND
dsm	MRDS DATA SUB-MODEL SEGMENT CREATED BY THE create_mrds_dsm COMMAND
ec	COMMAND FILE FOR THE exec_com COMMAND
fortran	FORTRAN SOURCE SEGMENT
gcos	SEGMENT IN GCOS STANDARD SYSTEM FORMAT FOR THE gcos COMMAND
info	SEGMENT FORMATTED FOR USE BY THE help COMMAND
iodt	I/O DEVICE TABLE

SEGMENT NAME SUFFIXES

linus	LINUS MACRO SEGMENT FOR THE LINUS invoke REQUEST
list	LISTING SEGMENT CREATED BY A LANGUAGE PROCESSOR
lister	LISTER DATA FILE (STRUCTURED) FOR THE process_list COMMAND
listform	LISTER FORMS DESCRIPTOR SEGMENT FOR THE process_list COMMAND
listin	LISTER DATA FILE (ASCII) FOR THE create_list COMMAND
mbx	MAILBOX SEGMENT FOR THE mail COMMAND
memo	DATA SEGMENT FOR THE memo COMMAND
motd	DATA SEGMENT FOR THE print_motd COMMAND
mrpg	SOURCE SEGMENT FOR THE MULTICS REPORT PROGRAM GENERATOR'S mrpg COMMAND
ms	MESSAGE SEGMENTS FOR RJE
pl1	PL/1 SOURCE SEGMENT
profile	DICTIONARY SEGMENT FOR THE ABBREV PROCESSOR
qedx	MACRO SEGMENT FOR THE qedx TEXT EDITOR
symbols	SPEEDTYPE SYMBOL DICTIONARY CREATED BY THE add_symbol COMMAND
wl	WORDPRO WORDLIST SEGMENT CREATED BY THE create_wordlist COMMAND

SUFFIX CONVENTION

- ALL COMMANDS WHICH ONLY WORK ON SEGMENTS HAVING A GIVEN SUFFIX WILL APPEND THAT SUFFIX TO ENTRYNAMES TYPED WITHOUT THE SUFFIX

```
archive a Field_work summary_report
archive a Field_work.archive summary_report


basic  ran_num_gen -list
basic  ran_num_gen.basic -list


cancel_abs_request  weird
cancel_abs_request  weird.absin


cobol A_alpha
cobol A_alpha.cobol


enter_abs_request weird -tm 6pm
enter_abs_request weird.absin -tm 6pm


exec_com A_create add sum
exec_com A_create.ec add sum


fortran array_dot
fortran array_dot.fortran


indent add
indent add.pl1


pl1 add -optimize
pl1 add.pl1 -optimize
```

CANONICAL FORM

- MOTIVATION

  - HOW WAS THE FOLLOWING LINE TYPED?

    The only one!

    B - backspace

    The onlyBBBB____ one!

    - or -

    standard form → The oB_nB_lB_yB_ one!

    - or -

    The _Bo_Bn_Bl_By one!

  - COMPARISON OF TWO INTERNAL LINES SHOULD BE BASED ON RESULTING PRINTED IMAGE

  - THEREFORE, ALL INPUT SHOULD BE CONVERTED TO A STANDARD (CANONICAL) FORM

  - MULTICS AUTOMATICALLY TRANSLATES ALL TERMINAL INPUT TO CANONICAL FORM (UNLESS THE USER REQUESTS OTHERWISE)

  - SHOULD THE USER NOTICE THAT SOME TEXT IS NOT STORED IN THE SAME SEQUENCE AS IT WAS TYPED, CANONICALIZATION SHOULD BE EXPECTED (FOR EXAMPLE, DURING CERTAIN TEXT EDITING OPERATIONS)

CANONICAL FORM

- CANONICAL FORM

  - OVERSTRIKES ARE STORED IN ASCENDING ASCII ORDER, SEPARATED BY THE BACKSPACE CHARACTER

  - EXAMPLES: (B = BACKSPACE, C = CARRIAGE RETURN, N = NEWLINE)

```
TYPIST:              >B<B_      _B<B>
TYPED LINE:          x̲          x̲
CANONICAL FORM:      <B>B_      <B>B_

TYPIST:              The onlyBBBB____
TYPED LINE:          The only
CANONICAL FORM:      The _Bo_Bn_Bl_By

TYPIST:              We see no prob BlemC__N
TYPED LINE:          We see no problem
CANONICAL FORM:      WB__Be see no problem
```

SET TTY COMMAND

- TERMINAL TYPES

  - MULTICS ATTEMPTS TO RECOGNIZE TERMINAL TYPE AT DIAL-UP TIME

  - EACH TERMINAL TYPE HAS A SET OF DEFAULT I/O MODES

  - TERMINAL I/O MODES AFFECT CHARACTER CONVERSION, DELAY TIMES, AND COMMUNICATION LINE CONTROL

- set_tty, stty

  - MODIFIES TERMINAL TYPE AND/OR MODES ASSOCIATED WITH TERMINAL I/O

  - USAGE:

```
set_tty -control_args

stty -terminal_type TN300

stty -ttp ARDS

stty -ttp ti745       or ti 700

stty -ttp rosy        honeywell terminal

stty -modes lfecho,fulldpx

stty -print           displays characteristics of terminal

stty -reset

stty -delay 1,0,0,0,0,0      μsec units of delay on terminal
                             associated with tab, return

stty -edit AB         changing lines delete &
                      character delete characters.
```

SET TTY COMMAND

^ – "not" or "opposite" or "off"

- TERMINAL I/O MODES

  - can, ^can

    - PERFORMS STANDARD CANONICALIZATION (DEFAULT IS ON)

  - capo, ^capo

    - OUTPUTS ALL LOWERCASE LETTERS IN UPPERCASE (DEFAULT IS OFF)

  - lfecho, ^lfecho

    - ECHOES AND INSERTS A LINE FEED ~~IN THE USER'S~~ INPUT STREAM WHEN A CARRIAGE RETURN IS TYPED (DEFAULT IS OFF)

  - crecho, ^crecho

    - ECHOES A CARRIAGE RETURN WHEN A LINE FEED IS TYPED (DEFAULT IS OFF)

  - fulldpx, ^fulldpx

    - ALLOWS TERMINAL TO RECEIVE AND TRANSMIT SIMULTANEOUSLY (DEFAULT IS OFF)

  - echoplex, ^echoplex

    - ECHOES ALL CHARACTERS TYPED ON THE TERMINAL (DEFAULT IS OFF)

SET TTY COMMAND

- edited, ^edited

  - SUPPRESSES PRINTING OF UNPRINTABLE CHARACTERS - LIKE \014 (DEFAULT IS OFF)

- tab, ^tab

  preset to 11, 21, 31, 41, etc.

  - INSERTS TABS IN OUTPUT IN PLACE OF SPACES WHEN APPROPRIATE (DEFAULT IS OFF)

- tabecho, ^tabecho

  - ECHOES THE APPROPRIATE NUMBER OF SPACES WHEN A HORIZONTAL TAB IS TYPED (DEFAULT IS OFF)

- polite, ^polite

  - DOES NOT SEND OUTPUT TO THE TERMINAL UNTIL THE CARRIAGE IS AT THE LEFT MARGIN (DEFAULT IS OFF)

- replay, ^replay

  - REPRINTS ANY PARTIAL INPUT LINE THAT IS INTERRUPTED BY OUTPUT. (DEFAULT IS OFF)

YOU ARE NOW READY FOR WORKSHOP #8

TOPIC XIX

ADDITIONAL COMMANDS

This page has intentionally
been left blank.

ACCESS TO THE SYSTEM

- dial, d

  - CONNECTS AN ADDITIONAL TERMINAL TO AN EXISTING PROCESS

  - ANSWERING SERVICE SEARCHES FOR A LOGGED IN PROCESS HAVING THE SPECIFIED Person_id AND Project_id THAT IS ACCEPTING DIAL-UPS

  - SEE THE dial_manager_ SUBROUTINE DESCRIPTION (AK92) FOR MORE INFORMATION

  - USAGE:
    ```
    dial dial_id Person_id.Project_id

    d 411 TransProc.HSD
    ```

- enter, e
enterp, ep

  - CONNECTS AN ANONYMOUS USER TO THE SYSTEM

  - THE enter REQUEST DOES NOT ASK FOR A PASSWORD WHEREAS enterp DOES

  - USAGE:
    ```
    enter {anonymous_name} Project_id

    enterp {anonymous_name} Project_id

    e JDoe FED

    ep JDoe FED
    ```

ACCESS TO THE SYSTEM

- login, l

  - USED TO GAIN ACCESS TO THE SYSTEM

  - USAGE:

```
login Person_id {Project_id} {control_args}

login TSmith

login TSmith F01 -cpw -ns -modes lfecho

login TSmith F01 -gpw -cdp -ring 5

login TSmith -force -hd >udd>FED>Kerr
```

- logout

  - TERMINATES A USER SESSION

  - USAGE:

```
logout {-control_args}

logout -bf

logout -hold
```

ACCESS TO THE SYSTEM

- MAP
  - TELLS MULTICS THAT THE TERMINAL IS AN UPPERCASE-ONLY TERMINAL
  - MUST BE INVOKED BEFORE THE ACCESS REQUEST (e.g., login) (PREACCESS ONLY)
  - ALL INPUT IS MAPPED TO LOWERCASE EXCEPT FOR CHARACTERS PRECEDED BY A \ (BACKSLASH)
  - USAGE: MAP

- 029 AND 963
  - TELLS MULTICS THAT THE TERMINAL IS AN EBCDIC OR IBM 2741
  - MUST BE INVOKED BEFORE THE ACCESS REQUEST (e.g., login)
  - USAGE: 029 -OR- 963

ACCESS TO THE SYSTEM

- hello

  - CAUSES THE GREETING MESSAGE TO BE REPEATED

  - MUST BE INVOKED BEFORE THE ACCESS REQUEST (e.g. login)

  - USEFUL IF THE GREETING WAS GARBLED (AS WOULD OCCUR WITH AN EBCDIC TERMINAL OR BECAUSE OF LINE NOISE)

  - USAGE: hello

STORAGE SYSTEM, SEGMENT CONTENTS

- edm

  - eventually taken off system
  - not as powerful as qedx

  ▯ INVOKES A SIMPLE, INEXPENSIVE TEXT EDITOR (A SUBSYSTEM)

  ▯ USAGE:

  ```
  edm {path}

  edm

  edm add.pl1
  ```

- qedx, qx

  ▯ INVOKES A SOPHISTICATED TEXT EDITOR (A SUBSYSTEM) HAVING MACRO CAPABILITIES

  ▯ USAGE 1:
  (MANUAL EDITING)

  ```
  qedx

  qx
  ```

  ▯ USAGE 2:
  (MACRO EDITING)

  ```
  qedx path {optional_args}

  qx conv_ft.qedx Random.fortran
  ```

- archive, ac (LIKE SARDINES)

  - COMBINES AN ARBITRARY NUMBER OF SEPARATE SEGMENTS INTO ONE SEGMENT (THE ARCHIVE SEGMENT)

  - A MEANS OF ORGANIZING SEGMENTS (IDENTITY OF EACH SEGMENT IS PRESERVED)

  - SAVES PHYSICAL SPACE BY COMPACTING SEGMENTS TOGETHER

  - THE ARCHIVE SEGMENT MUST HAVE A SUFFIX OF archive

  - CONSTITUENT SEGMENTS ARE CALLED COMPONENTS OF THE ARCHIVE SEGMENT

  - THE COMMAND IS ALSO USED TO EXTRACT COMPONENTS FROM THE ARCHIVE AND RETURN THEM TO INDIVIDUAL STORAGE SYSTEM SEGMENTS

  - USAGE:

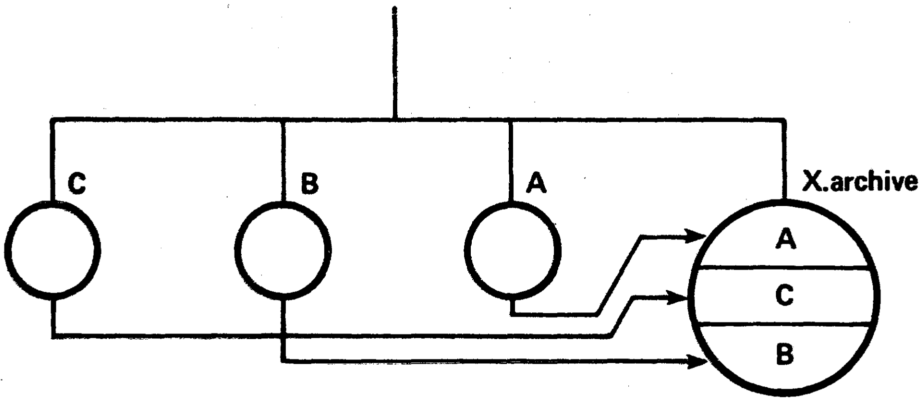
```
                1      2    3       4
USAGE:   archive key path components

         ac r bound_pl1_prgms.s add.pl1 sub.pl1

         ac rd bound_pl1_prgms.s [segs **.pl1]

         ac x bound_pl1_prgms.s.archive sub.pl1

         ac t bound_pl1_prgms.s

         ac d bound_pl1_prgms.s add.pl1
```

STORAGE SYSTEM, SEGMENT CONTENTS

KEY

- TABLE OF CONTENTS OPERATION (t, tl, tb, tlb)
- APPEND OPERATION (a, ad, adf, ca, cad, cadf)
- REPLACE OPERATION (r, rd, rdf, cr, crd, crdf)
- UPDATE OPERATION (u, ud, udf, cu, cud, cudf)
- DELETE OPERATION (d, cd)
- EXTRACT OPERATION (x, xf)

- Cannot perform 'normal' operations on a segment while it is ~~in~~ the archives.



ac a X A C B

- ● bind, bd (... AND THEY SHALL BECOME AS ONE)

  - ▯ PRODUCES A SINGLE BOUND (PRE-LINKED) OBJECT SEGMENT FROM ONE OR MORE UNBOUND OBJECT SEGMENTS

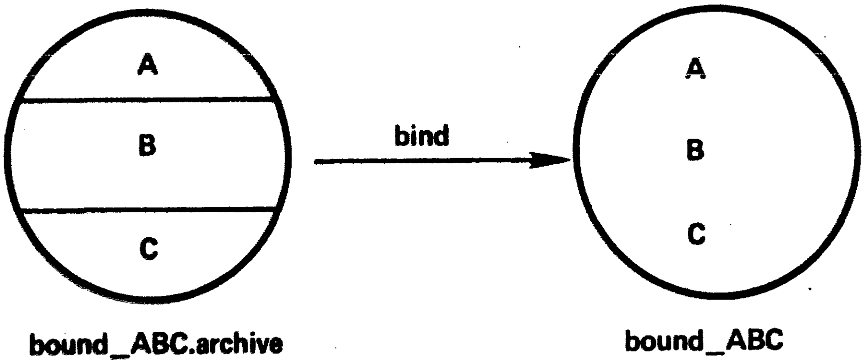
  - ▯ THE OBJECT SEGMENTS TO BE BOUND MUST FIRST BE PLACED IN AN ARCHIVE SEGMENT

  - ▯ THE CONTENTS OF THE ARCHIVE SEGMENT ARE THEN "BOUND" TOGETHER

  - ▯ USAGE:

    ```
    bind paths {-control_args}

    bd editor_mods.archive
    ```

cuts down overhead



```
bd bound_ABC
```

STORAGE SYSTEM, SEGMENT CONTENTS

- compare_ascii, cpa

  - COMPARES TWO ASCII SEGMENTS AND PRINTS THE CHANGES MADE TO THE CONTENTS OF PATH1 TO YIELD THE CONTENTS OF PATH2

  - THE AMOUNT OF LOOK AHEAD FOR RE-SYNCHRONIZING IS DETERMINED BY THE min_chars AND min_lines ARGUMENTS

  - DEFAULT min_chars IS 50, DEFAULT min_lines IS 5

  - USAGE: compare_ascii path1 path2 {min_chars} {min_lines}

    cpa genum.old.pl1 genum.new.pl1

- adjust_bit_count, abc

  - CORRECTS THE BIT COUNT OF A SEGMENT (AN ATTRIBUTE STORED IN THE CONTAINING DIRECTORY) TO REFLECT THE ACTUAL BIT COUNT OF THE SEGMENT

  - USEFUL ON FILES LEFT IN AN INCONSISTENT STATE (BY AN ABORTING PROGRAM, ETC)

  - USAGE: adjust_bit_count path {-control_args}

    abc temp

    abc output_file -lg

STORAGE SYSTEM, MANIPULATION

- link, lk (DON'T CONFUSE WITH DYNAMIC LINKING)

  - CREATES A LINK TO A SPECIFIED SEGMENT OR DIRECTORY

  - THE LINK "LOOKS" LIKE THE REAL THING TO MOST COMMANDS

  - USED TO SAVE PHYSICAL SPACE AND/OR REDIRECT STORAGE SYSTEM ACCESSES

  - FREQUENTLY USED BY A PERSON BELONGING TO SEVERAL PROJECTS TO LINK HIMSELF TO A COMMON MAILBOX, start_up.ec, profile, etc.

  - USAGE:

    ```
    link path1 {path2}

    lk >udd>FED>Kerr>dev>x_sort >udd>FED>Kerr>tools>sort

    lk >udd>FED>Kerr>dev>x_sort sort

    lk >udd>FED>Kerr>dev>x_sort

    lk [home_dir]>([segs[home_dir]>**])
    ```

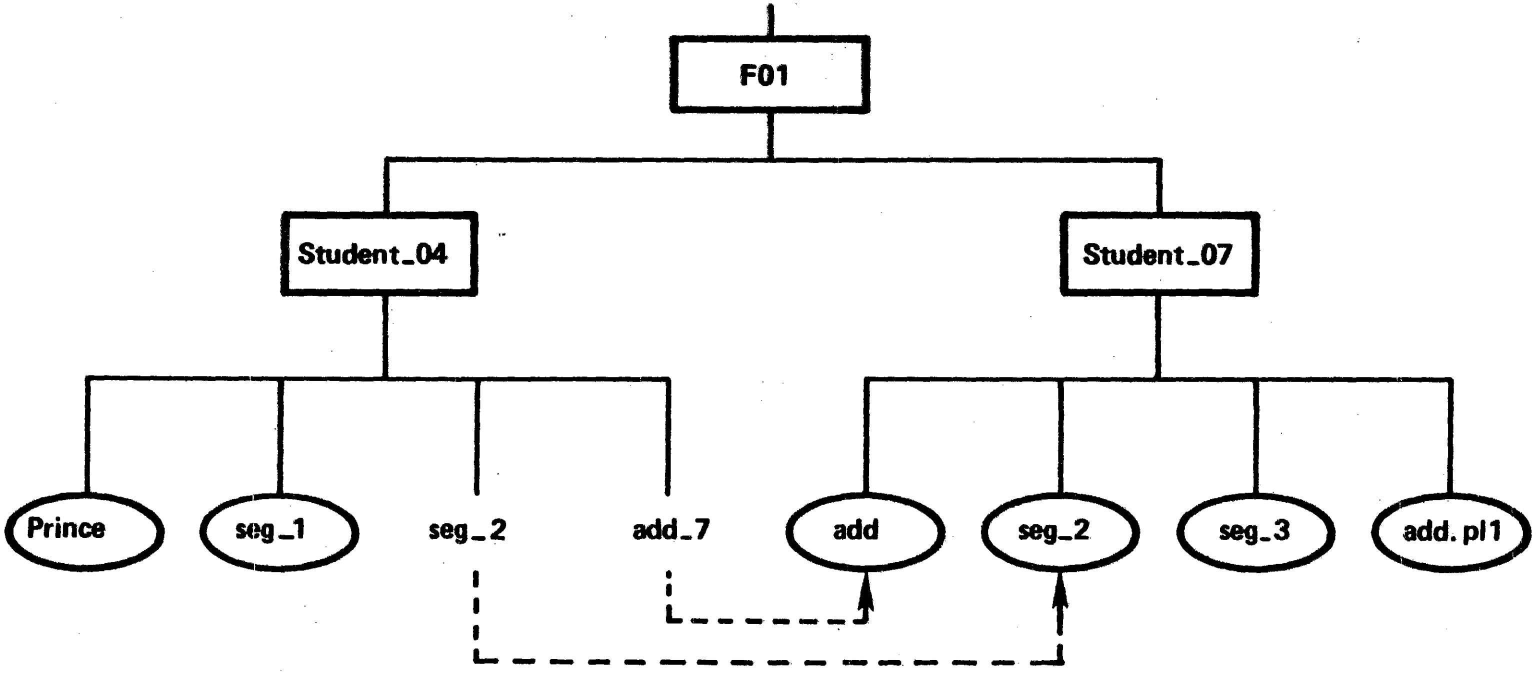
- unlink, ul

  - DELETES THE SPECIFIED LINK ENTRY

  - USAGE:

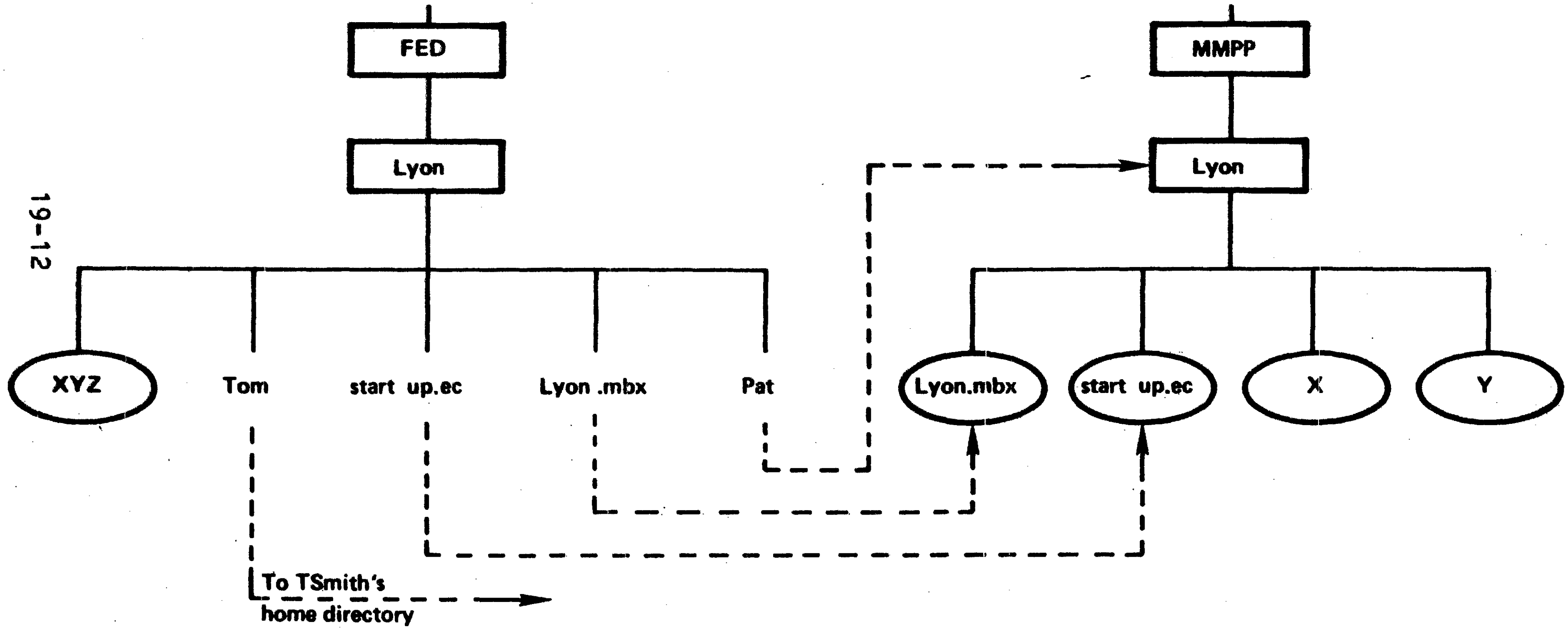
    ```
    unlink paths

    ul sort
    ```

LINKING EXAMPLE 1



```
link    >udd>F01>Student_07>add  add_7
link    >udd>F01>Student_07>seg_2
```

LINKING EXAMPLE 2



```
link   >udd>MMPP>Lyon>(start_up.ec  Lyon.mbx)
link   >udd>MMPP>Lyon   Pat
link   >udd>WOPS>TSmith  Tor
```

FORMATTED OUTPUT FACILITIES

- dump_segment, ds

  - PRINTS A SEGMENT'S CONTENTS IN OCTAL, ASCII, OR BCD

  - USAGE:

```
USAGE:  dump_segment path {first} {n_words} {-control_args}

        ds prince 400 20

        ds add -bcd
```

- sort_seg, ss

  - ORDERS THE CONTENTS OF A SEGMENT ACCORDING TO THE ASCII COLLATING SEQUENCE

  - SEGMENT IS BROKEN DOWN INTO SEPARATE SORT UNITS DELIMITED BY SPECIFIED DELIMITER STRING. SORT UNITS ARE THEN SORTED

  - USAGE:

```
USAGE:  sort_seg path {-control_args}

        ss tel_data.old

        ss tel_data.old -delimiter xx -descending

        ss tel_data.old -sm tel_data.new -unique
```

- cumulative_page_trace, cpt

  - ACCUMULATES PAGE TRACE DATA SO THAT THE TOTAL SET OF PAGES USED FOR A COMMAND OR PROGRAM CAN BE DETERMINED

  - USAGE:

    ```
    cumulative_page_trace command_line {-control_args}

    cpt add -reset

    cpt -print
    ```

- profile

  - PRINTS INFORMATION ABOUT THE EXECUTION OF INDIVIDUAL STATEMENT: WITHIN A PL/I, FORTRAN, OR COBOL PROGRAM

  - PROGRAMS MUST HAVE BEEN COMPILED WITH THE -profile CONTROL ARGUMENT

  - USAGE:

    ```
    profile path {-control_args}

    profile add

    profile add -reset
    ```

DEBUGGING

- trace_stack, ts

  - PRINTS THE USER'S STACK HISTORY - MOST RECENT FIRST

  - USAGE:

    ```
    trace_stack {-control_args}
    ts
    ts -depth 5
    ```

- trace

  - MONITORS CALLS TO SPECIFIED PROCEDURES

  - PROCEDURES MUST HAVE ORIGINATED FROM PL/I OR FORTRAN SOURCE

  - USAGE:

    ```
    trace {-control_args} names
    trace add
    ```

- line_length, ll

  - SETS THE MAXIMUM LENGTH OF OUTPUT LINES

  - WRAP-AROUND, IF IT OCCURS, IS PRECEDED BY "\c"

  - USAGE: line_length maxlength

    ll 118

- ready_off, rdf

  - TURNS OFF THE READY MESSAGE

  - USAGE: ready_off

    rdf

- ready_on, rdn

  - PRINTS A READY MESSAGE AFTER EACH COMMAND LINE HAS BEEN PROCESSED (THE DEFAULT)

  - USAGE: ready_on

    rdn

- **general_ready, gr**

  - ALLOWS USER TO FORMAT THE READY MESSAGE

  - USAGE:

```
USAGE:  general_ready {-control_arg}
        gr -string "DONE MASTER" -set
        gr -string * -call print_messages -set
        gr -string READY -hour : -minute -inc_cost
        gr -control * -set
```

- get_quota, gq

  - RETURNS INFORMATION ABOUT STORAGE QUOTA AND USAGE FOR A SPECIFIED DIRECTORY

  - DOES NOT "get" THE USER ANY MORE QUOTA

  - USAGE:

```
get_quota {paths} {-control_args}

gq dir_A

gq -long
```

- resource_usage, ru

  - PRINTS A REPORT OF RESOURCE CONSUMPTION FOR THE CURRENT BILLING PERIOD

  - USAGE:

```
resource_usage {-control_args}

ru

ru -long
```

ABSENTEE COMPUTATIONS

- fortran_abs, fa

  - SUBMITS AN ABSENTEE REQUEST TO PERFORM FORTRAN COMPILATIONS AND dprint COMPILER'S OUTPUT

  - USAGE:

    ```
    fortran_abs paths {-ft_args} {-dp_args} {-abs_args}

    fa array_dot.fortran

    fa array_dot -map -copy 2
    ```

- pl1_abs, pa

  - SUBMITS AN ABSENTEE REQUEST TO PERFORM PL/I COMPILATIONS AND dprint COMPILER'S OUTPUT

  - USAGE:

    ```
    pl1_abs paths {-pl1_args} {-dp_args} {-abs_args}

    pa add.pl1

    pa add -optimize -queue 1 -hold
    ```

ABSENTEE COMPUTATIONS

- runoff_abs, rfa

  - SUBMITS AN ABSENTEE REQUEST TO PROCESS TEXT SEGMENTS (USING THE RUNOFF COMMAND) AND dprint THE OUTPUT

  - USAGE:

    ```
    runoff_abs path {-rf_args} {-ear_args} {-dp_args} {-abs_

    rfa prince.runoff

    rfa prince -in 10 -tm 8pm -cp 3
    ```

- cobol_abs

  - SUBMITS AN ABSENTEE REQUEST TO PERFORM COBOL COMPILATIONS AND dprint COMPILER'S OUTPUT

  - USAGE:

    ```
    cobol_abs paths {cobol_args} {dp_args} {-abs_args}

    ca add.cobol

    ca add -optimize -queue 1 -hold
    ```

MISCELLANEOUS TOOLS

● calc

- PROVIDE THE USER WITH A CALCULATOR (A SUBSYSTEM)

- ACCEPTS FORTRAN-LIKE EXPRESSIONS

- pi AND e ARE BUILT-IN VARIABLES

- USAGE: calc

```
calc
x=pi * 3.4 ** 2
3.57 * 2**(x * 10.7)/sin (35.7)
q
```

● encode

- ENCIPHERS A SEGMENT'S CONTENT ACCORDING TO A KEY SUPPLIED BY THE USER

- encode ASKS TWO TIMES FOR THE ENCIPHER KEYWORD

- ENCIPHERED SEGMENT IS GIVEN A SUFFIX OF code

- USAGE: encode path1 {path2}

```
encode blacklist
```

MISCELLANEOUS TOOLS

- decode

  - RECONSTRUCTS AN ORIGINAL SEGMENT FROM AN ENCIPHERED SEGMENT IF PROPER KEY IS SUPPLIED

  - decode ASKS FOR THE ENCIPHER KEYWORD

  - USAGE:
    ```
    decode path1 {path2}
    decode blacklist
    ```

- new_proc

  - DESTROYS THE USER'S CURRENT PROCESS AND CREATES A NEW ONE

  - EFFECTIVELY THE SAME AS LOGGING OUT AND LOGGING IN AGAIN

  - ASSUMES THE SAME CONTROL ARGUMENTS THE USER GAVE AT LOG IN

  - USAGE:
    ```
    new_proc {-control_arg}
    new_proc
    ```

YOU ARE NOW READY FOR WORKSHOP #9

TOPIC XX

SOFTWARE OVERVIEW

This page has intentionally
been left blank.

THE OPERATING SYSTEM

- MORE THAN 95% OF THE OBJECT CODE ORIGINATED FROM PL/I SOURCE (1,132,000 LINES)

- LESS THAN 5% OF THE OBJECT CODE ORIGINATED FROM ALM ASSEMBLY CODE (226,000 LINES)

- TOTAL OBJECT CODE OCCUPIES MORE THAN SIX MILLION WORDS OF STORAGE

- HIGHLY STRUCTURED (3300 MODULES)

- CODE IS PURE, RECURSIVE AND RE-ENTRANT

- ON-LINE INSTALLATION OF SYSTEM MODULES

- EXTENSIVE ON-LINE METERING AND TUNING FACILITIES

SYSTEM SECURITY

- "...Multics is properly characterized as the most secure commercial operating system available."

  Prof. Peter J. Denning
  Computer Science Dept.
  Purdue University
  (Computing Europe, July 29, 1976)

- Multics security architecture is superior to any other commercially available system (by 2 to 1 ratio).

  Mitre Corporation Study for
  U.S. Air Force - Sept., 1975
  (USDC Order No: AD-A009221)

- SYSTEM ACCESS: USER AUTHENTICATION

  - USER AUTHENTICATION REQUIRED TO LOG IN

  - PASSWORD IS DETERMINED AND CHANGED BY USER, AT WILL (RANDOM PASSWORD GENERATION IS AVAILABLE IF DESIRED)

  - ONLY A NON-REVERSIBLE RESIDUE OF EACH PASSWORD IS STORED

  - NOTIFICATION OF INCORRECT PASSWORD USAGE

  - LAST LOG IN NOTIFICATION

SYSTEM SECURITY

- VIRTUAL MEMORY

  - INVISIBLE ABSOLUTE MEMORY ADDRESSES

  - ADDRESS SPACE UNIQUE TO PROCESS

  - RESIDUE CLEARED PRIOR TO PAGE ALLOCATION

- FILE AND PROGRAM ACCESS: ACL (DISCRETIONARY)

  - ACL MECHANISM SEPARATES AND PROTECTS USERS FROM OTHER USERS ON THE BASIS OF Person_id AND Project_id

  - EVERY SEGMENT AND DIRECTORY HAS AN ASSOCIATED ACCESS CONTROL LIST

  - ACL DETERMINES WHO MAY ACCESS A SEGMENT OR DIRECTORY AND HOW THEY MAY ACCESS IT

  - AN ACL IS MANIPULATED AT THE DISCRETION OF THE SEGMENT'S OR DIRECTORY'S OWNER

SYSTEM SECURITY

- FILE AND PROGRAM ACCESS: AIM (NONDISCRETIONARY)

  - AIM (ACCESS ISOLATION MECHANISM) MECHANISM SEPARATES AND PROTECTS USERS FROM OTHER USERS ON THE BASIS OF SECURITY LEVEL AND THE NEED TO KNOW

  - EVERY SEGMENT, DIRECTORY AND USER HAS AN ASSIGNED SENSITIVITY (SECURITY) LEVEL AND CATEGORY SET

  - AIM RESTRICTS SEGMENT AND DIRECTORY ACCESS TO USERS BELONGING TO THE SAME CATEGORY SET AND HAVING THE SAME, OR HIGHER, SENSITIVITY LEVEL

  - SENSITIVITY LEVELS AND CATEGORY SETS ARE ASSIGNED BY THE SYSTEM ADMINISTRATOR (UP TO 8 LEVELS AND 18 CATEGORIES)

  - USERS CANNOT "GIVE AWAY" ACCESS OR WRITE DATA INTO A LOWER SENSITIVITY LEVEL REGARDLESS OF ACL PERMISSIONS

  - AIM IS A SITE CONTROLLED OPTION

		CATEGORY			
		NATO	CIA	AIR FORCE	NAVY
LEVEL	TOP SECRET				
	SECRET				
	UNCLASSIFIED				

SYSTEM SECURITY

- FILE AND PROGRAM ACCESS: RINGS (INTRAPROCESS)

  - ▯ THE RING MECHANISM SEPARATES AND PROTECTS THE OPERATING SYSTEM FROM THE USERS

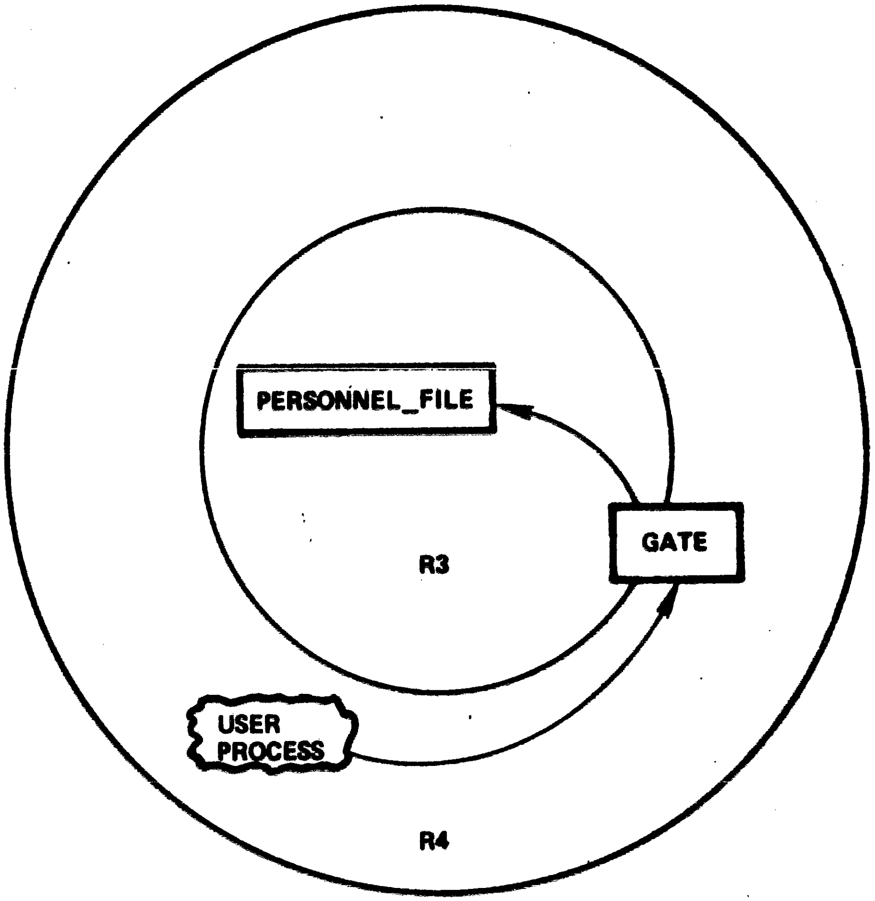
  - ▯ THE RING STRUCTURE IS AN 8 LEVEL (0 THRU 7) MASTER-MODE/SLAVE-MODE HIERARCHY

    - ▯ 0 - CENTRAL SUPERVISOR (MOST PRIVILEGED)

    - ▯ 1 - SYSTEM ROUTINES

    - ▯ 4 - NORMAL USER RING

    - ▯ 7 - HIGHEST USER RING (LEAST PRIVILEGED)


PERSONNEL_FILE
GATE
R3
USER PROCESS
R4

SYSTEM SECURITY

- EACH SEGMENT HAS AN ATTRIBUTE WHICH IS A SET OF INTEGERS KNOWN AS RING BRACKETS

- THE RING BRACKETS DEFINE FROM WHICH RING(S) A PROCESS MAY READ, WRITE, CALL, OR EXECUTE THAT SEGMENT

- EACH PROCESS IS CREATED IN A GIVEN RING DETERMINED AT LOGIN

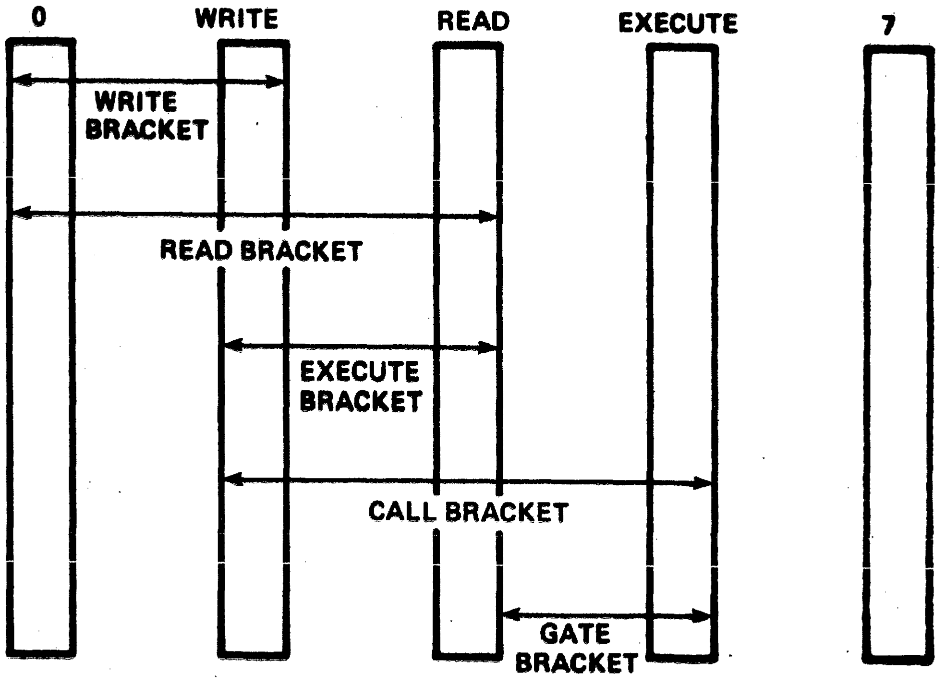
- A PROCESS MAY TEMPORARILY CHANGE ITS RING OF EXECUTION BY EXECUTING A PROGRAM CALLED A GATE

  - EXAMPLE:

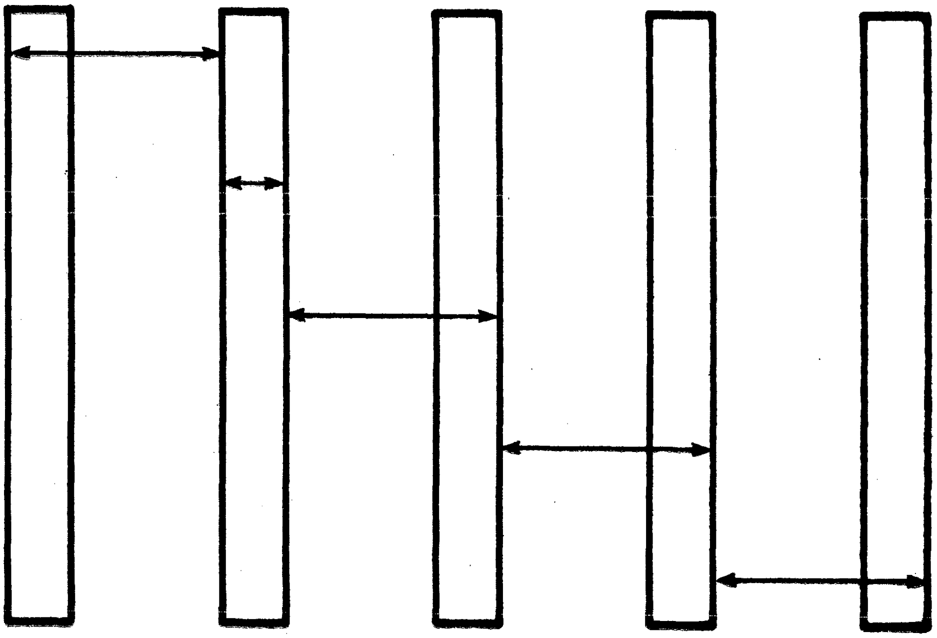
    - A DATA BASE IN A LOWER RING THAN A USER CAN ONLY BE ACCESSED BY THAT USER VIA AN OWNER WRITTEN "GATE" PROCEDURE - REGARDLESS OF AIM AND ACL PERMISSIONS

  - HARDWARE ENFORCED AT EVERY ACCESS

RING MECHANISM SUMMARY



RING OF EXECUTION



CORRESPONDING PERMITTED ACTION

READ, WRITE, EXECUTE (WITH RIN CHANGE)

READ, WRITE, AND EXECUTE

READ, EXECUTE

EXECUTE (IF A GAT ONLY, AND WITH RI CHANGE)

NONE

* SUBJECT, OF COURSE, TO ACL AND AIM

SYSTEM DAEMONS

- Daemon

  - A SYSTEM SERVICE PROCESS THAT PERFORMS A SPECIFIC TASK SUCH AS PROCESS CREATION, BACKUP, NETWORK CONTROL, PERIPHERAL I/O

  - LOGGED IN BY THE OPERATOR AND CANNOT "TIME OUT" AS A USER MIGHT

- Backup.SysDaemon

  - A PROCESS DESIGNED TO PRODUCE BOTH INCREMENTAL AND CONSOLIDATED BACKUP COPIES OF THE STORAGE SYSTEM

  - BACKUP IS TO MAGNETIC TAPE

  - SITE DETERMINES THE FREQUENCY OF THE BACKUPS

- Card_Input.Daemon

  - A PROCESS DESIGNED TO MANAGE THE SYSTEM CARD READER(S)

- IO.SysDaemon

  - A PROCESS DESIGNED TO MANAGE THE SYSTEM'S LINE PRINTERS AND CARD PUNCHES

SYSTEM DAEMONS

- Dumper.SysDaemon

  - A PROCESS DESIGNED TO PRODUCE A COMPLETE BACKUP OF THE STORAGE SYSTEM

- GCOS.SysDaemon

  - A PROCESS DESIGNED TO AID IN THE SIMULATION OF A GCOS ENVIRONMENT ON MULTICS

  - ALLOWS STANDARD GCOS JOBS TO BE SUBMITTED FROM EITHER PUNCHED CARDS OR IMCV TAPES

- Initializer.SysDaemon

  - THE SYSTEM'S PRIMARY PROCESS. PERFORMS THE FOLLOWING FUNCTIONS:

    - ANSWERING SERVICE OPERATIONS (login, dial, logout, etc)

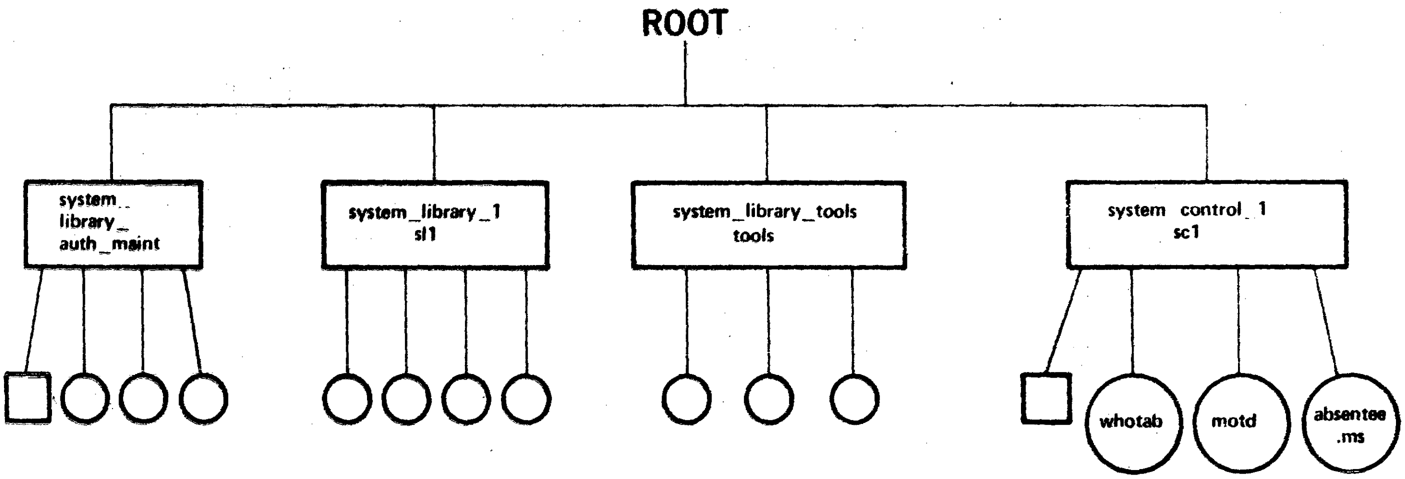
    - OPERATOR COMMAND SERVICE

    - OPERATOR TERMINAL MANAGEMENT AND MESSAGE ROUTING

    - SYSTEM ACCOUNTING AND ADMINISTRATION

    - USER REQUEST HANDLING (logout, new_proc, etc)

SYSTEM LIBRARIES AND DIRECTORIES

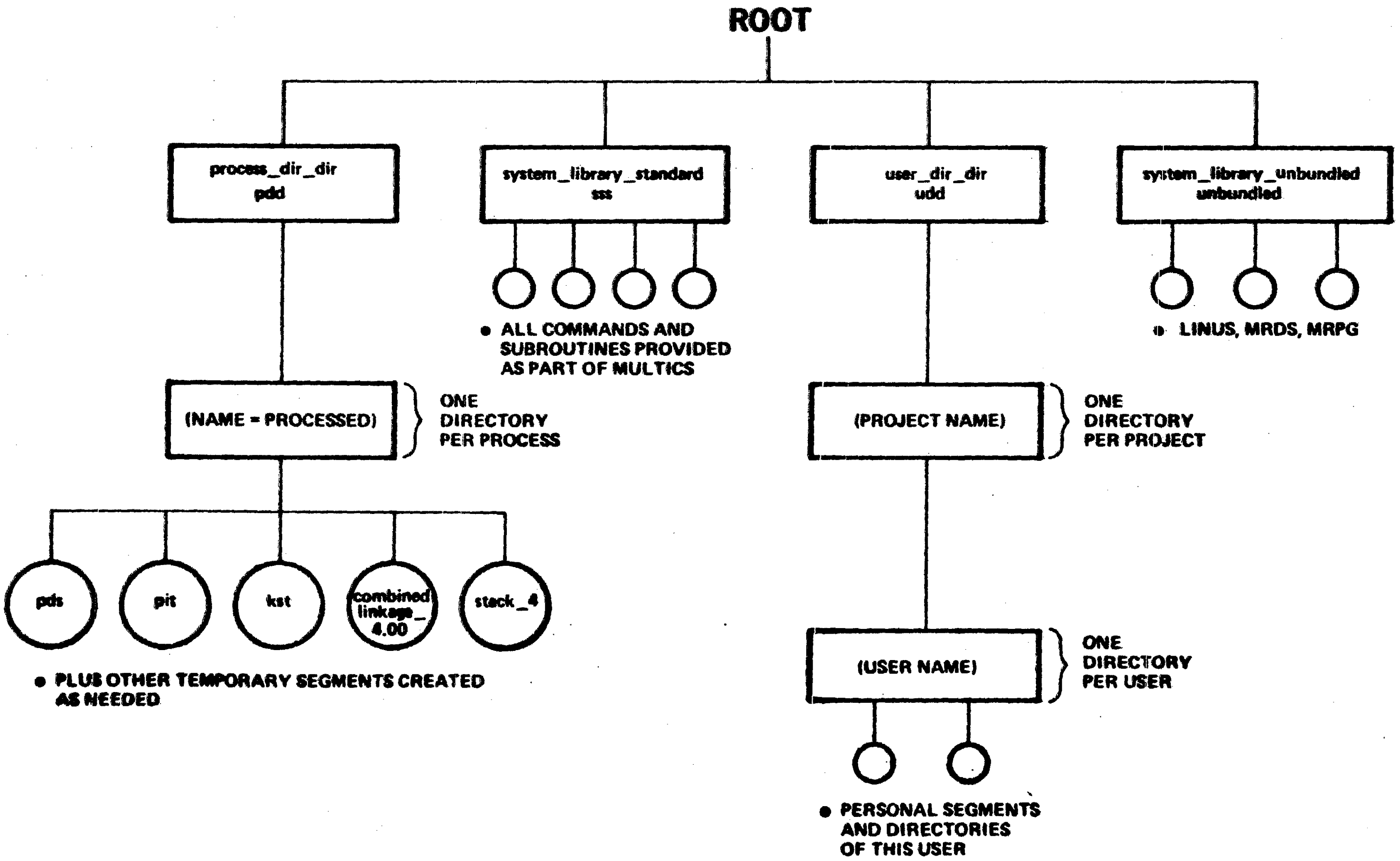


- COMMAND AND SUBROUTINES OF THE LOCAL AUTHOR-MENTION LIBRARY
- ted, lisp, pascal

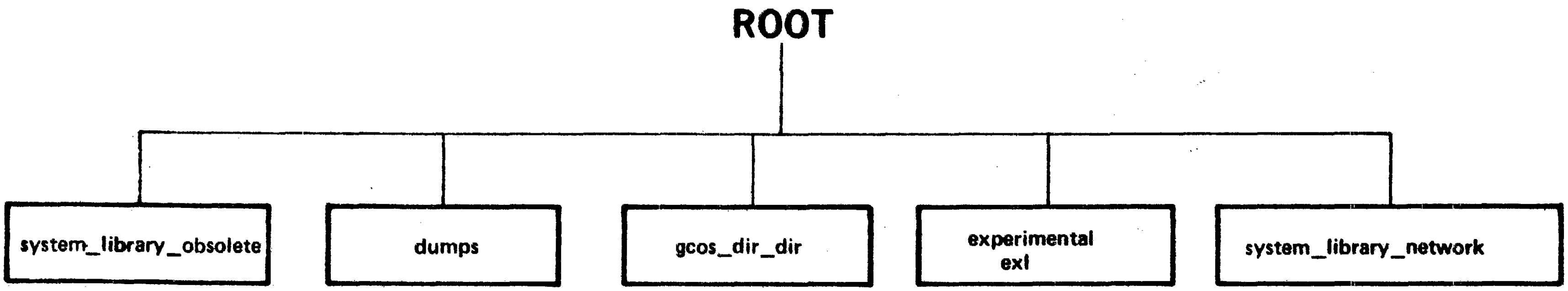
- HARDCORE OPERATING SYSTEM PROCEDURES
- RELOADED EACH TIME THE SYSTEM IS REINITIALIZED

- COMMANDS AND SUBROUTINES SUBROUTINES USED TO ADMINISTER, MEASURE, AND MAINTAIN THE SYSTEM
- PRIMARILY OF INTEREST TO SYSTEM PROGRAMMERS

- PLUS MISCELLANEOUS ACCOUNTING, LOG, LINE USAGE, PASSWORD SEGMENTS AND THE I/O RESOURCE CONTROL PACKAGES


ROOT
process_dir_dir
pdd
system_library_standard
sss
user_dir_dir
udd
system_library_unbundled
unbundled
• ALL COMMANDS AND SUBROUTINES PROVIDED AS PART OF MULTICS
• LINUS, MRDS, MRPG
(NAME = PROCESSED)
ONE DIRECTORY PER PROCESS
(PROJECT NAME)
ONE DIRECTORY PER PROJECT
pds
pit
kst
combined linkage_ 4.00
stack_4
• PLUS OTHER TEMPORARY SEGMENTS CREATED AS NEEDED
(USER NAME)
ONE DIRECTORY PER USER
• PERSONAL SEGMENTS AND DIRECTORIES OF THIS USER

SYSTEM LIBRARIES AND DIRECTORIES



ROOT

system_library_obsolete

- CONTAINS OBSOLETE OBJECT SEGMENTS

dumps

- CONTAINS SYSTEM DUMPS FOR CRASHES, ETC.

gcos_dir_dir

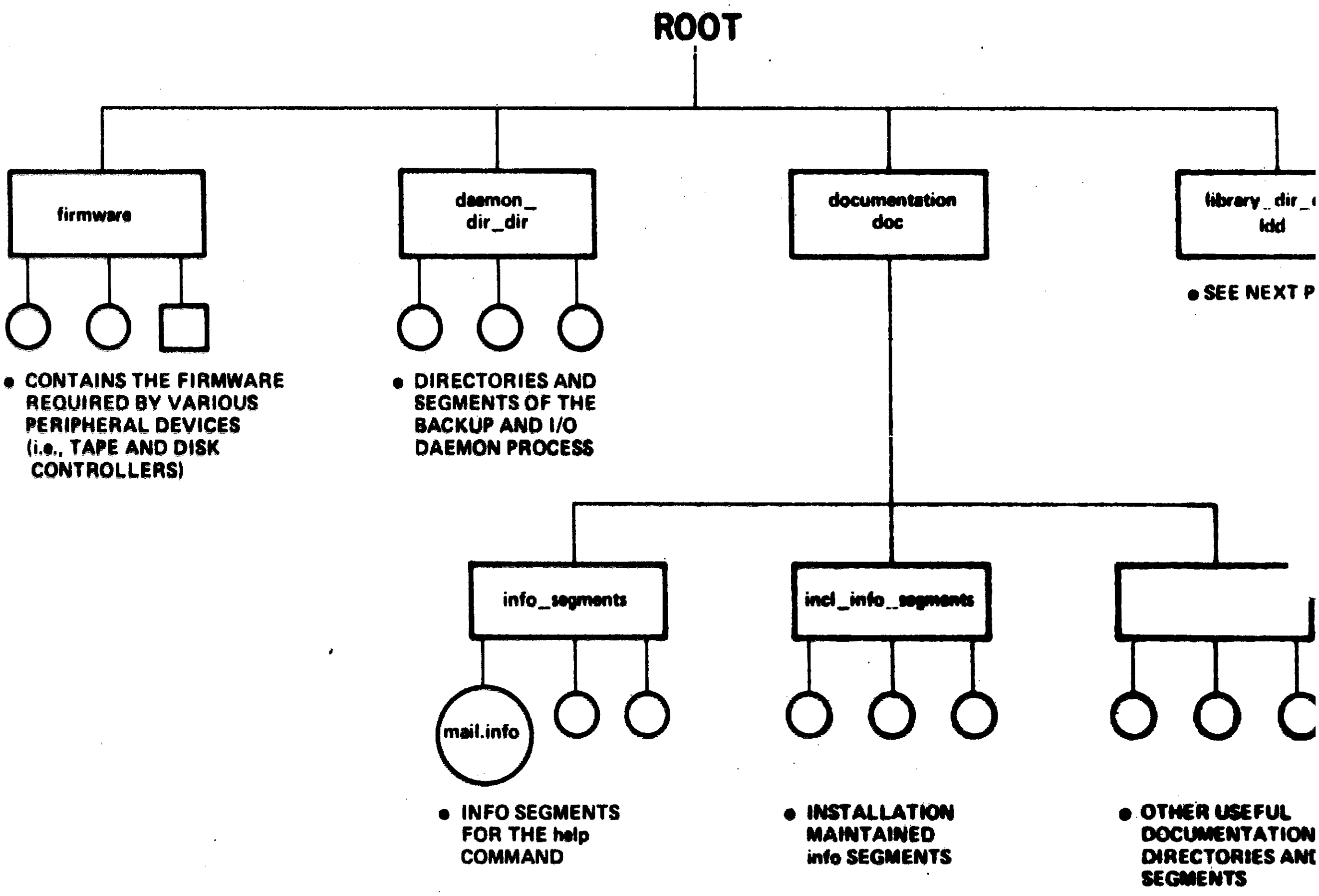
- CONTAINS SYSTEM SOFTWARE USEFUL FOR HANDLING THE GCOS ENCAPSULATION

experimental
exl

- CONTAINS SYSTEM SOFTWARE WHICH IS BEING DEVELOPED
- CONTAINS PROCEDURES BEING USED IN PLACE OF SYSTEM PROCEDURES WHICH ARE FAULTY

system_library_network

- CONTAINS OBJECT SEGMENTS FOR NETWORK PROCESSING



- CONTAINS THE FIRMWARE REQUIRED BY VARIOUS PERIPHERAL DEVICES (i.e., TAPE AND DISK CONTROLLERS)

- DIRECTORIES AND SEGMENTS OF THE BACKUP AND I/O DAEMON PROCESS

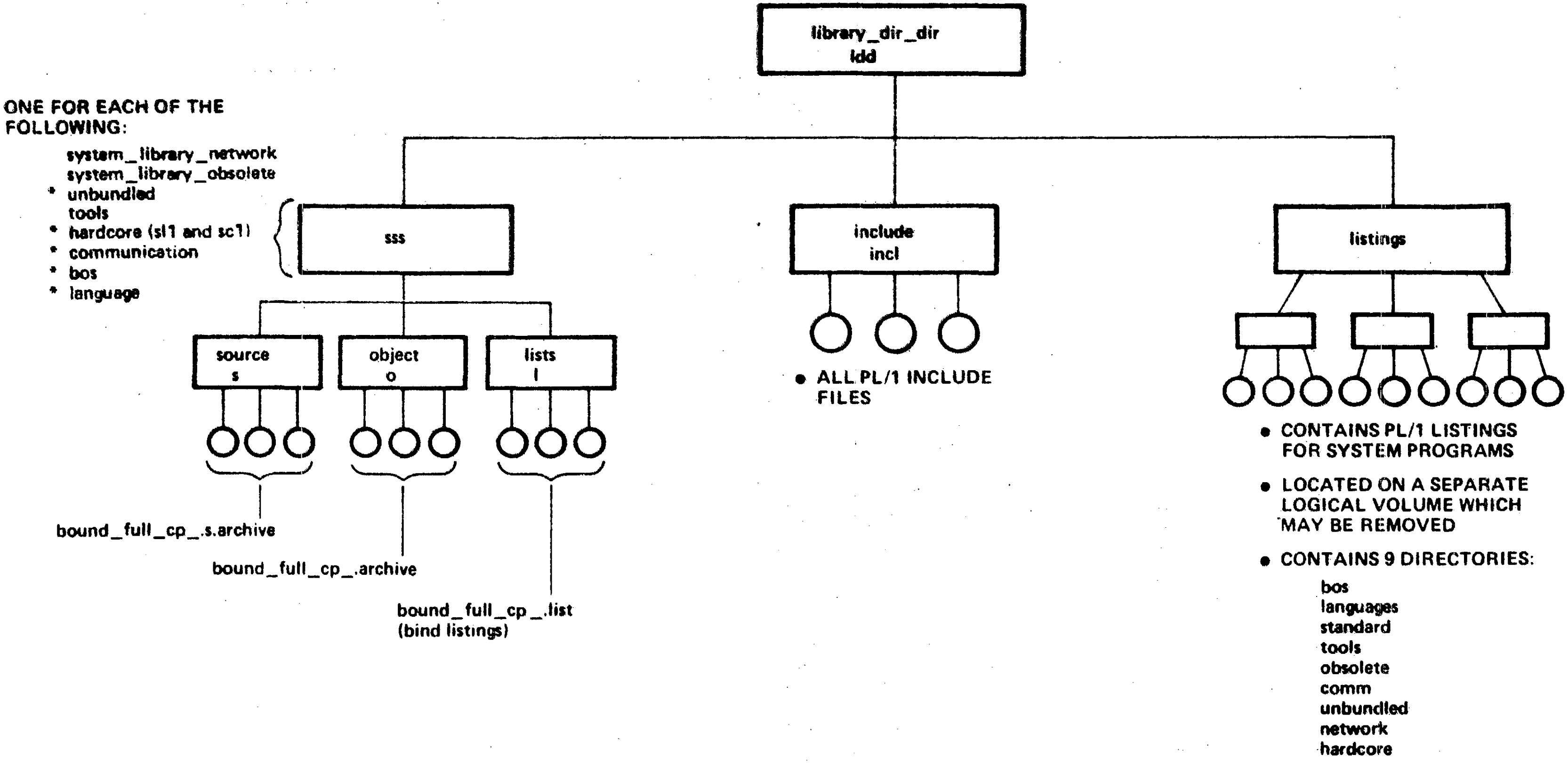
- SEE NEXT P

- INFO SEGMENTS FOR THE help COMMAND

- INSTALLATION MAINTAINED info SEGMENTS

- OTHER USEFUL DOCUMENTATION DIRECTORIES ANI SEGMENTS

SYSTEM LIBRARIES AND DIRECTORIES



library_dir_dir
ldd

ONE FOR EACH OF THE FOLLOWING:

- system_library_network
- system_library_obsolete
- * unbundled
- tools
- * hardcore (sl1 and sc1)
- * communication
- * bos
- * language

sss

source
s

object
o

lists
l

bound_full_cp_.s.archive

bound_full_cp_.archive

bound_full_cp_.list
(bind listings)

include
incl

- ALL PL/1 INCLUDE FILES

listings

- CONTAINS PL/1 LISTINGS FOR SYSTEM PROGRAMS
- LOCATED ON A SEPARATE LOGICAL VOLUME WHICH MAY BE REMOVED
- CONTAINS 9 DIRECTORIES:
  - bos
  - languages
  - standard
  - tools
  - obsolete
  - comm
  - unbundled
  - network
  - hardcore

* THESE CONTAIN ADDITIONAL DIRECTORIES

- MULTICS DATA BASE MANAGER (MDBM)
  - MULTICS INTEGRATED DATA STORE (MIDS)
    - SUPPORTS NETWORK DATA BASES
    - A SUBSET OF IDS-II
    - PROCEDURAL INTERFACE (USER SPECIFIES HOW TO SEARCH)
  - MULTICS RELATIONAL DATA STORE (MRDS)
    - SUPPORTS RELATIONAL DATA BASES
    - INDUSTRY'S FIRST COMMERCIALLY AVAILABLE RELATIONAL DBM
    - NON-PROCEDURAL INTERFACE USING ENGLISH-LIKE EXPRESSIONS (USER SPECIFIES GOAL OF SEARCH)
    - SET OPERATIONS ON RELATIONS (I.E. ON FILES):
      - UNION, INTERSECTION, DIFFERENCE
    - BOOLEAN OPERATIONS BETWEEN CONDITIONS FOR SELECTION:
      - AND, OR, NOT
    - ALGEBRAIC COMPARISONS ON ATTRIBUTES (I.E. ON DATA FIELDS):
      - EQUAL, GREATER THAN, LESS THAN, NOT
    - BUILT-IN FUNCTIONS
      - abs, after, before, ceil, concat, floor, index, mod, reverse, round, search, substr, verify

APPLICATION PACKAGES

- LOGICAL INQUIRY AND UPDATE SYSTEM (LINUS)

  - END USER FACILITY FOR ACCESSING RELATIONAL DATA BASES

  - ENGLISH-LIKE EXPRESSIONS

    - FIND THE AVERAGE SALARY OF EMPLOYEES IN THE SHOE DEPARTMENT

      ```
      avg {select salary
           from   employee_table
           where  dept = "Shoe"}
      ```

  - BOOLEAN OPERATIONS BETWEEN CONDITIONS FOR SELECTION:

    - AND, OR, NOT

  - ALGEBRAIC COMPARISONS ON ATTRIBUTES (I.E. ON DATA FIELDS):

    - EQUAL, GREATER THAN, LESS THAN, NOT

    - FIND THE NAMES OF EMPLOYEES WHO ARE EITHER IN THE ADMIN DEPARTMENT OR WHOSE TOTAL INCOME EXCEEDS $10,000

      ```
      select name
      from   employee_table
      where  dept = "Admin"¦salary+comm>10000
      ```

- SET OPERATIONS ON RELATIONS (I.E. ON FILES):
  - UNION, INTERSECTION, DIFFERENCE
  - FIND THOSE ITEMS WHICH ARE SUPPLIED BY LEVI AND SOLD IN THE MEN'S DEPARTMENT

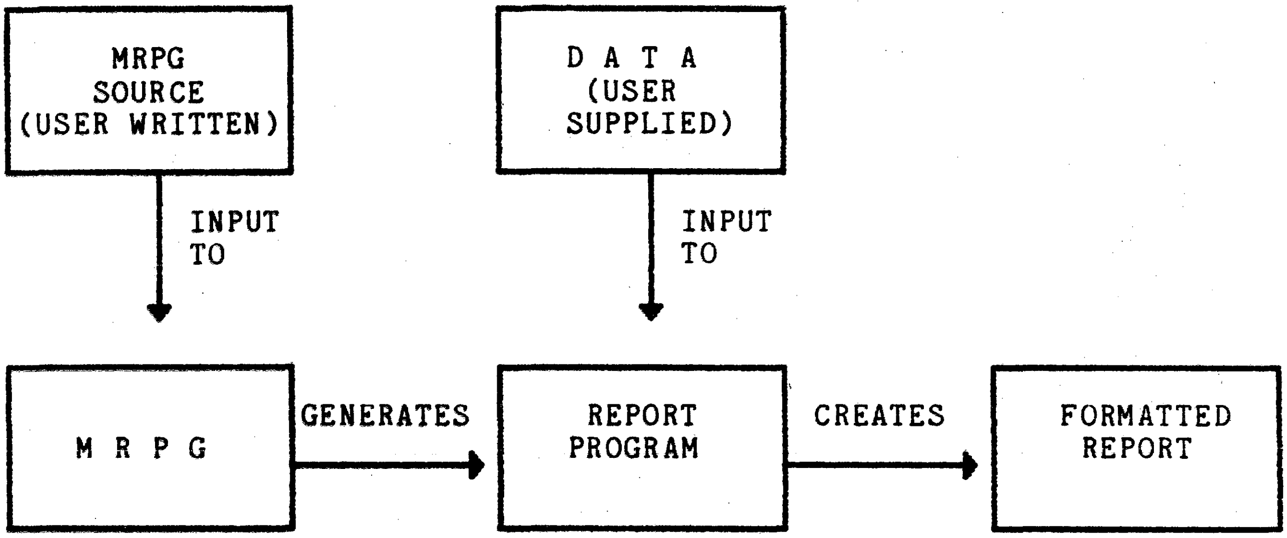
```
            select item
            from   supply
            where  supplier = "Levi"
     inter
            select item
            from   sales
            where  dept = "Men"
```

- BUILT-IN FUNCTIONS
  - abs, after, before, ceil, concat, floor, index, mod, reverse, r search, substr, verify
  - max, min, sum, ave, count
- PERMITS USER DEFINABLE FUNCTIONS (I.E., MACROS)

APPLICATION PACKAGES

- MULTICS REPORT PROGRAM GENERATOR

  - A FACILITY FOR GENERATING FORMATTED REPORTS

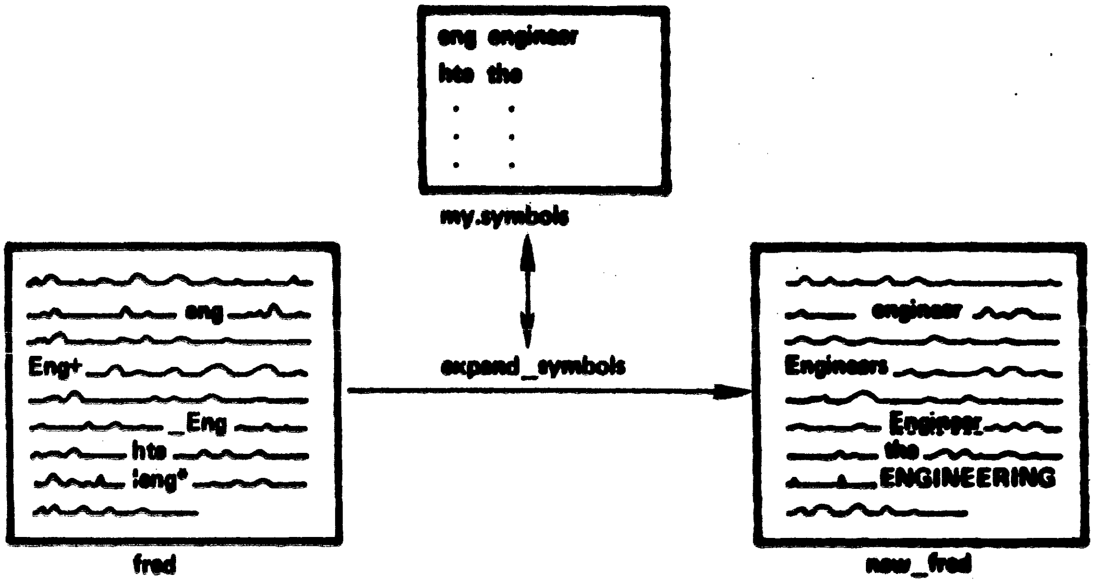


- WORD PROCESSING (WORDPRO)

  - A COLLECTION OF FACILITIES FOR ENHANCING THE ON-LINE PREPARATION DISTRIBUTION, AND MAINTENANCE OF DOCUMENTS

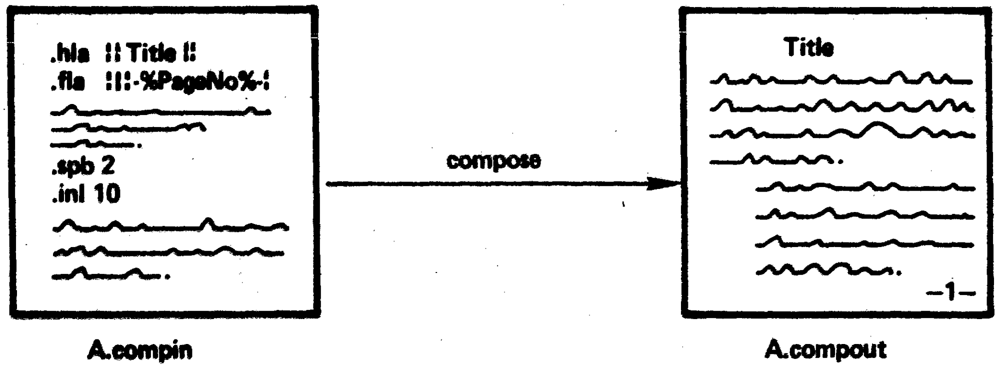
  - CONSISTS OF SUCH TOOLS AS:

    - SPEEDTYPE COMMANDS



APPLICATION PACKAGES

- compose FACILITY

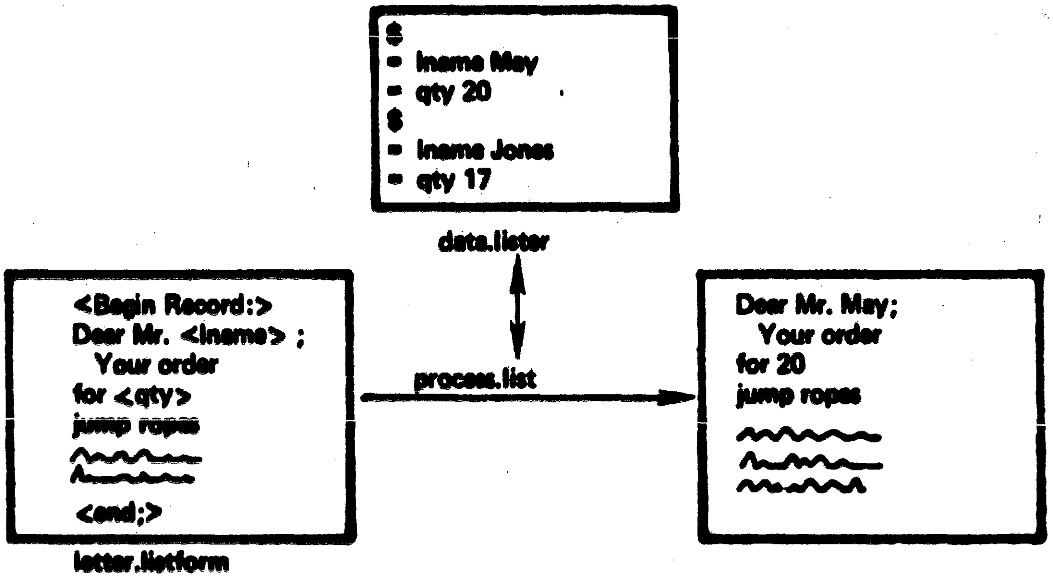


- WORDLIST COMMANDS

- ELECTRONIC MAIL COMMANDS

- TEXT EDITOR

▌ LIST PROCESSING COMMANDS



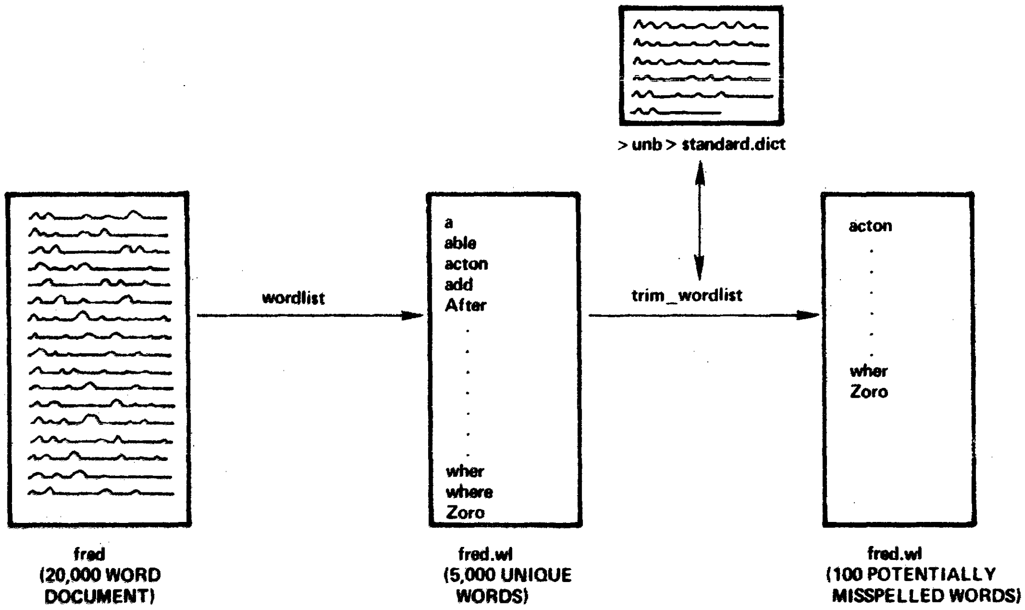
```
$
= lname May
= qty 20
$
= lname Jones
= qty 17
```

```
<Begin Record:>
Dear Mr. <lname> ;
  Your order
for <qty>
jump ropes

<end;>
```

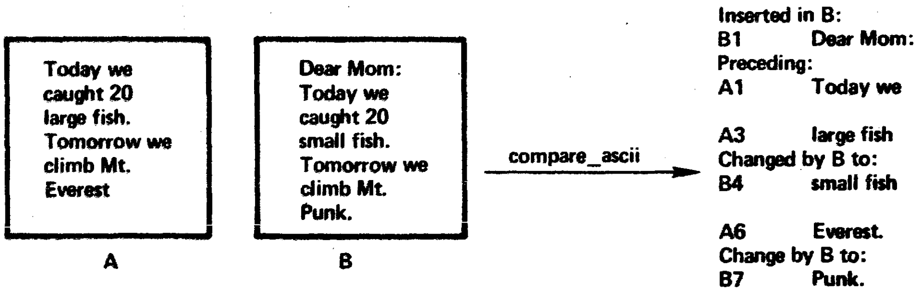
```
Dear Mr. May;
  Your order
for 20
jump ropes
```

APPLICATION PACKAGES

ONLINE DICTIONARIES



TEXT COMPARISON PROGRAMS



Today we
caught 20
large fish.
Tomorrow we
climb Mt.
Everest

Dear Mom:
Today we
caught 20
small fish.
Tomorrow we
climb Mt.
Punk.

```
Inserted in B:
B1        Dear Mom:
Preceding:
A1        Today we

A3        large fish
Changed by B to:
B4        small fish

A6        Everest.
Change by B to:
B7        Punk.
```

- Multics GRAPHICS SYSTEM
  - DEVICE INDEPENDENCE
    - USER SEES ONLY "VIRTUAL GRAPHICS TERMINAL" (VGT)
    - EACH DEVICE HAS A "GRAPHIC DEVICE TABLE" (GDT)
  - STRUCTURED GRAPHIC OBJECTS
    - SIMPLE ITEMS (LINES, POINTS) COMBINED TO CREATE MORE COMPLEX
    - GRAPHIC OBJECTS MAY BE "SHARED" (EX. CREATE A WHEEL ONCE, INCLUDE 4 TIMES IN AN AUTOMOBILE STRUCTURE)
    - ALL ITEMS ARE THREE DIMENSIONAL
    - OBJECTS MAY BE NAMED
    - OBJECTS MAY BE PERMANENTLY STORED
    - ANCILLARY INFORMATION STORED WITH OBJECT
      - SCALING
      - ROTATION

APPLICATION PACKAGES

    - EXTENT
    - INTENSITY
    - COLOR
    - DOTTEDNESS
    - BLINKING
    - SENSITIVITY TO LIGHT PENS, ETC.

- EDITING FACILITIES
  - MAY EDIT THE WORKING GRAPHIC SEGMENT (WGS)
  - REAL TIME EDITING AT TERMINAL
  - CONTROLLED BY Multics (ANIMATION, DYNAMIC GRAPHICS)
- PERMANENT STORAGE OF GRAPHIC OBJECTS
- TERMINALS SUPPORTED
  - TEKTRONIX 4002
    4012
    4013
    4014
    4015

APPLICATION PACKAGES

- ARDS

- CALCOMP 915/1036 (PLOTTER)

TOPIC XXI

HARDWARE OVERVIEW

Page

This page has intentionally

been left blank.

HARDWARE DESCRIPTION

- LEVEL 68 CENTRAL PROCESSOR UNIT
  - A MODIFIED LEVEL 66 CPU
  - 36-BIT WORD MACHINE (9 BITS/BYTE, 4 BYTES/WORD)
  - VIRTUAL MEMORY HARDWARE
    - DIVIDES MEMORY INTO SEGMENTS
    - SEGMENTS CONSIST OF 0 TO 256 PAGES
    - PAGE = 1024 WORDS (1K WORDS)
  - RING PROTECTION HARDWARE
  - ACCESS ENFORCING HARDWARE
  - HIGH-SPEED CACHE MEMORY
    - 2048 WORDS
    - HIT RATIO GREATER THAN 85%

- EXTENDED INSTRUCTION SET (EIS)
  - BLOCK MOVE INSTRUCTION (ENTIRE SEGMENT)
  - BOOLEAN OPERATION INSTRUCTIONS
  - PICTURE EDITING INSTRUCTIONS
  - 4, 6, OR 9-BIT ARITHMETIC INSTRUCTIONS
  - POINTER MANIPULATION INSTRUCTIONS
- GCOS MODE (ONE SWITCH)

SYSTEM REQUIREMENTS

- A SMALL MULTICS CONFIGURATION

	NUMBER
▯ 68/60 CENTRAL PROCESSOR (CPU)	1
▯ WORDS OF MEMORY (K = 1024)	256K
▯ SYSTEM CONTROL UNIT (SCU)	1
▯ I/O MULTIPLEXER (IOM)	1
▯ FRONT-END NETWORK PROCESSOR (FNP)	1
▯ MSU0400 2 MASS STORAGE UNITS (MSU)	2
▯ MTU0400 MAGNETIC TAPE UNITS (MTU)	2

SYSTEM REQUIREMENTS

- A LARGE MULTICS CONFIGURATION (1 TO 10 GROWTH CAPABILITY)

	NUMBER	RATIO
▯ 68/80 CENTRAL PROCESSORS	6	(1:6)
▯ WORDS OF MEMORY (M=1,048,576)	16M	(1:64)
▯ SYSTEM CONTROL UNITS	8	(1:8)
▯ I/O MULTIPLEXER (# IOM'S + # CPU'S ≤ 8)	2	(1:2)
▯ FRONT-END NETWORK PROCESSORS	4	(1:4)
▯ MSU0451 MASS STORE UNITS	512	(1:256)
▯ MTU0500 MAGNETIC TAPE UNITS	16	(1:8)

CONFIGURATION LIMITS & RECORDS

MEASURE	RECORD	SITE	KNOWN LIMIT
NUMBER OF USERS	457	HIS, PHOENIX (JUNE '77)	1632 (408 USERS * 4 DATANETS)
NUMBER OF CPU's	6	AFDSC,PENTAGON (AUG '77)	7 (PORT LIMITATION)
MAXIMUM REAL MEMORY	4M	HIS, Phoenix (SEPT '78)	16M (8 SCU's * 512K)
MINIMUM REAL MEMORY	192K	HIS, CAMBRIDGE (1971 - 1977)	192K (BOOT LIMITATION)
VIRTUAL MEMORY PER PROCESS	----	-------	256M (1024 SDW's * 256K)
TOTAL VIRTUAL MEMORY	----	-------	19.4B (512 MSU'S * 38,000K)
NUMBER OF DISK DRIVES	28	AFDSC,PENTAGON (JAN '76)	512 (CABLE LENGTH LIMIT)

K =	2**10 =	1,024
M =	2**20 =	1,048,576
B =	2**30 =	1,063,741,824

This page has intentionally

been left blank

APPENDIX A

MULTICS DOCUMENTATION

This page has intentionally
been left blank.

MULTICS PROGRAMMERS' MANUAL (MPM)

Reference Guide Order No. AG91

The MPM Reference Guide contains general information about the Multics command and programming environments. It also defines items used throughout the rest of the MPM's and, in addition, describes such objects as the command language, the storage system, and the input/output system.

Commands and Active Functions Order No. AG92

The Commands MPM is organized into four sections. Section I contains a list of the Multics command repertoire, arranged functionally. Section II describes the active functions. Section III contains descriptions of standard Multics commands, including the calling sequence and usage of each command. Section IV describes the requests used to gain access to the system.

Subroutines Order No. AG93

The Subroutines MPM is organized into three sections. Section I contains a list of the subroutine repertoire, arranged functionally. Section II contains descriptions of the standard Multics subroutines, including the declare statement, the calling sequence, and usage of each. Section III contains the descriptions of the I/O modules.

Subsystem Writers' Guide Order No. AK92

The MPM Subsystem Writers' Guide is a reference of interest to compiler writers and writers of sophisticated subsystems. It documents user-accessible modules that allow the user to bypass standard Multics facilities. The interfaces thus documented are a level deeper into the system than those required by the majority of users.

Peripheral Input/Output Order No. AX49

The MPM I/O manual contains descriptions of commands and subroutines used to perform peripheral I/O. Included in this manual are commands and subroutines that manipulate tapes and disks as I/O devices. Special purpose communications I/O, such as binary synchronous communication, is also included.

MULTICS POCKET GUIDES

Commands and Active Functions Order No. AW17

This pocket guide presents an abbreviated version of the commands and active functions described in detail in the Multics Programmers' Manual Commands and Active Functions Order No. AG92

MULTICS ADMINISTRATORS' MANUALS (MAM)

Project Administrator Order No. AK51

The Project MAM is a guide to the operation of programs in the project- administration area. The information in this manual is of interest not only to project administrators but also to accounting administrators (who may function as project administrators) and to system administrators (who may function in any administrative capacity).

Registration and Accounting Administrator Order No. AS68

The Accounting MAM is a guide to the operation of Multics billing and accounting programs. It is necessary that both the accounting and system administrators know how to perform the Multics billing operations.

System Administrator Order No. AK50

The System MAM is a guide to the overall administration of the Multics system. This manual discusses the contents of administrative directories and data bases and special user identifies (such as the daemons), describes installation parameters and system logs, explains the various tasks that are the responsibility of the system administrator, and includes the commands needed to carry out these responsibilities. Also, the functions of the system security administrator are explained in the MAM System.

PROGRAM LOGIC MANUALS (PLM)

NOTE: The Distribution of Program Logic Manuals is Restricted

ALM Assembler	Order No. AN69
Carry Facility	Order No. AN76
Hardware & Software Formats	Order No. AN87
Reconfiguration	Order No. AN71
Storage System	Order No. AN61
System Dump Analysis	Order No. AN53
System Initialization	Order No. AN70
System Metering	Order No. AN52
System Tools	Order No. AN51
User Ring I/O System	Order No. AN57

OTHER MULTICS MANUALS

APL Users' Guide	Order No. AK95
BASIC	Order No. AM82
COBOL Reference Manual	Order No. AS44
COBOL Users' Guide	Order No. AS43
DFAST Subsystem Users' Guide	Order No. AT59
FAST Subsystem Users' Guide	Order No. AU25
FORTRAN Reference Manual	Order No. AT58
GCOS Environment Simulator	Order No. AN05
Graphics System	Order No. AS40
Hardware Diagnostic Aids	Order No. AR97
Logical Inquiry and Update System (LINUS)	Order No. AZ49
Multics Integrated Data Store Reference Manual	Draft

OTHER MULTICS MANUALS

Multics Relational Data Store Reference Manual	Order No. AW53
New Users' Guide	Order No. AL40
Online T&D Reference Manual	Order No. AU77
Operator's Handbook	Order No. AM81
PL/I Language Specification	Order No. AG94
PL/I Reference Manual	Order No. AM83
Processor Manual	Order No. AL39
SORT/MERGE	Order No. AW32
Site Preparation Manual	Order No. DC79
System Summary Description	Order No. AK15
Virtual Memory	Order No. AG95
WORDPRO Reference Guide	ORder No. AZ98

HELP MANUALS

- THE MULTICS TECHNICAL MANUALS ARE LISTED BELOW. ALL MANUALS/UPDATES THAT SPECIFY A DATE IN THE DATE COLUMN CAN BE ORDERED FROM THE DISTRIBUTION CENTER; FOR INFORMATION ABOUT HOW TO ORDER MANUALS TYPE:

```
help order_manuals
```

- IF THE "Date" COLUMN DOES NOT CONTAIN A DATE FOR A MANUAL, THAT MANUAL IS "IN PROGRESS;" WHEN A DATE IS SPECIFIED, YOU CAN ORDER IT.

- PROGRAM LOGIC MANUALS (PLMS) ARE LISTED SEPARATELY, AFTER THE LIST OF CUSTOMER MANUALS. MARKETING DOCUMENTS (E.G., BROCHURES, PRODUCT BRIEFS, ETC.) ARE NOT INCLUDED.

THIS MANUAL LISTING WAS CURRENT AS OF JULY 23, 1979

Order			Through		
Number	Rev.	Add.	Release	Date	Title
CUSTOMER MANUALS					
+AG90	-01	---	---	05/73	MPM - Introduction
AG91	-02	---	7.0	03/79	MPM - Reference Guide
		A	8.0	-----	
AG92	-02	---	5.0	01/77	MPM - Commands and Active Functions
		A	6.0	11/77	
		B	7.0	02/79	
		C	7.0a	-----	
		D	8.0	-----	
AG93	-02	---	6.0	02/78	MPM - Subroutines
		A	7.0	12/78	
		B	8.0	-----	
AG94	-02	---	5.0	07/76	PL/I Language Specification
		A	6.0	10/77	
		B	7.0	12/78	
		C	8.0	-----	
+AG95	-00	---	---	06/72	Virtual Memory

HELP MANUALS

AK50	-01	---	5.0	10/76	MAM - System
		A	5.0	03/77	
		B	6.0	03/78	
		C	7.0	10/78	
		D	8.0	-----	
AK51	-01	---	4.0	08/76	MAM - Project
		A	6.0	04/78	
		B	7.0	02/79	
		C	8.0	-----	
AK92	-02	---	7.0	03/79	MPM - Subsystem Writers' Guide
		A	8.0	-----	
AK95	-01	---	7.0	03/79	APL Users' Guide
	-02	---	8.0	-----	
AL39	-00	---	---	04/76	Processor Manual
		A	---	09/76	
	-01	---	---	-----	
AL40	-01	---	6.0	07/77	Introductory Users' Guide
	-02	---	8.0	-----	Programmer's Introduction to Multics
AM81	-01	---	6.0	10/77	Operators' Handbook
		A	6.0	01/78	
		B	7.0	02/79	
	-02	---	8.0	-----	
AM82	-00	---	1.0	02/74	BASIC
AM83	-00	---	4.0	06/76	PL/I Reference Manual
		A	7.0	09/78	
	-01	---	8.0	-----	
AN05	-01	---	7.0	10/78	GCOS Environment Simulator
AN50	-01	---	7.0	03/79	Index to Multics Manuals
	-02	---	8.0	-----	
AN52	-01	---	7.0	02/79	System Metering
AN76	-01	---	7.0	11/78	Carry Facility
		A	7.0	12/78	
AR97	-01	---	6.0	07/78	Hardware Diagnostic Aids
		A	6.0	11/78	
AS40	-00	---	6.0	10/77	Graphics System
	-01	---	8.0	-----	
AS43	-01	---	5.0	12/76	COBOL Users' Guide
		A	6.0	04/78	
		B	7.0	02/79	
		C	7.0b	-----	
AS44	-01	---	5.0	12/76	COBOL Reference Manual
		A	6.0	10/77	
		B	7.0	01/79	
		C	7.0	05/79	
		D	7.0b	-----	
AS68	-00	---	4.0	11/76	MAM - Registration and Accounting
		A	7.0	01/79	
		B	8.0	-----	
AT58	-01	---	5.0	02/77	FORTRAN
		A	6.0	11/77	
		B	7.0	06/78	

HELP MANUALS

		C	7.0	01/79	
		D	8.0	-----	
AT59	-00	---	3.1	03/76	DFAST Subsystem Users' Guide
AU25	-00	---	3.1	03/76	FAST Subsystem Users' Guide
		A	8.0	-----	
AU77	-01	---	6.0	10/77	Online Test and Diagnostics Reference Manual
	-02	---	7.0	-----	
AW17	-00	---	3.1	04/76	MPG - Commands and Active Functions
	-01	---	8.0	-----	
AW32	-00	---	4.0	07/76	SORT/MERGE
AW53	-02	---	7.0	10/78	Relational Data Store (MRDS) Ref. Manual
AX49	-00	---	5.0	06/77	MPM - Peripheral Input/Output
		A	7.0	01/79	
		B	8.0	-----	
AZ03	-00	---	5.0	08/77	System Programming Tools
	-01	---	8.0	-----	
AZ49	-01	---	7.0	10/78	Logical Inquiry and Update System (LINUS)
AZ98	-01	---	7.0	11/78	WORDPRO Reference Manual
		A	8.0	-----	
CC34	-01	---	7.0	03/79	Bulk Input/Output
		A	8.0	-----	
CC69	-00	---	6.0	03/78	Report Program Generator (MRPG)
CC70	-00	---	7.0	01/79	FORTRAN Users' Guide
	-01	---	8.0	-----	
CC74	-00	---	7.0R	11/78	MAM - Resource Control Package (RCP)
		A	8.0	-----	
CC75	-00	---	7.0	11/78	MAM - Communications
		A	8.0	-----	
CC92	-00	---	7.0	10/78	MPM - Communications I/O
		A	8.0	-----	
*					
¦CC96	-01	---	7.0TP	06/79	Transaction Processing
CG18	-00	---	7.0	-----	Remote Batch Facility (Lev 68/Lev 6)(Prelim)
CG40	-00	---	7.0a	-----	qedx User's Guide (Preliminary)
CG41	-00	---	8.0	-----	compose User's Guide
CH23	-00	---	8.0	-----	Debugging User's Guide
CH24	-00	---	8.0	-----	New User's Introduction to Multics - Part I
CH25	-00	---	8.0	-----	New User's Introduction to Multics - Part II
CH26	-00	---	8.0	-----	Error Messages
CH27	-00	---	8.0	-----	emacs User's Guide

PROGRAM LOGIC MANUALS

AN51	-00	---	---	02/75	System Tools
AN53	-00	---	---	06/75	System Dump Analysis
AN57	-00	---	---	05/77	User Ring Input/Output System
AN61	-00	---	5.0	07/74	Storage System
		A	6.0	09/78	
AN63	-00	---	---	02/75	ALM Assembler
AN69	-00	---	---	02/75	Message Segment Facility
AN70	-00	---	---	02/75	System Initialization
AN71	-01	---	---	04/77	Reconfiguration
AN80	-00	---	---	-----	Library Maintenance
AN85	-01	---	7.0a	-----	Communications System
AN87	-00	---	---	07/76	Hardware & Software Formats

NOTES:

*	A line has been deleted since the last time this segment was updat
¦	This line has been modified since the last time the segment was up
+	Some of the information in this manual is obsolete
MPM	Multics Programmers' Manual
MAM	Multics Administrators' Manuals
MPG	Multics Pocket Guide

APPENDIX B

MULTICS TECHNICAL BRIEFS

This page has intentionally

been left blank.

Honeywell

SERIES 60 (LEVEL 68/DPS)

Level 68/Distributed Processing System

Honeywell's Level 68/Distributed Processing System (DPS), with the Multics operating system, offers the user many powerful features, including virtual memory addressing, controlled access to data, modular design, and advanced segmentation concepts that simplify processing. The Level 68/DPS possesses capabilities that enable large-scale computer users to solve complex processing problems quickly and easily.

The Level 68/Distributed Processing System consists of a base system to which performance modules can be added in incremental steps, thus offering a choice of various levels of performance. This easy expansion allows a user to configure the exact system needed and helps protect equipment investment.

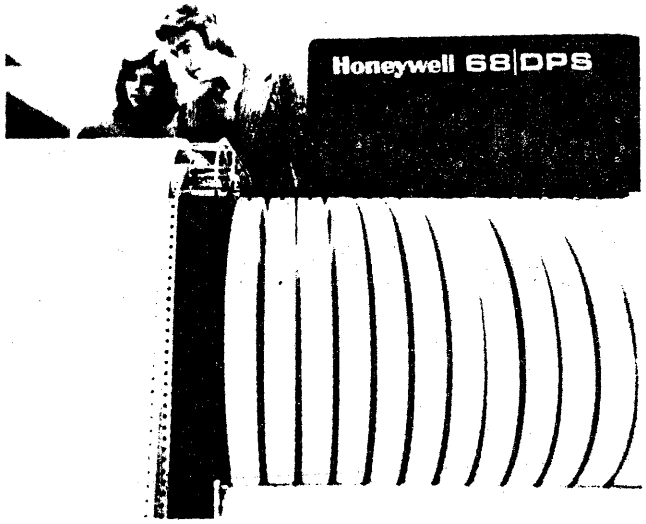
VIRTUAL MEMORY

Level 68/DPS virtual memory efficiently and automatically moves information between main memory and secondary storage – independent of hardware configuration and without programmer intervention. Thus, programs are not constrained by main memory limitations and no overlays are required. Furthermore, user I/O can be handled logically, without concern for physical addresses. Integration of the 68/DPS storage (file) system with virtual memory addressing forms a powerful data handling capability, allowing programmers to directly access more than 300 billion bytes of stores information.

SECURITY

Level 68/DPS hardware and software systems are structured for maximum data security. With simple commands, access to files can be granted to specific persons or groups. Different access rights (e.g., read only, execute, write, or combinations of these) can also be granted to different users of the same file.

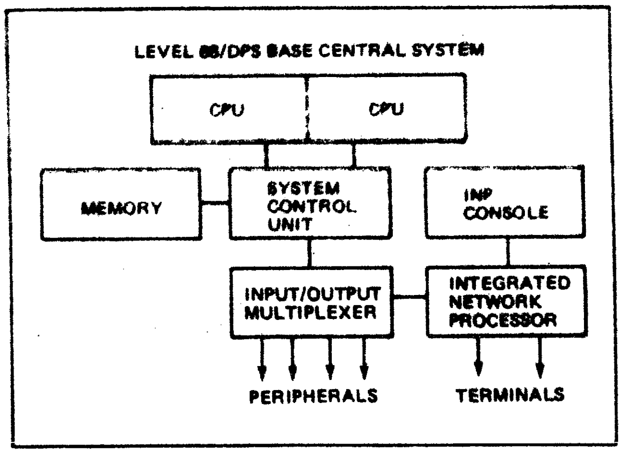
Within the central processor, a hardware mechanism maintains the integrity of several levels, or rings, of access controls. These rings of protection limit access to sensitive data and permit the creation of closed subsystems that are mutually exclusive and mutually protected.



Since hardware enforces the basic 68/DPS security mechanisms, very little system overhead is introduced by access control enforcement. After the initial access to a file and on every memory access, hardware compares the attempted access against the user's permission. Thus, no additional machine instructions are executed for security.

MODULARITY

The modular design of the Level 68/Distributed Processing System has significant advantages for processing flexibility and for system growth. The optimum combination of processor, memory, and input/output multiplexer modules can be selected for each installation. Integrated Network Processors, mass storage subsystems, and peripherals are additional modules that can be added to tailor a configuration. A Level 68/DPS system supports up to 16 million bytes of memory (four million words) and employs two or more processors for maximum availability. Memory can be added in increments of one million bytes up to four million bytes. Beyond this level, memory may be added in increments of two million bytes.



This approach allows a site to purchase equipment for today's processing needs and add modules to meet a growing data processing workload. Furthermore, each configuration can be given the proper mix of processing, storage, and I/O capacity, and modified as necessary to meet changing workload characteristics. Regardless of the configuration selected, the full functionality of Multics is available. Any program that can run on the largest system can also run on the smallest, and vice versa.

Another significant advantage of Level 68/DPS modular architecture is the capability for configuring fully fail-safe systems. This is accomplished by including additional system control units. I/O multiplexers, network processors, and appropriate peripheral subsystems. Note that it is not necessary to add processor modules since Level 68/DPS always contains at least two.

PROCESSOR MODULE

The Level 68/DPS processor module executes programs and handles all computations. For example, it performs instruction fetching, relative and absolute address preparation, memory protection, data fetching, and data storage. These functions are overlapped for quick instruction execution.

Other features of the processor include:

- Hardware for handling segmentation and paging in virtual memory
- Hardware for interrupting a process in execution at any point (including in mid-execution of an instruction), saving processor status, and restoring the process later without loss of continuity
- High-speed cache memory for improved performance
- Hardware for enforcing several modes of memory access
- Hardware for implementing data integrity and security mechanisms
- Associative memory for fast hardware access to virtual memory
- Program-addressable registers for preparing virtual memory addresses

Processor Organization

The Multics processor module is organized around functional units:

- *Control Unit* – provides the interface between the Operations Unit and the system controllers
- *Operations Unit* – Contains the logic to execute binary arithmetic and logical functions
- *Decimal Unit* – includes an Extended Instruction Set (EIS) within the processor's basic repertoire of instructions, including instructions for processing character string, decimal data, and bit strings
- *Appending Unit* – implements segmentation and paging of the virtual memory; provides 24-bit addressing; contains 16 segment descriptor words and 16 page table words on a most recently used basis; and provides a descriptor segment base register, eight segment pointer registers, and ring protection hardware
- *Cache Memory Unit* – holds the most recently used information from main memory and improves system performance by reducing instruction and data fetch time

Processor Modes of Operation

The processor operates in three modes: absolute, privileged, and nonprivileged. All instructions are available in the absolute mode. Privileged instructions, such as those that operate on the descriptor base register and input/output devices, are available only in absolute and privileged modes. Most, but not all, of the instructions are available in nonprivileged mode. General users are restricted to the nonprivileged mode and thus are prevented from executing any instructions that could interfere with other programs or with the Multics system software.

The full segmentation and paging capability of the processor is used in the privileged and nonprivileged modes for fetching instructions and operands. Addressing in the absolute mode does not use the segmentation and paging capability and is not generally available to user programs.

Segmentation

Segmentation divides the user's address space into many parts and assigns attributes (access control and length, for example) to these parts based on their logical use. Like a conventional file, a Multics segment is a collection of instructions or data specified by the user. It has a symbolic name and access control list and can vary in length. A segment can be addressed directly, as memory can, and does not have to be read or written record-by-record as a conventional file would.

The segment is the basic unit of information sharing. Different users can incorporate a single segment into their programs merely by specifying the segment name. A program doesn't need to copy a segment to use it, saving time and eliminating duplication in main memory. To control this sharing, each segment has an access control list containing the name and access privileges of each person who can use the segment. The hardware checks these access privileges on each reference to a segment by any user.

Certain segments, containing only instructions or constants, are especially easy to use and efficient in occupying storage space. These "pure-procedure" segments store their data and control variables separately. Pure procedure segments do not modify themselves; they are also totally reentrant. More than one user can use the same reentrant procedure simultaneously without having to copy it. This enhances sharing and saves storage space. The Multics operating system, compilers, and application programs all utilize pure procedure. The major benefits of segmentation are:

- Stored data and procedures can be referenced easily and directly
- Logical units such as programs and data are protected by hardware
- Users can directly share procedures and data bases

Paging

Segments can be of different sizes, and their sizes can change during the operation of a program. In order to simplify allocation of main memory, each segment is divided automatically into fixed-size storage units called pages. This division – and the subsequent manipulation of the pages – is totally transparent to the user and requires no action on the user's part. In addition, any access controls established for a segment apply to the pages that make up that segment.

The pages of a given segment need not be located in contiguous storage blocks. They do not even have to be in main memory all at once. As a page is needed in main memory, it is retrieved automatically from secondary storage and placed in any available block in main memory. When main memory is filled and more pages are needed, some pages have to be displaced. Pages not used recently will be moved (swapped) to secondary storage. (For added system efficiency, pages that are part of a pure-procedure segment or have never been written into do not have to be swapped out, since a copy still exists in secondary storage. These blocks of memory can simply be cleared and overwritten with other pages.)

Paging has distinct advantages:

- A user can write and operate a program without planning for its storage allocation needs or for the management of the segments.
- Paging provides a simplified technique for dynamic storage management and reduces operating system overhead by allowing optimum loading of main memory and avoiding compaction problems.
- Paging uses the system's high-speed storage effectively by fetching only pages that are actually referenced, rather than an entire program or file.

Ring Structure

The Level 68/DPS ring structure extends the concept of a two-state machine (i.e., master-mode and slave-mode) to a multi-state machine. Level 68/DPS provides eight states of execution with adequate tools to allow proper administration of access privileges to the system users within them. This implementation allows segments to be grouped into rings. The number of each ring (0-7) designates the level of privilege assigned to procedure segments executed in that ring. Ring 0 has the highest level of privilege. Privileged ring segments, such as the supervisor and special user subsystems, are protected from uncontrolled use by less privileged rings. These segments can only be used by procedures in less privileged rings if called via a special "gate" mechanism. The access permission checking is still required as well.

The ring structure, with its obvious applications to information protection and security, is an integral part of the paging and segmentation hardware. The ring structure offers users numerous benefits, for example:

- Users can create protected programs and data bases for controlled use by others
- A supervisor program can be implemented in layers with differing degrees of privilege
- A programmer can debug a program in an unprivileged environment and then move it to a privileged environment with no recompilation or modification

SYSTEM CONTROL UNIT

The system control unit (SCU) is the principal interface between all central system components. It provides complete system interrupt control which regulates communication between components and services all demands on memory under priority control. The system control unit handles the switching of all control signals, addresses, and data into and out of the memory units. Memory units are modular and each connects directly to a system control unit.

Up to eight system control units may be configured in a Level 68/DPS system, with each SCU capable of controlling one megaword (four megabytes) of memory. However, note that maximum memory size currently available is four megabytes.

Additionally, the SCU checks integrity on all data and control paths to and from memory units as well as the paths to and from the other system components. It also provides memory configuration switching.

The system clock within the SCU a 52-bit binary counter that increments at one-microsecond intervals – is used as a calendar clock. The 142-year capacity of the clock makes it possible for Multics software to operate on a consistent time base.

MEMORY

The metal-oxide semiconductor (MOS) memory of the Level 68/DPS Systems can range from 512K words to four million words. Two words, plus Error Detection and Correction (EDAC) bits, are accessed in each memory cycle.

INPUT/OUTPUT MULTIPLEXER

The input/output multiplexer (IOM) interfaces the system control units with the peripheral units and integrated network processors. The IOM can operate many different types of devices. It is controlled by information stored in memory.

The IOM can transfer data between I/O devices and memory while processors continue to run programs. I/O transactions are controlled by lists of control words prepared by the Multics operating system and stored in memory. When an I/O transaction is complete, or when special conditions are detected, the IOM causes a program interrupt.

The IOM has attractive performance characteristics:

- Peak IOM transfer rate of more than four million bytes per second
- Up to 56 simultaneously active data channels per IOM
- Peak channel transfer rates of more than one million bytes per second
- Scratchpad storage for control words
- Eight special channels for specific system functions

The IOM offers complete memory protection for all I/O data transfers. Each data channel functions independently, with its own memory assignment. Parity is generated and checked on all information sent to and from the system controllers and the peripheral subsystems.

PERIPHERAL SUBSYSTEMS

The Level 68/DPS mass storage subsystem uses the freestanding MSP0603 mass storage processor to control up to 32 mass storage units. Multiple mass storage subsystems can extend the virtual memory size to a maximum of 512 mass storage units. Three different devices are supported; the MSU0402 (78 megabytes), MSU0451 (157 megabytes), and the MSU0500 (626 megabytes). These devices can be intermixed on the same mass storage processor.

The MTP0601 Magnetic Tape Processor supports the MTU0400, MTU0500, and MTU0600 tape units. These devices range up to 200 inches per second and 1600 bits per inch. Other peripherals include the PRU1100/1200/1600 ASCII line printers (1100, 1200, and 1600 lines per minute, respectively), 1050-card-per-minute readers, and 100- to 400-card-per-minute punches, all under the control of the Unit Record Processor (URP). Up to eight individual devices can be controlled by a single URP.

NETWORK PROCESSOR

The Integrated Network Processor (INP) controls all remote terminal interaction with Level 68/DPS host system. Connected to the central system via an IOM, the Integrated Network Processor provides the various interfaces required by the elements and protocols of a distributed system as well as a facility for dialog with the host system. By performing the tasks of message management and message handling, the INP frees the host for other processing functions. The resources of the central system are called upon only when the message is submitted for information processing. However, some networking functions (e.g., a message switch) can be accommodated by the INP without any involvement of the host processor.

TERMINALS SUPPORTED

Level 68/DPS systems can communicate with various types of terminals, including the following:

Interactive Devices

Honeywell VIP 7105, 7205, 7705, and 7800
Teletype Models 33, 35, 37, 38, and 40

IBM3270
IBM2741 and 1050 (EBCDIC and Correspondence)
IBM 1050
IBM 2780
Trendata Models 1000 and 4000
Datel 30
Dura 1021
GE TermiNet 300 and 1200 (up to 1200 bps both half or full duplex)
Execuport 310C and 320C
Texas Instruments Silent 700 Series
Adage Inc. Advanced Remote Display Station
IMLAC PDS-1D Graphic Display Computer
Tektronix (graphics devices to 9600 bps)
DIGI-Log Telecomputer Model 109
Data Products Portacom
Computer Devices Incorporated Teleterm 1030 (including ASCII/APL models) 1132, 1203
Teleray Model 3711
DEC GT40 Display Processor, DECwriter II, DECwriter LA36 (300 baud ASCII)
DEC Graphics Models 12 and 15
Hazeltine 2000 (ASCII)
Delta Data Systems (Alphanumeric CRT up to 2400 bps)
Xerox 2700 (Diablo printer with plotting capability)
Anderson-Jacobson Models AJ630 (ASCII), AJ832, and AJ841
Gen Com Systems GSI300 (300 baud ASCII) and GSC 300Q
Lear-Siegler ADM-2 Display Terminal
ADDS Consul Model 980
DTC300 Series
Infoton Vistar/II
Bedford 575 (Selecterm)
BeeHive Super Bee

Remote Job Entry Devices

Absentee (batch) prọcessing is supported at the central site and at remote sites. Remote batch terminals such as the Honeywell Model G115, Data 100/78 (using Honeywell Model G115), RNP702, IBM 2780, Honeywell Level 6, and Mohawk 2400, offer remote bulk I/O capability and remote job entry.

RECONFIGURATION

The memory modules, central processors, mass storage devices, and terminals in a 68/DPS system can be reconfigured dynamically, without interrupting user service. This allows failing devices to be removed from processing for maintenance and reconfigured automatically following repair. In addition, failing memory pages are automatically deallocated whenever a double-bit (uncorrectable) error is discovered. Large configurations can also be split into smaller separate systems for block time processing or testing without service shutdown.

SYSTEM CONFIGURATIONS

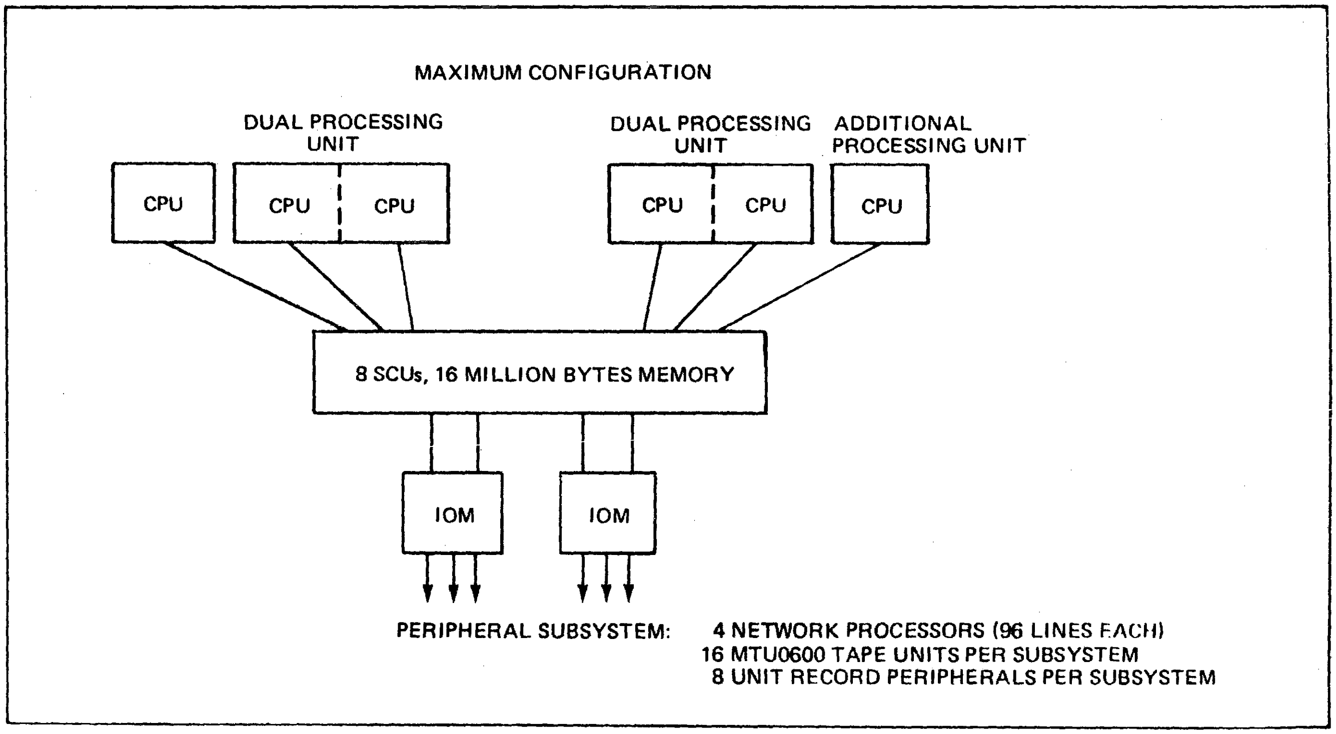
System configurations can be tailored to user requirements. An entry-level configuration consists of:

A Level 68/DPS System Control Unit with 512K words of memory
An Input/Output Multiplexer
An Integrated Network Processor
An MTP0601 Magnetic Tape Processor with a minimum of two MTU0500 tape units
An MSP0603 Mass Storage Processor with two MSU0402 Mass Storage Units providing 156 million bytes of storage
A printer
An operator console
A full selection of terminals

The system can be significantly expanded to a maximum complement of equipment:

Level 68/DPS with power options to 4.3 times processing power of entry systems
8 System Control Units with a total of four million words of memory
2 Input/Output Multiplexers
4 Integrated Network Processors
32 MSP0603 Mass Storage Processors and 512 MSU0500 Mass Storage Units providing 300 billion bytes of storage
MTP0601 Magnetic Tape Processor and 16 MTU0600 Magnetic Tape Units (per subsystem)
8 unit macro devices (per subsystem)
An operator console
A full selection of terminals

Specifications may change as design improvements are introduced.


MAXIMUM CONFIGURATION
DUAL PROCESSING UNIT
DUAL PROCESSING UNIT
ADDITIONAL PROCESSING UNIT
CPU
CPU
CPU
CPU
CPU
CPU
8 SCUs, 16 MILLION BYTES MEMORY
IOM
IOM
PERIPHERAL SUBSYSTEM:
4 NETWORK PROCESSORS (96 LINES EACH)
16 MTU0600 TAPE UNITS PER SUBSYSTEM
8 UNIT RECORD PERIPHERALS PER SUBSYSTEM

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

20994, 3678, Printed in U.S.A.

Honeywell

SERIES 60 (LEVEL 68/DPS)

Distributed Processing System: Multics Overview

Multics, one of the most powerful and comprehensive large-scale systems in the world today, provides general purpose data processing service for users dealing with challenging business and scientific problems. Based on the concept that the computer is only as productive as it is accessible, Multics offers a broad range of features and capabilities within a service-oriented environment and addresses the reliability, availability, and system growth requirements of distributed processing users.

PROCESSING CAPABILITIES

Multics is for large and small users ... experts and novices. Its total online, interactive orientation is ideally suited to a variety of processing activies including:

BATCH Multics supports both local and remote batch processing. Interactive users can submit batch jobs for execution; batch jobs can also initiate other batch jobs. Jobs written for batch execution can also be run interactively without change.

REMOTE JOB ENTRY Multics supports numerous devices for remote job entry. These include: Mohawk Data Sciences 2400, DATA 100 Model 78, IBM 2780, Honeywell Model G-115, Honeywell RNP702, and several Honeywell Level 6 models.

TIME SHARING Full time sharing capabilities are available on native mode, interactive Multics. In addition, two other environments, the Multics FAST and DFAST subsystems, provide the user with varying levels of time sharing power and processing performance.

TRANSACTION PROCESSING Multics transaction processing offers flexibility and scope unmatched in other systems. Terminal oriented, the system doesn't require special executive programs to monitor terminal input and then to process user requests in batch. Applications can be written in any language and directly accessed from any number of terminals or batch jobs ... simultaneously ... in a completely shared environment. System facilities allow interfaces to specialized data bases as well as concurrent access control, journalization, recovery, and online forms generation.

WORD PROCESSING Multics' advanced word processing facilities include:

- Powerful text editors
- Document formatting capabilities
- Error detection tools
- SPEEDTYPE (shorthand for typists)
- Online dictionaries
- Artwork macros
- Electronic mail

Together, these features enable the Multics user to create and maintain error-free documents and produce formatted output.

GRAPHIC PROCESSING A general purpose interface enables user and application programs to create, edit, store, display, and animate graphic material. Multics' graphic features include:

- Terminal independence
- Powerful editing capabilities
- Permanent storage capabilities
- Sharing subobjects and structures
- Dynamic animation
- Local editing
- Incremental picture updating

REAL TIME PROCESSING Multics can provide real-time response to specified users or jobs (deadline job scheduling, for example). Thus, the system is useful in operations control functions such as process control monitoring.

USER ORIENTATION

Uniform User Interface

Key among Multics' unparalleled accessibility features is its uniform user interface. All the system's processing functions from batch to time sharing to graphics are available via a single, consistent interface. There are no format or execution differences between usage types. A program written in an interactive environment will run in batch *without conversion or modification*, and vice versa. Any terminal attached to the system can, unless specifi cally restricted by the system's administrators, access any program or feature.

Also implicit in the Multics environment is the concept of total compatibility. There are no restrictions on languages used to access data bases or files. Programmers can develop COBOL applications, for example, with certain modules written in other languages, and be assured of total compatibility and equal processing efficiency. Moreover, data and programs associated with one processing dimension (e.g., transaction processing) are totally accessible from any other dimension.

Total Online Orientation

Multics' architecture is oriented directly toward online applications. Whereas other systems provide online capabilities through executive packages, Multics is completely interactive, and does not use such packages. As a result, in high-volume transaction-oriented environments where there is significant terminal activity, Multics provides unique advantages over other architectures.

Total Sharing

Multics permits the controlled sharing of operating system software and libraries, language processors, data bases, and user code and data. Even with multiple users simultaneously compiling COBOL jobs, for example, only one copy of the COBOL compiler is in use. And since all Multics language processors generate reentrant code, even users' programs can be shared without special programming.

EASE OF USE

From its inception, Multics has incorporated features and capabilities that make it one of the most accessible and easy-to-use systems on the market today.

Interactive Orientation of All Facilities

Every aspect of Multics is online-oriented, including the language processors, applications, data base management facilities, utilities, administrative tools, and metering and tuning capabilities.

No Job Control Language

Unlike other systems – which require that the user learn a job control language (JCL) prior to running a job – Multics provides control functions via a standard command processor approach, thus eliminating complex JCL. System commands and routines supply the logical branching, conditional execution, file system, and I/O control required to direct a job through simple and complex execution paths. Thus, the Multics user need not learn a new batch interface, or become a JCL expert to use the system for problem solving.

Flexible Environment Shaping

Multics matches the computer's processing environment with the user's particular needs. Providing a flexible interface, Multics enables users to continue using the computer in the way they are accustomed even though they are changing to a new system. The concept of shaping environments is particularly beneficial for data processing organizations that service many diverse user groups simultaneously. Each group can develop – using standard administrative tools specialized interfaces to satisfy unique operating requirements. Typical of these tools is the special command processor with which users can define their own abbreviations for frequently-used command lines or sequences of commands, as required.

EXEC-COM Facility

With this feature, users can write programs to execute stacked command lines. The additional control capabilities provided by the EXEC-COM facility allow for logical branching, the maintenance of variables, command-level I/O, and conditional execution. Thus, it is possible to develop routines for functions which typically require higher level languages without having to actually use such languages.

Help Files

Multics help files contain printed text which provides immediate, online assistance to users requesting data on various system topics, including the use of Multics commands and subroutines. Tutorials on each system feature are thus readily available. Similarly, users can document their own programs and routines, and avoid continuous referral to cumbersome hard-copy documentation.

Memo Facility

The memo facility allows users to indicate when specified events are to occur on the system. Events to be scheduled can be inter-job signals, messages, or program executions. Typical examples include simple date reminders and the automatic scheduling of an installation's batch applications.

Intelligent Defaults

Multics – via its intelligent defaults lets users perform certain functions on the system without having to consider concepts that do not pertain to their particular application. In a typical situation where the user wishes to archive data on tape, numerous details such as tape density and blocking factors must also be defined. For users to whom these details are not relevant, Multics standard defaults permit the archive tape to be written via simplified I/O routines from command level.

VIRTUAL MEMORY

The Multics virtual memory implementation, totally invisible to user programs, means that programmers can concentrate on problem solving rather than be concerned with real memory constraints and memory management (e.g., partitions and overlays).

Virtual memory is limited only by the amount of available mass storage. So, programs written for the largest configuration can run on the smallest without modification. No standard, fixed main memory-to-mass storage ratio is required to ensure efficient system operation. Regardless of the ratio, Multics can be tuned to perform within its real memory constraints.

Segmentation

Multics organizes information into segments logical, named units containing data, programs, or directories of other segments. The segments and their directories form a uniform file system for all users, the administrative and accounting system, and the system software itself. Segments can increase dynamically up to 1 megabyte in length, and files can span nearly 1000 segments.

Paging

Multics designers devised the paging concept to avoid the system performance limitations that result from swapping large files (segments in the case of Multics) in and out of main memory. In Multics, segments are subdivided into "pages" (4096-byte blocks). Address mapping at the hardware level enables the system to determine if the page of a requested segment is in memory and if not, to locate that page, transport it to memory, and schedule it for execution by the waiting process. Demand paging eliminates space allocation and compaction problems and maximizes system performance. Paging is completely transparent to the user. And only those pages required for the execution of a program are brought into memory at any given time.

The actual movement of information in and out of main memory is completely automatic and transparent. Data required by the user, for computation or manipulation, is retrieved from peripheral storage and inserted in main memory without the user ever knowing the transier took place.

PROGRAM DEVELOPMENT

Multics ranks as one of the industry's premiere software development tools. It has powerful source code manipulation techniques for entering, editing, and archiving code, and for automatically structuring programs for easy reading and use. Its online debugging tools facilitate checking out new code, and aid in the fine tuning of programs. Multics also provides options for dynamic linking or prelinking of programs, and standard calling sequences for system libraries and user programs.

Multics' fully compatible language processors significantly contribute to the system's outstanding program development capability. These processors including PL/I, COBOL-74, FORTRAN, APL, ALM (assembler), and BASIC can be fully shared. Because of their compatibility, programs written in APL, for example, can call those written in PL/I or FORTRAN. Compatibility is restricted only by the data types supported by each language. And since all Multics compilers generate reentrant code by default, all user programs are shared; no special coding procedures are required.

APPLICATIONS DEVELOPMENT

During applications development, the programmer typically must address concerns such as terminal control, data base management, interfaces to system functions, data security and integrity, and I/O interfaces. Multics provides a standard applications environment that can be shaped to individual needs. Thus, programmers can avoid these problems and concentrate on programming, thereby significantly shortening the development cycle.

DATA BASE MANAGEMENT

The Multics Data Base Manager (MDBM) offers two data base management interfaces: Multics Integrated Data Store (MIDS), a subset of I-D-S/II; and Multics Relational Data Store (MRDS). MIDS supports schema/subschema data base definitions to provide data and program independence. Data base structures which can be developed include sequential, network, hierarchical, or cyclical. MRDS provides data and program independence via model/submodel data base definitions and nonprocedural user retrieval and update mechanisms. Both interfaces allow interactive or batch usage, sharing, concurrent access, and access via programs written in any language available on Multics.

ADMINISTRATIVE CONTROL

A significant strength of Multics is its ability to provide service to a wide variety of users simultaneously without the workload of any one group adversely affecting that of another. Multics' comprehensive set of administrative controls make this high level of processing service possible. These controls include:

- Decentralized control options
- Guaranteed resource allocations
- Priority scheduling with specified response characteristics
- Deadline job scheduling
- Flexible service pricing
- Automatic or on-demand billing
- Automatic user or project cut-off when resources are expended
- Online metering and tuning
- Standard environment shaping

EASE OF OPERATION

The operational features in Multics make its utility-grade service readily available to a wide variety of users. For example, there is *no* system or library generation or edit. The operator can start the system simply by typing one command at the console. System software and libraries -- delivered patch-free -- can all be updated online. A new compiler or application can be added without shutting the system down, and with only one command, new software can be installed without affecting users working with other existing packages.

Multics can run unattended. An automatic reboot feature automatically restarts the system in the event of a failure. The system's online test and diagnostic capabilities permit the user to check out a malfunctioning component online, remove it from service if necessary, and dynamically reconfigure the remaining system components ... all without interrupting user service.

File integrity is fundamental in a service-oriented environment. Multics offers automatic mechanisms which optionally journalize recent file updates within the system's virtual memory. Should a failure cause temporary loss of data, these journals can be reloaded to continue service. Duplicate file copies are not necessary since Multics itself provides file backup.

Additional ease-of-operation features include:

- Online administration and billing
- Operation from any terminal
- Dynamic control of the priority scheduler

COMPATIBILITY WITH LEVEL 66

Level 66 and Level 68/DPS systems share a high degree of compatibility. In their respective hardware configurations, only the central processors differ. The Multics central processor is a superset of the Level 66 unit and has a switch which allows it to run Level 66 GCOS, thus providing users of both systems added flexibility and backup.

A special Multics subsystem called the GCOS Environment, allows GCOS job decks or IMCVs to be run without change. GCOS files can also be transferred between the two systems using standard GCOS tapes.

Multics, through the GCOS Environment, also supports several languages that run under GCOS, including JOVIAL, ALGOL, COBOL-68, GMAP, and FORTRAN-Y.

DISTRIBUTED SYSTEMS ORIENTATION

Multics is ideally suited to the development of networks of distributed mainframe systems. With existing hardware and software, Multics can be interconnected with a variety of other systems to form networks, that address a broad range of user requirements.

With this approach, users establish networks in which files, programs, and data can be shared among the various systems easily and with maximum security. For example, a programmer could log onto an IBM 370/158, and then have required data transmitted from a CDC-7600 to a Multics system for processing, with the output going to the 370. Numerous variations of this approach are possible. In fact, Multics can currently interface with systems such as IBM 360/370s, Burroughs 4700 and 6700s, CDC Cybers, and Univac 1100/90s.

SECURITY

Multics offers high levels of security unattainable on other systems today. Several elements cooperate to make this outstanding system, data, and program protection possible. These elements are passwords, access control lists, multistate ring protection mechanisms, and access isolation methods. No special coding is required to make use of any of these elements; only standard system commands need be executed.

Passwords

The Multics password mechanisms control access to the system and verify users' identities. Each user has a system-maintained password which can be changed at any log-on. Passwords are stored in encrypted form so that a user's password will not be revealed accidentally. When the user attempts to log on the system, the typed in password is encrypted and verified by comparison against the one stored in the system. Precautions are taken to ensure that passwords are not exposed during the log-on sequence.

Access Control Mechanisms

An Access Control List (ACL) is used to control access to every segment in the storage system. Through the ACL users can grant specific access rights (e.g., read, write, execute) to individuals or groups of users. System hardware enforces access control during the execution of each individual machine instruction.

Ring Protection

A special hardware implementation, the Multics ring structure is a multilevel approach to data and program access control. The ring structure contains eight levels of execution (rings 0 through 7; 0 being most privileged, and 7 the least). Within this structure, users can access information only in those segments at the same level or higher (less privileged) than the current state in which they are executing. The Multics operating system resides in the most privileged ring (0) while users generally execute in the less privileged rings.

Access Isolation Mechanism

The Access Isolation Mechanism (AIM) incorporates administrative controls to grant or deny access to information in the data base. AIM is a way of organizing users into groups among which communication can be restricted or denied. Like all other types of Multics access control, AIM is initially verified by Multics software and is hardware-enforced at every reference thereafter. However, AIM also can prevent users from granting – to other users access to even their own information. AIM can be invoked or disabled at the discretion of each Multics site. In addition to administrative controls, AIM provides extensive security auditing controls to monitor user activity.

GROWTH

Like other Honeywell systems, Multics' hardware architecture provides the benefits of modularity and an easy, paced growth that does not require swapouts to upgrade to higher performance levels. This easy expansion allows a user to configure the exact system needed and protects equipment investment. It is not necessary to change the operating system, system libraries, or user codes in order to move to a more powerful Multics system.

SAMPLE SYSTEM REQUIREMENTS

A typical entry-level Multics configuration consists of the following components:

- Central System, with 512K words of memory (2 million bytes)
- One System Control Unit
- One Input/Output Multiplexer
- One Integrated Network Processor
- Two MSU0402 Mass Storage Units
- A minimum of two MTU0500 Magnetic Tape Units

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

19626, 11277, Printed in U.S.A.

DF51, Rev. 1

Honeywell

LEVEL 68

Multics Virtual Memory and Storage System

The Multics Storage System is a modular, hierarchical file system augmented by a comprehensive virtual memory. An integral component of the Multics Operating System, the storage system is just another reason why Honeywell believes Multics to be the most advanced computer system available.

HIERARCHICAL STORAGE SYSTEM

In the Multics System, all information is grouped into segments, collections of instructions and/or data associated with a particular name. All of the segments are stored in a tree structured hierarchy (see illustration), the beginning of which is called the root. The branches emanating from the root lead to either nondirectory or directory segments.

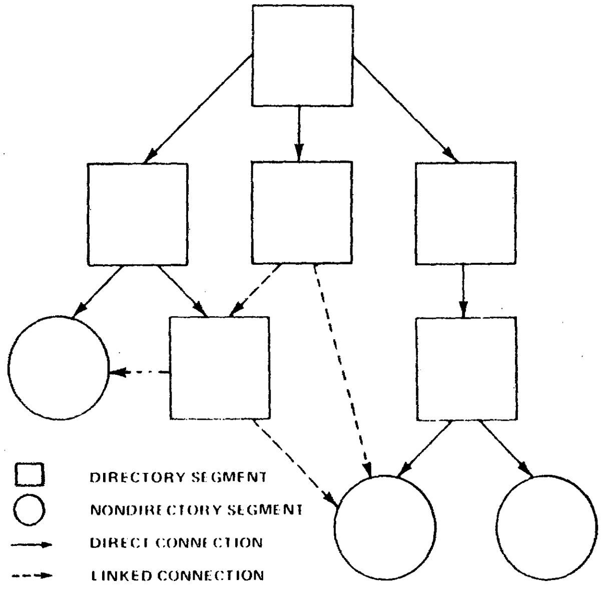
The Directory Concept

The sole function of a directory segment is to catalog the segments residing below it in the tree. Each directory contains the names of the subordinate segments and lists their attributes including length, virtual memory address, date and time the segment was created, list of users allowed to access the segment and with what access mode (read, write, execute, or null).

The directory concept is the key to several Multics features, including storage structure, administrative control, access control, search rules, and naming conventions. For example, all users registered on the system are grouped into projects. Each project has a directory, and each user in that project has his own directory subordinate to the project directory. A user may create additional subordinate directories under his own directory or under directories to which he has been granted specific access. He may also create "links" in his directory to segments to which he has specific access. This capability is often used to share data and/or programs.

The Segmentation Concept

All information within the storage system is stored in the form of segments. Provided the user has the proper access rights, all information is directly addressable. In addition, all of the information within the storage system is placed within the Multics Ring Structure (see "Controlled Access to Segments") a hardware-software feature that provices maximum security over information sharing.



Hierarchical Storage System

Each segment is identified by a user-assigned symbolic name (making use of the full ASCII character set) as well as by a unique, system-assigned identification.

The fully specified name of any one segment is the list of subnames that reflect that segment's position in the directory hierarchy with respect to the root directory. This name, called a pathname, shows the "path" from the root directory to the specific segment and is the symbolic name by which the user must reference the segment.

VIRTUAL MEMORY ENVIRONMENT

Segmentation

In the Multics System, all segments are directly addressable by the hardware. With the addressing scheme used in the Multics processor, all references to information are mapped through descriptors (Segment Descriptor Words). These descriptors are listed in a table (Descriptor Segment Table) that identifies segment attributes and defines the access a user can have to the segments. Most importantly, segmentation encourages users to view memory as a collection of independent linear core memories, each associated with a descriptor.

A user program can create a segment by issuing a call to the system specifying as arguments the symbolic name of the segment plus additional information about who may or may not access this segment. Then the system constructs a descriptor according to the access information given by the originator of the segment.

The originator has control over every segment in his directory; he can grant or restrict access to these segments in any way he chooses. In fact, he can grant different access privileges (read, write, execute, or null) to different users of the *same* segment.

Once the segment is created, the user program can address any item within the segment using *name, i* where "name" is the symbolic name of the segment and "i" denotes the place of the desired item within the segment. The maximum segment size is 256K words (1,048,576 bytes).

Paging

With most computer systems, the limiting physical resource is main memory. The amount of main memory online is a major factor in determining the performance of a system. The problems associated with "swapping" large files into and out of main memory severely limit system performance. Even if files were not all large, there would still remain the difficult problem of core management. Since the Multics System allows users to create and/or manipulate large segments, it is neither feasible nor desirable to have an entire segment in main memory when in use.

Therefore, in the Multics System, segments are automatically subdivided by hardware into "pages" with a fixed size of 1024 words. Additional address mapping at the hardware level allows the system to determine whether or not a page of a referenced segment is in main memory.

If the page is not in main memory, a missing page exception occurs (called a "page fault"). The software system intervenes at this point and processes the page fault by locating the desired page in the storage system and transferring it into main memory. During this phase, the process that generated the page fault relinquishes control of the processor and the system dispatches the execution of another process (process multiplexing). Once the page does arrive in main memory, the system notifies the "waited" process and schedules it for continued execution.

By using this "demand paging" technique with a fixed page size, space allocation problems are simplified, and the cost performance factor of the system is significantly enhanced: only those segment pages that are currently needed are in memory at any one time.

USER ACCESS

Direct Access to Files

With most large-scale computer systems, the programmer must interface with the file system through complicated Job Control Language (JCL). He also must know the specifications of the storage device on which his files reside and must issue and control (either explicitly or through macro specification) his own I/O requests.

The Multics System, with the aid of the virtual memory, requires no JCL, nor are users concerned with or even aware of where and on what devices their segments reside. Instead, users communicate with the storage system by asking the system to make available to them a segment within their own virtual memory.

Controlled Access to Segments

Since all information is stored online in the Multics System and data can be accessed directly, access limits and controls are mandatory. The access control mechanism in the Multics System is a highly sophisticated and reliable means for specifying the usage attributes of directories and other segments. There are two access control lists recognized by the system: one for directory segments, one for nondirectory segments. These access control lists are carried in directory segments.

Access modes for nondirectory segments are:

read (r) data in the segment can be read

write (w) data in the segment can be modified

execute (e) an executing process can transfer to, and execute instructions in, this segment

null (n) access to segment is denied

Access modes for directory segments are:

status (s) the attributes of segments, directories, and links contained in the directory can be obtained

modify (m) the attributes of existing segments, directories, and links contained in the directory can be changed or deleted

append	(a)	new segments, directories, and links can be created in the directory
null	(n)	access to directory is denied

Access validation is checked and enforced at the hardware level on each memory reference. For example, if the originator of a data base decides to grant access to some user, he can issue a Multics command specifying "read" access for that user. If that user is currently attempting to reference the data base, he is given access instantly.

As a further control to accessing segments, the Multics System uses the Ring Structure. Logically, the Ring Structure is eight concentric rings, each representing a different level of virtual memory access rights. The highest level of privilege is the innermost ring, designated as ring zero; the outermost is ring 7. Each ring is protected against uncontrolled access by programs in any ring with a higher number designation and thus a lower level of privilege. The Multics ring-handling mechanism is enforced at the hardware level.

USER INTERFACE

In using the Multics Storage System, the user has two available interfaces. With the first, the "command level" interface, the user can create and/or manipulate segments residing in various user directories. In many time-sharing operating systems, the command level interface is supported by the physical editing of the "command" into various system tables in the operating system at system generation time. With this technique, the problems encountered while trying to expand the repertoire of the command language can be extreme.

In the Multics System, however, there is no "command language"; everything executed at Multics command level is simply an object program from a system directory. When a user types a command at his terminal (e.g., create xyz), the system first interrogates the user's own directory to see if the "create" program exists. If so, that version is executed. If not, the Multics System searches a set of system/user-supplied directories (called Search Directories) for the "create" program. The user also has the ability of expanding the default search directories to include other directories (possibly common to a specific project).

The second user interface is available via program execution. Users' object programs can issue call statements to create and/or manipulate segments. The call statements themselves are of the standard PL/I form. Any program executed while at Multics command level can be called and executed internally from within an object program.

FEATURES

Various features of both the Multics Virtual Memory and the Storage System are summarized below.

- All information stored in the Multics Virtual Memory is directly addressable provided the user has proper access rights.
- Established protection rings allow users to effectively partition data within concentric ring structures.
- Two or more users can share a single copy of data and/or programs in main memory.
- Users may create "links" to other accessible segments of virtual memory in order to share data and/or programs.
- Each user can specify various access rights to his own segments, even specifying different access to different users of the *same* segment.
- Movement of data between main memory and secondary storage and back is automatic, and is of no concern to the user.
- Tree-structured storage hierarchy offers an organized scheme of classification and facilitates efficient search for a particular segment.
- Use of directories within the storage system serves as a convenient place to look up addresses and access rights of other segments.
- Access rights to a segment are checked by Multics hardware on every reference to the segment.
- Multics has no Job Control Language; everything executed at command level is merely a standard object segment. Thus, the "command language" can be dynamically developed, expanded, and tailored to individual installation even individual user requirements.

SYSTEM CONFIGURATION

The functions herein described are applicable to any Level 68 configuration.

Specifications may change as design improvements are introduced.

The Other Computer Company:

Honeywell

HONEYWELL INFORMATION SYSTEMS

8743
7.5774
Printed in U.S.A.

In the U.S.A.: 200 Smith Street, MS 061, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale Ontario

AK57, Rev. 1

Honeywell

SERIES 60
LEVEL 68/DPS

Administration and Operating Features

The Level 68/Distributed Processing System, with its Multics operating system, is an advanced large-scale computer system providing general purpose data processing service for business and scientific users.

Level 68/DPS administration and operating features make its processing capabilities available with simplicity and ease. These features give the Level 68/DPS user more efficient control over online applications, improved response to individual user and group needs, and optimal utilization of all processing-related resources.

INTERACTIVE ARCHITECTURE

Level 68/DPS architecture is specifically directed toward interactive, online applications. It provides common interfaces for the implementation of these applications and for their administration and control as well. In addition, special online executives are not required to take advantage of these interfaces. Level 68/DPS employs standard, built-in tools for controlling the use of all system resources.

MODULAR GROWTH

Because of the Level 68/DPS hardware modularity, its users benefit from a long configuration life span, free from disruptive programming or operational changes. Level 68/DPS offers growth in small, calculated steps, paced with the growth of the user's workload. Users can add more memory, new processors, or front-end networking facilities according to specific workloads. Level 68/DPS uniformity and consistency make it possible to grow from the smallest configuration to the largest without changing the Multics operating system, libraries, or user programs.

DECENTRALIZED ADMINISTRATION

Fundamental in the Level 68/DPS design is the concept that productivity is tied directly to accessibility. The system's approach to administration is consistent with that concept. Level 68/DPS administration is decentralized, so that specific resources can be allocated to specific projects and accounted for accordingly. The project administrator can in turn allocate these resources to individual users within the project as necessary.

Allowing remote users and user groups autonomy in their use of the system, and enabling them to control their own resources, gives them the opportunity to more effectively deal with their day-to-day problems and varying processing requirements.

Resource billing flexibility illustrates the advantage of Level 68/DPS decentralized administration. This billing can be automatic or on demand. Users on individual projects can even do their own sub-billing, substitute their own billing algorithms, use different algorithms for different users, or install new algorithms dynamically.

The Level 68/DPS administrator can establish up to eight separate work shifts, with different rates applied to each, encouraging use during slack periods. Users can also be restricted to working on specified shifts, and these restrictions can be changed dynamically. Results: more efficient load leveling and more effective use of resources.

RESOURCE UTILIZATION CONTROL

Resource control in Level 68/DPS involves three primary areas: online storage utilization, physical access to system communication lines, and user job prioritization. Standard, built-in tools are used in each area.

Online Storage Utilization

Level 68/DPS enables an administrator to control the use of online storage on a "per project" basis. To maximize the use of storage resources, the administrator can allocate this storage to individual users within a project.

Two techniques illustrate this control:

- A storage quota, which prohibits the allocation of storage space when a specified workload limit is reached.
- An automatic data migration function, which allows data to be transferred to another medium when that data has not been accessed for a certain period of time.

Physical Access

Controlling the physical access to Level 68/DPS, which includes restricting user groups to specified communications lines, also means controlling the workload mix on the system. The administrator can establish the maximum work units that the system will adequately support for the particular site. Issuing load unit weighting factors for each user ensures that the system's capacity cannot be exceeded to the detriment of its users. Different users can be issued different weighting factors to reflect different processing requirements. To ensure that specified privileged or high priority jobs will always be executed regardless of the total workload on the system, certain users can be given guaranteed access status.

User Job Prioritization

Dynamic Scheduling

User and job prioritization is accomplished via dynamic control of the priority scheduler. With this capability it is possible, for example, to add either more batch jobs or more interactive users to a system without negatively affecting the productivity of time sharing programmers or transaction processing activities already in execution.

Workload Balancing

Level 68/DPS also incorporates a workclass concept which further ensures that the workload of certain users will not negatively affect the workload of others. Users can be grouped into classes and allocated percentages of processing capability. Regardless of the total system workload, each group of users can be ensured a predetermined percentage of available central processor time. "Free time," any time not utilized by a given workclass, is made available to other workclasses requiring more than their allocated time, maximizing central processor utilization.

Deadline Scheduling

This feature can be used for a limited number of users or applications to ensure that a predefined amount of processing time is available to these users after an interaction or job submission. Deadline scheduling ensures that batch jobs will finish within a specified time period, and that interactive users will receive a response within a predefined time span after a transaction, regardless of the overall system workload.

Changes to allocated resources, response characteristics, and batch job scheduling can be accomplished dynamically, or scheduled to happen automatically at a predetermined time. For example, a key application can be deadline scheduled so that it is ensured a four-second response time during prime shift, but only an eight-second response time on second shift.

ENVIRONMENT SHAPING

Level 68/DPS environment shaping tools let you define interfaces to match users' application processing needs. These tools not only make it possible for nonexperts to use 68/DPS for problem solving, but also enable system administrators to succinctly define and limit what users can do with the system and what they must know to process their applications.

The two major environment shaping tools are the limited service subsystem and the closed subsystem.

- The limited service subsystem enables the project administrator to define exactly those functions which the user cannot perform; all other system functions are available to that user by default.
- The closed subsystem lets the administrator define all the functions which the user can perform; all other functions are unavailable by default.

The limited service and closed subsystems can also be used for defining new command languages and interfaces that convert "foreign" system commands into Multics operating system commands. This can be useful to first-time Level 68/DPS users. In addition, these subsystems allow nontechnical users to access the system via a highly simplified command interface. The capabilities afforded by these structured interfaces do not have to be programmed; they can be established via simple commands. The administrator need only create a new file with the new restrictions or guidelines in it, and it is immediately available to all authorized users.

SIMPLIFIED SYSTEM OPERATION

Level 68/DPS is surprisingly easy to operate, especially considering its size and power. This relative ease of use is due largely to its dynamic reconfiguration capability, simplified system maintenance, and responsive online metering and tuning capabilities.

Dynamic Reconfiguration

The dynamic reconfiguration capability of Level 68/DPS is used to maintain continuity of processing service in the event of a malfunction in a system component. If a failure occurs in a processor, for example, a single command automatically moves jobs and data in that processor to another unit, notifies the other system components of the malfunction, removes the processor from service, and reconfigures the remaining devices. Processors, memory, and peripheral devices can be added to or deleted from a configuration dynamically, via operator command. Failed pages of memory (4K-byte units) are automatically reconfigured by the operating system whenever a double-bit error occurs.

This reconfiguration process can also assist in the testing of certain components. Users can dynamically remove certain devices from service, submit them to test and diagnostic routines and reconfigure them automatically, without interrupting processing service. Alternatively, devices can be dropped from service on one system, configured as a separate system, used for testing of new software, and then reconfigured in the original system without shutting down service.

Level 68/DPS can run unattended, providing processing service on holidays and weekends, without the need for operators. During unattended operation, it can be run in the automatic reboot mode, and, in the event of a system failure, the system will automatically reinitialize itself so service can continue.

Level 68/DPS can be operated from terminals inside or outside the computer room. Due largely to the system's extensive security, an administrator with the proper authority can log into the system and issue any command that could be issued from the main operator's console. This gives Level 68/DPS greater operational flexibility, and frees it from relying on a single device (which might fail) for control functions.

Level 68/DPS provides batch job status and control capabilities as well as communication between system users. Operations messages can be sent to individual users; messages can be broadcast to the entire user community or to those on a particular project (or vice versa).

System Maintenance Features

For simplified maintenance, the Multics operating system does not require system or library generation or edit. Multics software releases are also delivered patch-free.

Users can easily install their own software or add programs to the time sharing library. New software can be installed without a system shutdown or link-edit. New compilers, applications, and procedures can be installed without interrupting processing service, even while users are using old versions.

Metering and Tuning System Performance

Level 68/DPS offers standard metering and tuning tools that allow an administrator to monitor and adjust system performance according to specific needs and changing workloads. Using data that is constantly being gathered by the system itself, administrators can retune the system, move users from one processing class to another, or even change the percentage of processing power allocated to a user, depending on any number of variables and requirements.

This control is possible because the administrator can analyze all facets of system performance from this constant flow of metering data. Some of the information supplied includes:

- I/O and device activity data
- Application response, processor utilization, communications, and I/O queuing
- Average CPU time spent on certain functions, or used by certain applications
- State and characteristics of the communications lines attached to the system
- EDAC (Error Detection and Correction) data associated with main memory hardware errors.

PRE-SCHEDULED ADMINISTRATIVE FUNCTIONS

A system facility called MEMO provides generalized capabilities for scheduling the initiation and execution of any program or set of programs. This allows the administrative or operational staffs to pre-schedule the running of administrative routines, such as billing. Scheduling can be self-repeating, such as scheduling a program to be run every Monday at 3 p.m. MEMO also allows either a precise definition of day/date/time (down to a microsecond level) or more general, logical definition ("on Tuesday").

INTEGRITY MECHANISMS

To maintain file integrity in the event of a malfunction, Level 68/DPS has powerful file back-up mechanisms.

- An automatic file archiver journalizes all changes to the file system and makes it possible to "roll forward" following a failure.
- Online file system integrity checks ensure a consistent, reliable file system.
- Main memory flush-to-disk is an automatic data protection mechanism invoked after a system service interruption. All data in main memory which has been modified is written out to mass storage to reflect all changes made up to the CPU cycles prior to the failure.
- Support of optional "shadow copy" logical volumes within the storage system. Volumes designated to have shadow copies automatically have spare devices assigned with copies of each physical volume. When updates are made to any of these devices, the copy is updated as well. If a device fails, the system automatically shifts to the copy without user interrupt.

Specifications may change as design improvements are introduced

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

22405, 1.51278, Printed in U.S.A.

Honeywell

Interactive Programming Environment

SERIES 60 (LEVEL 68/DPS)

To the user, a computer system is only as productive as it is accessible. Based on human engineering design concepts, the Level 68/Distributed Processing System (68/DPS) uses interactive, remote terminal access – the most natural and convenient mode for the user – as the primary mode for programming.

With the advent of Level 68/DPS, the computer is to be measured, not merely by hardware speeds, but by how well it helps solve a problem – from the very inception of that problem to its best solution. Rather than wait for computer availability in a batch mode and submit many sequential jobs, a Level 68/DPS user prepares, compiles, and checks out programs in one continuous interactive terminal session.

The Level 68/DPS interactive programming environment provides a complete range of facilities that satisfies both the novice user and the professional programmer. Both enjoy appropriate software tools and both can work on the same system, protected by advanced hardware/software security features.

The Level 68/DPS user interface provides an environment for a nearly unlimited scope of applications regardless of size, complexity, or storage requirements. The multiprocessing, multiprogramming capabilities of Level 68/DPS and its diversity of languages and utility routines provide the user with all the support needed.

THE PROCESS, A UNIQUE CONCEPT

When first accessing (logging into) a Level 68/DPS System, a user is allocated system resources in an environment termed a "process." Specifically, the process is dynamically assigned space within the virtual memory (address space) and other system resources as required. As a result, each user views his process as if it were the only one in the system. In this environment, the user's address space dynamically grows and shrinks as program requirements expand and contract and the activity is totally transparent to the user and under control of the shared operating system. The system creates a process at log-in time and destroys it at log-out time on behalf of each user. The user executes his program and system commands in coexistence with the processes of all other logged-in users under the multiprogramming control of the Multics operating system.

SYSTEM FEATURES

Some of the more important features of the Level 68/DPS interactive programming environment include:

- Flexibility of environment shaping
- Information sharing in the Level 68/DPS Virtual Memory and Storage System
- Powerful language processors
- Extensive support facilities and tools
- Powerful command processor
- Protection (security)
- Special user interfaces

Flexibility of Environment Shaping

The administration of a typical Level 68/DPS system includes one system administrator and multiple project administrators. Each project administrator defines the working environment for users in that project. He may give a user maximum flexibility by allowing him complete control in creating his own initial process, or he may limit the user's capabilities by restricting access to various software functions.

The project administrator, then, defines the range of access each user has to system software functions. If the user has complete control of his own process environment, he may change parts of that environment and still be within the normal operating conventions of the system.

Information Sharing in the Level 68/DPS Virtual Memory and Storage System

A Level 68/DPS system provides total sharing of data between users. In addition to sharing the operating system's modules, libraries, language processors, and applications, users can even share user code and data. This is possible because all Level 68/DPS compilers generate pure, reentrant code by default.

All procedures and data are contained within the Level 68/DPS Storage System and its associated virtual memory including facilities that provide the user with extensive control over file manipulation and file sharing. A user can specify which individuals may access his files, and by which mode of access. Access can be given to one user, to a group of users (project), or to a particular class of users (interactive or batch).

The Level 68/DPS Storage System is supported by a powerful virtual memory, totally transparent and available to the user as needed. Virtual memory dynamically expands and contracts according to user requirements and system resources. Thus, programmers no longer need to be concerned about overlaying or partitioning program modules to satisfy limited main memory resources. Instead, they can concentrate on program synthesis and on developing the most efficient algorithm to solve their particular problems.

Powerful Language Processors

Level 68/DPS includes several fully compatible language processors. Foremost is a functional PL/I compiler that is used for both system programmers and applications programmers. The present PL/I compiler has undergone several major design iterations to become perhaps the most stable and reliable PL/I compiler in existence. This is the same PL/I compiler that is used to produce the Multics operating system software itself, 95 percent of which is written in the PL/I language.

Level 68/DPS supports COBOL-74. As with all Level 68/DPS compilers, COBOL programs can call programs written in any other language, thereby offering developers optimal flexibility.

A complete FORTRAN compiler is available to satisfy any FORTRAN requirement as well as to facilitate the transfer of software from other computer systems.

A BASIC compiler offers quick compilation and execution. It can be used as an independent language processor or in the simple time sharing subsystem called FAST.

An APL language processor is also available. This is an interactive interpreter with extensive functionality.

For users who find it necessary to write portions of their software in the language of the host computer, Level 68/DPS includes the ALM (Assembly Language for Multics) assembler. This assembler supports all system requirements for interprogram communication.

A program written in any language available on Level 68/DPS can also call programs written in another language by merely following that language's calling conventions. For example, APL functions can call PL/I procedures.

All compilers will automatically generate pure, reentrant code for users, making all programs immediately shareable.

Extensive Support Facilities and Tools

Stable and reliable software components within the Multics operating system provide numerous utility and support functions. Foremost among these are the text editors. These text editors have undergone several design iterations to increase their reliability and sensitivity to human engineering requirements. Level 68/DPS text editors range from a simple editor supporting line-numbered files to advanced editors for experienced users.

Several extensive interactive debugging packages permit a user to analyze and correct a compiled program at both the original source level and the more specific machine-register level.

Tools to measure performance permit the user to analyze a program's behavior and facilitate the development of optimum applications software.

Inter-user communication facilities, both immediate and deferred, permit online messages to be transmitted among users. In addition, online documentation facilities provide the user with useful word processing and document preparation tools.

Powerful Command Processor

The command processor, by which a user communicates his requirements to the system, accepts input from a console, interprets the user's request, and invokes the software component to perform the desired function. The software component can be either system- or user-supplied; there is no distinction at the command level. The command processor allows recursive, iterative commands and the embedding of function calls in the command line.

The command processor is a shared, replaceable module, written in PL/I. Therefore, if the project administrator desires, a user can be required to interface with a special version of the command processor (possibly user-created), thereby limiting the software requests or commands available to him. The command processor thus permits an extremely wide range of interfaces to all system facilities on either a controlled or open-ended basis.

Tools are available to the user which allow the abbreviation of commands or character strings for the development of personalized shorthand methods for directing program execution or accessing files.

A facility also exists which allows users to program in commands with logical branching, variable maintenance, file management, and I/O control. This allows the development of complex applications without the involvement of language processors.

The command syntax has been designed to provide as sophisticated and flexible a user interface as any user might possibly require for both a commutative and associative syntax form. However, simple requests have a simple form.

SUMMARY

- The Level 68/DPS interactive programming environment provides facilities for both the novice and advanced user, for a wide range of applications.
- The user's virtual memory (address space) dynamically changes as program and data requirements change.
- A unique process environment exists for each user, and this environment can be reshaped as needed.
- Files are protected by user-specified access controls.
- Level 68/DPS includes several language processors: COBOL-74, PL/I, FORTRAN, BASIC, APL, and ALM.
- The support facilities of Level 68/DPS include text editors, program debugging aids, performance measurement tools, inter-user communication facilities, and online documentation aids.
- The Level 68/DPS command processor allows a wide range of interfaces to all system facilities either on a controlled or open-ended basis.
- Included within Level 68/DPS are special user interfaces that permit the development of other operating systems, closed subsystems, or limited service facilities.

SYSTEM CONFIGURATION

The functions described herein are applicable to any Level 68/DPS configuration.

Specifications may change as design improvements are introduced.

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

20936, 2.5578, Printed in U.S.A.

Honeywell

SERIES 60 (LEVEL 68/DPS)

Controlled Sharing and Security

In today's data processing environment computer security is a major concern. News stories tell of millions of dollars being stolen; company proprietary information being sold to competitors; and valuable information being deliberately destroyed or altered. In addition, there is growing concern regarding the protection of individual rights -- just what type of data can be compiled on an individual and how should such information be protected from unauthorized disclosure. At this time, laws protecting privacy carry both civil and criminal penalties. So the processing of sensitive information on an insecure system exposes companies to loss and fraud and, potentially, to civil and criminal proceedings. Secure computer systems have become a vital concern to the data processing industry.

SECURITY MECHANISMS

Level 68/DPS is the most secure computer system commercially available.[1] Level 68/DPS and its Multics operating system provide security by a combination of methods unique in the industry. These methods are designed into, and are an integral part of, the functioning of the system. Because the security mechanisms are implemented in both hardware and software and are applied universally to all user and system functions, they are far more difficult to subvert and security benefits are achieved with insignificant system overhead.

Initial Access

Initial access to the Level 68/DPS system is controlled by means of the user identification and password.

The password mechanism is the first and most important line of security since it controls access to the system and verifies the user's identity, upon which further access decisions are based. To ensure that users treat passwords with the care the system requires, Multics provides facilities to allow users to change passwords easily, to permit the System Security Administrator to compel periodic password changes, and to require users to use system-generated, pronounceable passwords.

When the system requests the entry of the password at log-in time, the print mechanism of the user's terminal is turned off (if the terminal has this feature) or a mask is printed, over which the password is typed. Thus, a password is never displayed in readable form when it is entered.

Passwords are stored in encrypted form within the system. When a user logs in, the typed password is similarly encrypted and the scrambled values are compared.

Multics audits the usage of each password. It also counts incorrect passwords and types a message at log-in, telling the user how many times that password has been given incorrectly since its last correct entry. This alerts the user to the possibility that someone has attempted to guess the password.

The System Security Administrator can also set an entry in the system message table causing a message to be sent whenever the user is logged in and someone else attempts to log-in with the same name, project, and password.

Once verified and accepted by the system, the user is screened for information sharing (i.e., the kind of access permitted to the file system and to user and system software).

Information Sharing Controls

Beyond the password control (which screens every person attempting to use the system), three additional controls regulate access rights to all data and programs in terms of individual users and processes. These information controls are access control lists, access isolation mechanism, and the ring protection mechanism.

ACCESS CONTROL LIST (ACL)

The access rights for each segment are defined in its access control list (ACL). Through the ACL, users can, at their own discretion, grant or deny access to their segments and directories. The ACL specifies the users who have been granted access to the segment and the mode of access allowed them. Users who do not appear on the ACL have no access to the segment. Read, write, execute, and null[2] permissions may be specified for segments (both data and program); status, modify, and append accesses may be specified for file system directories. These permissions may be specified by user name, by project, or by "instance" (whether a process is absentee or interactive), or by any combination thereof. Classes of users can also be specified; for instance, all the users in a project, specific users in a project, or even all users in all projects. Access is initially verified by Multics software and is enforced by the hardware every time the segment is referenced thereafter.

ACCESS ISOLATION MECHANISM

The Access Isolation Mechanism (AIM) allows administrators of the system to define several levels of privilege, which the system itself enforces rigidly. Enforcing the separation of these levels is totally independent of other access control or user action. This administrative mechanism overrides user discretion in granting access and ensures privacy by preventing inadvertent or malicious disclosure of information between these privilege levels, even by those who "own" the information.

AIM can be explained in simple terms. At log-in time, each process is assigned a sensitivity level and category (clearance) based upon the clearance of the user, the terminal, and the project. Also, every directory and segment (object) within the storage system has a sensitivity level and category (classification) associated with it. If the clearance of a process is equal to the classification of the desired object, all access to the object (allowed by other access control mechanisms) is permitted. If the clearance of the process is greater than the classification of the desired object, read, execute or status permissions are allowed to the object (within constraints imposed by the other access control mechanisms). Finally, if the clearance of a process is less than the classification of the desired object, all access to the object is denied.

AIM supports 8 clearance and 18 "need-to-know" categories within each level. Access is granted or denied explicitly on the basis of the security classification of a file or program and the security clearance of the user. This mechanism supplements the access control lists. Like the ACL mechanism, access is initially verified by Multics software and thereafter enforced by hardware at every reference. AIM can be invoked or disabled at the discretion of each Multics site. AIM also provides extensive security auditing controls to monitor user activity.

RING PROTECTION

The Level 68/DPS Security System uses a hardware-implemented, multilevel ring structure to control its users and to protect itself. The ring structure is a generalization of the two-state capability of other computer systems (master/slave mode, supervisor/program state, etc.). With Multics, the structure has been expanded to eight states or "rings of protection" numbered 0 (most privileged) through 7 (least privileged). The operating system resides in the most privileged rings, 0 through 2, while users generally operate in the less privileged rings, 3 through 7. The segments of the operating system are in the most privileged rings to prevent uncontrolled access or modification by the users of essential system information.

The basic rule is that users can only reference those segments in the same or less privileged levels than the ring in which they are currently executing. Access to higher privileged rings is only possible through a gate program. (This is analogous to master mode entry, supervisor call, etc.) The use of eight levels of protection rather than two allows user programs to take advantage of features for protection normally reserved for operating system software.

[1] In a 1975 study conducted for the Air Force, the MITRE Corporation concluded that Multics is the most secure operating system available.

[2] Null access is implied by default; that is, if a user does not issue a command granting another user access to a segment or directory, that other user cannot access the data in any way. However, null is useful in selecting a small number of individuals from a project to whom it is desirable to deny access.

Specifications may change as design improvements are introduced.

Honeywell

Honeywell Information Systems
In the U S A. 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico Avenida Nuevo Leon 250, Mexico 11, D F
20909, 1578, Printed in U.S.A.

Honeywell

Multics PL/I

LEVEL 68

Multics PL/I is a language processor designed for commercial, scientific, and system programming applications. It was developed on Honeywell equipment in conjunction with the Massachusetts Institute of Technology. Multics PL/I is the language defined by the American National Standards Institute's PL/I standardization committee and is scheduled to become a standard.

The compiled code is extremely efficient. The compiler was specifically designed for Multics and has been used to compile itself and most Multics software.

ADVANTAGES

Compilation and execution may be initiated through absentee (Multics batch processing) or interactive mode.

Programs written in PL/I ensure permanent compatibility and ease of maintenance.

Object modules are produced such that no relocatable edit is required. The normal mode of operation is to execute with dynamic linking and loading so that unreferenced data and unused programs are never loaded into main memory.

Relocatable object permits the binding of separately compiled programs together into one segment which has fewer pages than its unbound components.

A run-time symbol table may be created by the compiler and used by the Multics debugger to make symbolic references to the program data at run time. There is no special checkout compiler and therefore no recompilation necessary to debug a program.

An optional optimizer performs extensive optimization of common expressions, conversions, and accessing code throughout a procedure or begin block. Register allocation is based on usage statistics gathered by the optimizer resulting in intelligent use of pointer registers by the object code.

CAPABILITIES

PL/I is a block-structured language that allows both internal and external names. This feature facilitates the development and maintenance of modular PL/I programs. All procedures are recursive and sharable.

Multics PL/I has a comprehensive set of data formats. These include eight distinct types of data: arithmetic, string, locator, format, label, entry, file, and area data. These formats give PL/I considerable descriptive power.

In addition to fixed-point and floating-point binary arithmetic, Multics PL/I provides variable-precision true fixed-point and floating-point decimal arithmetic of up to 59 decimal digits directly supported by hardware. Structure variables (similar to the hierarchical descriptions of COBOL) enable the programmer to explicitly define data structures as any aggregate of elementary data formats.

Dynamic allocation for scalar variables and aggregates is provided by the automatic, controlled, and based storage classes.

PL/I has powerful bit string and character string handling capabilities. Operations and functions are performed on either fixed or variable length strings. The extended instruction set of the central processor is fully utilized to perform character- and bit-string operations, picture editing, as well as decimal arithmetic and arithmetic base conversions.

Arithmetic, string, or pointer variables declared with the "unaligned" attribute are packed into the minimum number of bits, giving the programmer complete control over the packing of structures and arrays.

Through the use of pointer-valued Multics functions and PL/I based variables, a user can easily access any bit in the entire virtual memory.

Declaration of initialized arrays and data structures is permitted. Components can be freely interspersed in PL/I programs to aid in program documentation.

Multics PL/I utilizes the full ASCII character set defined in American National Standards Institute standard X3.4-1968. Both uppercase and lowercase letters can be used to form names up to 256 characters long. This offers the user greater naming flexibility.

The % include macro provides for the inclusion of program text without the use of a preprocessor.

Complete symbol listings show how each name was declared, as well as its attributes, and its address allocation.

The compiler diagnoses over 350 errors, giving complete, readable diagnostics that include the erroneous statement or name. Warning diagnostics are given for common mistakes such as an undeclared name or implicit conversion of data types.

PL/I programs may call procedures written in other languages or vice versa provided they observe the interface conventions, as is the case with the other compilers in the system.

Multics PL/I and FORTRAN compilers have similar options, program listings, and error messages, and in fact share the same compiler code generation module phase and are, therefore, completely compatible.

PL/I input/output facilities provide a convenient method of constructing and maintaining large files within the virtual memory or on removable media.

The PL/I "do" statement and "if" statement allow the programmer to construct flexible program logic without the proliferation of statement labels.

The "on" statement of PL/I permits the programmer to make arrangements to handle special conditions which arise during execution. These conditions can arise as the result of errors recognized by the hardware or be signalled by the program itself.

Multics virtual memory, coupled with PL/I pointer data, facilitates the programming of complex list processing techniques.

OTHER MULTICS FEATURES

Multics PL/I is a powerful language on a powerful system. One of the most advanced computer systems in the world, Multics offers extensive security provisions, virtual memory, interactive programming environment, hierarchical storage of data, and highly functional administrative control features. For more information on the Honeywell Multics System, contact your Honeywell Marketing representative.

SYSTEM CONFIGURATION

The functions herein described are applicable to any Level 68 Multics configuration.

Specifications may change as design improvements are introduced.

HONEYWELL INFORMATION SYSTEMS

9063
7.5774
Printed in U.S.A.

In the U.S.A.: 200 Smith Street, MS 061, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale Ontario

Honeywell

MULTICS APL

SERIES 60 (LEVEL 68/DPS)

Multics APL is an advanced version of the APL programming language - an interactive system for use with Honeywell's large-scale Level 68/DPS computers. Multics APL is a general purpose language that is both easy to learn and powerful to use. It is interactive by design – problems can be attacked swiftly, error messages are informative, errors can be corrected quickly and easily – all within the APL environment.

APL brings the full scope of data processing to business and technical problem solvers who may have little programming experience.

It is particularly well suited for business and scientific applications requiring the manipulation of arrays of data. Typical business applications include financial modeling, investment analysis, sales forecasting, and payroll and budget analysis. Scientific applications include linear programming, regression analysis, and pipe stress networks.

BENEFITS

Multics APL offers substantial benefits:

- Provides powerful language statements for easier problem solving
- Provides a set of tools for interactive development, debugging, and execution of programs
- Offers data processing capabilities for any level of programming expertise
- Reduces time for programmer system development thereby increasing programmer productivity

FEATURES

The following features make Multics APL particularly attractive:

- File access capabilities
- Ability to store APL functions and programs for later execution
- Ability to automatically start a function executing when program is loaded
- Powerful execute operator which interprets a character string operand and may produce a character string result
- Ability to diagnose errors in a longer, more explanatory format to assist new APL users
- Support of ASCII terminals and a variety of APL terminals (especially graphics terminals)
- Several preprogrammed workspaces, including a fully documented, tutorial course for user self-instruction in Multics APL
- ASCII-compatible character set
- Accurate floating point computations
- Sizable arrays and unlimited workspace in virtual memory
- File system that does not require knowledge of job control language
- Full security and integrity protect programs and data from unauthorized use or modification
- Convenient interface to other Multics programming languages
- Software to convert from IBM to Multics format
- Ability to access Multics data bases via PL/I subroutines

SYSTEM DESCRIPTION

System operation is straightforward. At a terminal, the user simply calls Multics APL, and then types an expression to be evaluated. The Multics APL interpreter performs the calculations, prints the results, and awaits a new input line. The results of an expression evaluation can also be assigned to a variable and retained from line to line for use in subsequent evaluations.

In addition, a sequence of calculations can be stored as a function to be recalled by name and interpreted as a single entity. The result of the function is processed as though the entire sequence of expressions had just been entered at a terminal. Results from any form of expression evaluation may be printed at the terminal, presented in graphic form (on a suitable terminal), formatted into a tabular report, or stored in a file for subsequent retrieval and processing by Multics APL, or other programming languages. The entire state of any Multics APL session may be saved for subsequent use, including all variable names, their current values, and stored program functions.

The APL language employs a special character set (normally requiring a special APL terminal) in which most operations are represented by a single character. Besides the more common operations (add, multiply, etc.), Multics APL provides single-character operations for more complex functions, such as sorting, random number generation, and matrix inversion. Some of these functions are suited particularly to the manipulation of large or complicated data arrays.

Access to a special APL terminal is not required; Multics APL can be used from an ASCII terminal through a special set of over-strike conventions. Unlike similar conventions in any other APL, this convention preserves the visual appearance of APL programs. It is as easy to read Multics APL programs on an ASCII terminal as it is on an APL terminal. Multics APL supports all APL terminals – including EBCDIC, Correspondence, APL/ASCII bit-paired, and APL/ASCII typewriter-paired.

The Multics APL character set is a compatible superset of ASCII; hence, character data can be easily and efficiently shared between APL and other Multics programming languages.

By using powerful APL language elements, the user can devote full attention to solving the problem at hand. Multics APL helps the user avoid the complexities of the individual programming steps required and the interrelationships of the system operation. For example, inversion of a matrix, which would involve many program statements with nested loops and indexing in other programming languages, is accomplished with a single-character operation in Multics APL.

Multics APL can call APL functions written in PL/I, which, in turn, can call programs written in BASIC, COBOL, or FORTRAN. The PL/I coded function can be niladic, monadic, or dyadic, and can optionally return a result. It can diagnose the same errors any APL operator can diagnose and can call out to any other Multics subroutine or system interface.

Thus, this facility enables the APL user to get at *any* system interface or application program by writing a short PL/I program to transform the call. This means the Multics APL user can easily interface to the Multics Graphics System, the Multics Relational Data Base Manager, and other packages.

SYSTEM REQUIREMENTS

Multics APL runs under the control of the Multics Operating System on all models of Level 68/DPS.

Specifications may change as design improvements are introduced.

Honeywell

Honeywell Information Systems
In the U S A 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico Avenida Nuevo Leon 250, Mexico 11, D F
20883, 1678, Printed in U.S.A.

Multics Data Base Manager

Honeywell's Multics Data Base Manager (MDBM) provides Level 68/DPS users with a powerful, versatile, efficient, and easy-to-use data base management capability. MDBM functions as a subsystem of the Multics operating software, and makes use of the Level 68/DPS virtual memory and file management subsystems. It is designed to support concurrent access to up to 64 data bases of up to 180 billion characters each.

Much of the versatility of MDBM derives from the fact that it offers the user a choice of two different methods for structuring and manipulating a given data base: a relational approach and a procedural, CODASYL-standard (Conference on Data System Languages) approach. While all data is stored in a relational format, two differing interfaces are visible to the user.

The relational technique greatly simplifies the job of programmers and end-users, as a detailed knowledge of the logical structure of the data base is not required to use it. MDBM performs the retrieval function automatically. For example, a person seeking data writes a statement that defines the nature of the data required; he does not provide specific instructions as to where and how the data is to be retrieved.

At the same time, the procedural approach is available to those programmers who are familiar with and prefer CODASYL data base techniques.

MDBM is the industry's first fully implemented relational data base manager commercially available from a computer manufacturer. It is also the only data base manager to offer both relational and CODASYL capabilities in the same system.

MDBM BENEFITS

The Multics Data Base Manager provides the following benefits:

- *Improved programmer productivity* Programmers can accomplish data base tasks with much less effort when employing MDBM's relational capabilities.
- *New end-user flexibility* Because of the simplicity of relational techniques, end-users can independently retrieve data base information without support from the programming staff.
- *Ease of maintenance* – The tasks of entering and changing data are greatly simplified.
- *Data storage efficiency* – One data base system can meet the needs of an entire organization without redundant files and effort.
- *Improved data accuracy* – The elimination of redundancy ensures consistent information with fewer chances of error.
- *Ensures data integrity and security* – Inherent Level 68/DPS integrity/security features are available to the MDBM user.

MDBM FEATURES

The Multics Data Base Manager includes the following significant system design features:

Relational interface – Multics Relational Data Store (MRDS), a component of the data base manager, represents data relationships by means of formal algebraic entities. A user structures and accesses data files without concern for how or where the data is actually stored. As a result, the user's task is greatly simplified.

Procedural interface – Multics Integrated Data Store (MIDS) provides an interface with the data base manager following CODASYL standards. MIDS is a subset of Honeywell's I-D-S/II data base management system. This capability is highly flexible, allowing the building of network, hierarchical, sequential, or cyclical structures.

Language independence – Any Level 68/DPS supported language may be used to access MDBM facilities, including COBOL-74, PL/I, FORTRAN, APL, BASIC, and Assembler. Well-defined CALL statements are employed.

Independence of processing modes MDBM supports all processing modes such as transaction processing, time sharing, batch, remote job entry and direct access. All of these modes can be supported simultaneously.

Controlled sharing All user data (as well as operating system software, libraries, and user code) is potentially shareable at the discretion of its owner. Since all Level 68/DPS language processors generate only pure reentrant code, no copies or reloads are required.

Data definition and program independence – Data definition is an independent function. In most cases, changes to the data base will not require reprogramming of user applications.

Query capability – A special MDBM query language, termed LINUS (Logical INquiry and Update System), provides comprehensive query capabilities.

Online access and update – Records may be easily added, modified, or deleted online. Multiple users may access the same data base concurrently. MDBM can be invoked by as many users as are allowed on the system.

Concurrent access and update controls – Update privileges can be assigned to individual users or classes of users. To ensure integrity, users may specify exclusive use of the data base when it is opened; or, if sharing a data base, users may temporarily reserve a record type and associated sets during critical update operations. It is possible for privileged users to specify exclusive update, which locks out all other processes attempting to access the data base; and it is also possible to specify exclusive retrieval, which locks out all updaters from the data base.

Report generation – The Level 68/DPS Report Generation Language (RGL) facilitates the production of reports, in conjunction with either the LINUS query language or ASCII files.

Automatic data recovery and restart – MDBM uses Level 68/DPS backup/retrieval mechanisms. They provide recovery of a data base after system failure or when a disk has been damaged.

Monitoring – Tools exist to monitor data base usage from various aspects.

Dynamic tuning – A system administrator can view current monitoring data from a terminal and dynamically alter parameters to affect performance.

Data integrity and security – Level 68/DPS is the only system where all data integrity/security features are implemented in both software and hardware. MDBM derives its superior integrity/security characteristics not from provisions within the data base manager, but rather from design features of the Multics operating system and Level 68/DPS hardware.

TERMS DEFINED

The relational and CODASYL approaches have developed separately, so that terms used in describing one approach differ substantially from those used for the other. Table 1 is provided to bridge this terminology gap.

TABLE 1 - TERMS

Traditional Data Processing Terminology	Honeywell MDBM	
	Relational Approach	**CODASYL Approach**
Record type, class or format (physical)	physical record	physical record
Record type, class or format (logical)	relation	record type
Record occurrence	tuple	record occurrence
Field/element	attribute	data element/field
Range of values (the set of all values associated with a field type)	domain	largely ignored
Total data base definition	data model	schema
Program view of data base	data submodel	subschema

DATA BASE DEFINITION PROCESS

Figure 1 illustrates the MDBM data base definition process. Users may create and access a data base using either the MRDS or MIDS interfaces with the following restrictions:

- A data base created using the MIDS facility may be accessed only with MIDS programs.
- A data base created using the MRDS facility may be accessed only with MRDS programs.

The definition of a data base in MDBM is accomplished by the user or data base administrator defining either:

- A *data model* for a relational data base.
- A *schema* for a CODASYL data base.

The data descriptions are not directly referenced by user application programs, and each application program may have a different view of the data base. The definition of that portion of the total data base affecting a particular program is accomplished by defining either:

- A *data submodel.*
- A *subschema.*

Partial views (data submodels or subschemas) must be proper subsets of the total data base definition.

In the case of a relational (MRDS) data base, the data model definition file contains the complete description of the different data elements to be found in the data base. This is the data base administrator's view of the data base. The data submodel definition file contains the definitions of those relations and attributes of interest to a specific program. This is the user's or programmer's view of the data base. The data submodel is defined independently of the data model and no binding is done until application program execution open-time.

Data types for data submodel attributes are not defined within the data submodel; they are defined within the application program. They may differ from the data types defined in the data model for the corresponding attribute name. Allowable data types (in PL/I terms) are real and complex fixed binary, real and complex floating binary, real and complex decimal, varying and nonvarying bit string, and varying and nonvarying character strings. Binary data types may be single or double precision.

When a data base administrator defines an MIDS data base, both a *schema* and a *data model* definition file are automatically created (Figure 1). The schema definition file contains only information associated with a CODASYL data base relevant to the data structure/organization of that data base. Schema entries contain data and network structure information while data model entries contain descriptions of the data elements in the data base.

Again, data descriptions in the schema definition file are not directly referenced by user application programs. The user or data base administrator defines that portion of a total data base of interest to a particular program by use of a subschema. A subschema definition contains only the application program view of the data base and is the MIDS counterpart of the data submodel in MRDS (just as the schema is the MIDS counterpart of the data model).

In practice, MIDS controls structural definitions, whereas MRDS controls data definitions.

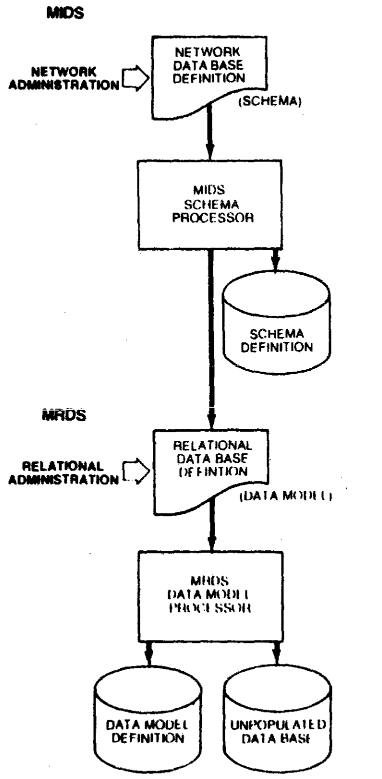


Figure 1. MDBM Data Base Definition

THE RELATIONAL DATA STORE

The Multics Relational Data Store (MRDS) provides an integrated set of functions to support the description and processing of a wide variety of data base structures. Data independence is achieved through the use of data model/submodel concepts.

The MRDS user defines a data model using a technique known as "normalization" or "reduction to third normal form." The process of normalizing a data base consists of a subjective process performed by the data base administrator whereby complex relations are reduced or transformed into simpler relations without loss of information or dependencies. The rationale for using the third normal form relational model derives from certain anomalies which can otherwise occur in a relational data base.

The data model or data base definition is defined, accessed, and maintained by the data base administrator. The administrator, working in conjunction with application users, defines a valid data submodel to be referenced by each application program. Under certain user-controlled circumstances, a user may act as his/her own administrator and deal directly with the data model. The data submodel provides each application program with its view of a data base and must be a proper subset of the data model.

The definition of a data submodel may differ from the data model in several ways:

- Attribute names may differ.
- Attributes may be omitted from a data model relation.
- Attributes may be ordered differently within a relation.
- Relations may be omitted.

A data submodel definition may be created separately or in conjunction with an application program. Several data submodels may be associated with the same data model. Several data submodels may be referenced by the same application program. Data submodel definitions may intersect.

When defining a relation in a data model or submodel, it is necessary to specify which attributes or fields are to be used as components of the primary key (i.e., key attributes). Each tuple or record occurrence must be identifiable by some non-null primary key value which must be unique. The expression for defining a relation and its attributes resembles the format used in most relational literature.

The "open" procedure is called when the MRDS user wishes to access the data base. At this time the data model and submodel are resolved. Figure 2 illustrates access activity.

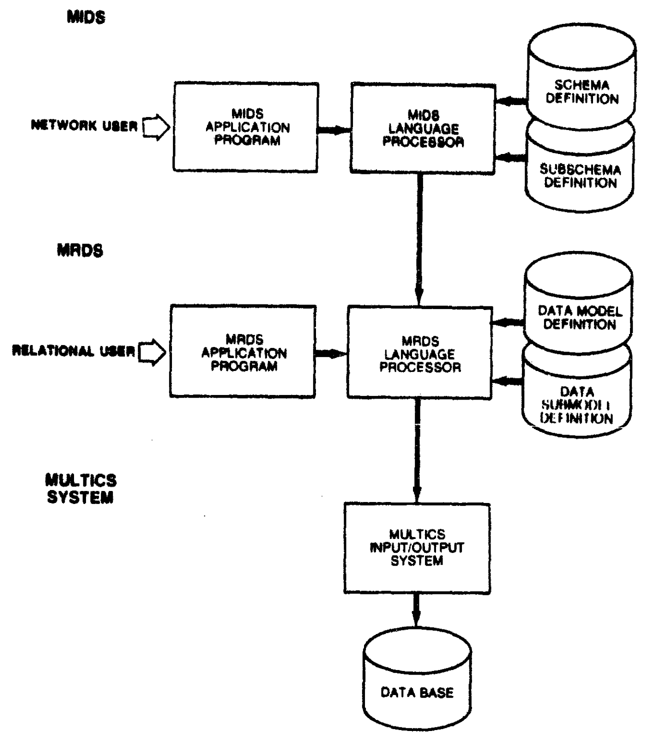


Figure 2. MDBM Data Base Access

The MRDS Language Processor provides the user with the following capabilities:

- Open and close a data base defined by a specified data submodel.
- Retrieve data based on a flexible selection capability.
- Modify and delete items within a data base.
- Enter new information into the data base.
- Perform the above while allowing for the maximum possible concurrent access capability.

The data manipulation capability provided is relationally complete. That is, it possesses the full power of the relational calculus. Any query expressible in first-order predicate calculus is expressible in an MRDS selection mechanism.

THE MULTICS INTEGRATED DATA STORE

The Multics Integrated Data Store (MIDS) provides a CODASYL language interface to the functions required to support the description and access to CODASYL data base organizations, including hierarchies and networks. Data independence between the physical data base and application programs is achieved through use of schema and subschema concepts as specified by CODASYL.

The schema describes the total data base and is defined and maintained by the data base administrator. The administrator, working with prospective users, defines valid subschemas to be referenced by individual application programs.

The subschema must be a proper subset of the schema. The subschema contains only the information necessary to define the data required by the application program.

MIDS relies primarily on access control lists rather than privacy locks or passwords to provide security at the level of the record and set. Concurrent update protection is provided at the record type and set name level.

The following specific CODASYL schema definition functions are supported by MIDS:

- A data base, records, fields, and sets may be given names.
- Record location mode may be specified as CALC or VIA SET. (All records in MIDS must be given a unique control key field even if the location mode is VIA SET.)
- Field types may be BINARY, DECIMAL SIGNED, or CHARACTER.
- Sets may be defined.
- ORDER of set insertion is defined as PERMANENT INSERTION and DUPLICATES NOT ALLOWED for SORTED ORDER.
- Set members may be ordered as FIRST, NEXT, LAST, PRIOR, or SORTED BY DEFINED KEYS.
- Set membership is defined as MANDATORY and AUTOMATIC.
- Set selection may be THRU OWNER IDENTIFIED BY CALC-KEY or BY APPLICATION.

A subschema may describe only that portion of the entire data base (i.e., schema) which is of particular interest to the application program. (A schema may be shared by a number of subschemas.) A subschema may differ from the schema in the following ways:

- Field names may be renamed or omitted from a subschema record.
- Field names may be ordered differently within records.
- Field types may differ from the corresponding schema field type (although field types are not defined in the subschema, per se).
- Records may be renamed or omitted.
- Sets may be renamed or omitted.

Several subschemas may be referenced by an application program and subschema definitions may intersect. Subschema record, field, and set names must be associated explicitly with schema record, field, and set names. Field types are defined in the using application program.

At execution time the validity of the subschema is verified and the views of the data base represented by the schema and subschema are resolved.

The following basic functionality is provided by MIDS Data Manipulation Language (DML):

- READY
- STORE
- ERASE
- FIND
- GET
- MODIFY
- FINISH
- KEEPX
- FREEX

It is possible to READY multiple subschemas within one process.

Specifications may change as design improvements are introduced.

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

21522, 2878, Printed in U.S.A.

Honeywell

LINUS

SERIES 60 (LEVEL 68/DPS)

LINUS (Logical Inquiry and Update System) is a powerful and easy-to-use facility for accessing data bases from a remote terminal. It provides complete data base management capabilities, including both retrieval and update operations. LINUS functions as a subsystem of the Multics operating software and uses Multics Relational Data Store (MRDS) for data base access. LINUS is used by a variety of business professionals, including inventory control personnel, budget planners, geologists, and school administrators.

LINUS uses a high level, nonprocedural data selection language called LILA (LInus LAnguage). LILA allows people, who are not trained programmers, to solve problems by working with data bases. The user views the data base as a simple set of tables, consisting of rows and columns; LILA provides an easy way to "look up" information contained in those tables.

System operation is straightforward. At a terminal, the user simply calls LINUS and types a request for LINUS to open the desired data base. If the user is unsure of how to proceed, he types "help" for assistance. After the data base is opened, the user may call LILA and type a selection expression specifying the data to be accessed. LILA then processes the selection expression, retaining it in a form suitable for reference by subsequent LINUS requests.

After selecting the data, the user may type a LINUS request to manipulate it. Among the actions which can be performed are display the data on a terminal, write a formatted report, place the data into a Multics file for later use, and update the data base. The data may be acted upon by several manipulation requests, and new data may be specified at any time by typing a new LILA selection expression.

BENEFITS

LINUS is easy to use. With three short words even an inexperienced computer user can get information from a data base using LINUS. These words are "select," "from" and "where." For example, a telephone directory can be thought of as being a table with three columns of information: name, address, and phone number. To find the phone number of John Smith, the user scans the name column for "Smith John" and reads the value from the phone number column in the same row. With LINUS, using LILA, this operation is described as:

```
select number
from phone book
where name = "Smith John"
```

Studies and experience have shown that inexperienced users can attain substantial competence in LINUS within a matter of hours.

Other benefits include:

- *Direct End-User Access* -- Using LINUS, the user can independently retrieve and update data base information without support from the programming staff.
- *Support of Unforeseen Requirements* The LINUS user can immediately respond to unexpected information requirements by interactively selecting the desired data via a terminal, without writing a program.
- *Complete Selection Capability* LINUS is powerful and "complete" in the sense that the user can select any information contained within the data base using LILA, subject to security constraints.

FEATURES

- *Complete Data Base Functionality* - LINUS provides facilities enabling the user to add and delete rows from a data base table, to modify column values in a table, to define temporary tables for personal use, and to retrieve information from one or more specified tables.
- *Macro Facility* -- A parameterized macro facility allows the user to invoke previously saved sequences of LINUS requests. This allows the tailoring of an environment to an individual user.
- *LILA Line Editor* A BASIC-like line editor built into LILA simplifies construction of data selection expressions.
- *Built-in and User-defined Functions* Built in functions allow the user to determine sums, averages, and counts, of data as well as to search on such items as partial character string values, and rounded or truncated numeric values. In addition, there is

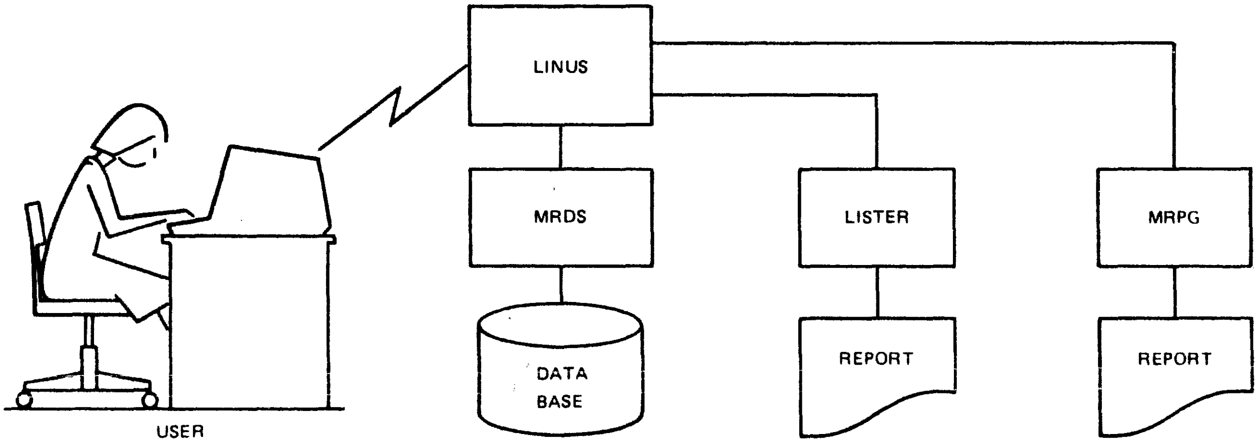
a well defined method for dynamically adding functions required by local users.

- *Internal Variables* – The LINUS user can assign retrieved data values to internal variables, allowing subsequent data selections to be dependent upon previously retrieved data.
- *Help Facility* – A help request available within LINUS provides information on how to use its many facilities. The user need only type "linus" to invoke the subsystem, and "help" in order to begin using LINUS.
- *Table of Contents* – The contents (tables and columns) of the data base may be displayed allowing a person unfamiliar with the structure and content of the data base to use LINUS.
- *Report Writing Capability* – Formatted reports from retrieved data can be created using either the Multics Report Program Generator (MRPG) or the WORDPRO facility, Lister. Either facility can be invoked with a simple LINUS request.
- *Multics* – Any Multics capability can be invoked from within the LINUS subsystem. Some possible uses of this feature are sorting retrieved data via the Multics sort facility, or editing retrieved data with one of the Multics text editors.
- *Data Base Creation* – The user can create a private data base via easy-to-use MRDS commands. Special documentation oriented to the LINUS user makes this process especially simple. The data base can then be maintained and accessed using LINUS.
- *Data Sharing, Integrity, and Security* – LINUS fully utilizes MRDS and Multics facilities for concurrent usage control, data base security, and data base backup and recovery.

SYSTEM REQUIREMENTS

LINUS operates under Multics on any Series 60, Level 68/DPS configuration. The availability of MRDS is a prerequisite for LINUS operation.

Specifications may change as design improvements are introduced.



LINUS (LOGICAL INQUIRY AND UPDATE SYSTEM)

Honeywell

Honeywell Information Systems
In the U S A 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.
21422, 1878, Printed in U.S.A.

Honeywell

Word Processing System (WORDPRO)

SERIES 60 (LEVEL 68/DPS)

WORDPRO, another dimension of Honeywell's powerful Level 68/Distributed Processing System – Multics – provides a comprehensive set of software tools for developing a wide range of documents online. And because WORDPRO is integrated within the Multics operating software, its users can develop and maintain documents ranging from simple form letters to complex technical presentations simultaneously with other data processing activities. The end result can be rapid turnaround time, improved productivity, and optimal quality in document preparation.

WORDPRO BENEFITS

Since WORDPRO is an integral part of Level 68/DPS, it offers advantages unavailable with other less comprehensive word processing systems:

- Ease of Use – People with little or no knowledge of Level 68/DPS or word processing can easily use WORDPRO for numerous text processing tasks. Merely by following WORDPRO-generated instructions, even inexperienced users can readily perform functions such as text entry and simple editing.
- Security – The security provided by WORDPRO is the same security provided for all information stored in the Multics virtual memory. Because WORDPRO-prepared documents reside in the virtual memory, they receive the same high degree of security afforded any other job, file, or program.
- Document Management Tools – WORDPRO document management tools make it easy to maintain documents online or offline in a standard format. These tools can be used for document file manipulation, archiving, document linkage, etc. The same tools which control 68/DPS storage also control WORDPRO documentation.
- Tailored Environments – Level 68/DPS lets the WORDPRO user define exactly how text processing is to occur. Unlike other systems, WORDPRO users specify the types of terminals to be used and the various document formats to be accepted. Interfaces to these devices can be changed at any time.
- Selectable Administration – The WORDPRO user is not bound by a restrictive administrative approach. Each site can define what the individuals using the system (terminal operators and document administrators) need to know. The roles of these individuals can be large or small depending on how a user's document processing activities are established.
- Maximum Equipment Utilization – WORDPRO can help users make maximum use of slack computer time. Rather than have the system sit idle when not processing data, WORDPRO can use this extra time efficiently for text processing thus maximizing equipment utilization.
- Total Integration with Data Processing – Within Level 68/DPS, word processing and data processing are fully integrated. Data and text files created and maintained by WORDPRO can be accessed and used by data processing applications. Conversely, files created and maintained by data processing applications are available to WORDPRO users, all without special programming or conversions.

DEVICE FLEXIBILITY

Any terminal accepted by Level 68/DPS can be used for word processing. Users need not purchase special equipment or dedicate terminals for WORDPRO applications. If one set of terminals is used for input and update, other devices can be used for output. Text entered from one terminal can be modified by another without regard to terminal characteristics.

TEXT ENTRY TOOLS

Several tools are provided with WORDPRO for the entry of new text. With these tools, an individual with little or no prior word processing experience can easily power type raw text into the system for later update or incorporation into a document.

The *DOCUMENT* routine is a simplified interface to all the WORDPRO tools. It lets the user predefine document formats, and automatically generates the final document with paragraph and page numbers, footnotes, table of contents (optional), etc. Paragraph editing is provided to allow paragraph addition, modification, and deletion with automatic renumbering. Whole paragraphs may be inserted from a prepared list of paragraphs. Speed typing, hyphenation, and all of the quality control and document management tools are also provided within the DOCUMENT interface. Once processed through DOCUMENT, the text can be printed at a terminal or high-speed printer.

SPEEDTYPE – similar to typing shorthand – allows users to specify abbreviations for input of character sequences. When a document is printed, these abbreviations are automatically expanded to their predefined strings. The result is fewer key strokes typed, higher document quality, and reduced storage requirements.

TEXT EDITING

Input text can be modified using either of the WORDPRO editors: cursor or string. Using a CRT device and the cursor editor, terminal users can simply type over portions of a document which are to be changed.

String editing, which is more comprehensive than cursor, enables the user to manipulate strings of characters in a text for editing or updating purposes. String editing functions range from simple, line-oriented editing to context-oriented searching and replacement.

Together, the WORDPRO cursor and string editors give the user the freedom to select the desired mix of simplicity and comprehensiveness.

DOCUMENT FORMATTING AND HYPHENATION

The WORDPRO formatter provides document format control. In addition to margin and page length control, the formatter handles automatic page and paragraph numbering, widow processing, table of contents and index generation, font and forms control, artwork placements, and automatic hyphenation. Further, the formatter can control:

- Headers/Footers – Up to 20 headers and footers can be specified. These can be page numbers, copyrights, logos, or the current date. These need only be specified once; they will be inserted automatically thereafter.
- Footnotes – They are automatically generated, numbered, inserted, and maintained with the proper page. If footnotes are added or deleted, the remaining ones are automatically renumbered.
- Pagination – Pages can be automatically numbered in Arabic or Roman numerals, in upper- or lowercase, or alphabetically.

QUALITY CONTROL

WORDPRO incorporates an extensive set of quality control tools for detecting and removing typographical errors from documents. First, SPEEDTYPE can be used to correct typos at entry time. For example, common typing errors such as "hte" instead of "the" can be corrected by predefining the former string as an abbreviation to be expanded to the latter. Or, long difficult-to-spell words can be abbreviated, thus eliminating frequent misspellings.

Online dictionaries can also be used to detect misspelled words within a document. WORDPRO offers a dictionary containing over 50,000 correctly spelled English words. The content of a document is compared with the dictionary's entries, and any words not found are entered in an error file or printed at the user's terminal. Multiple dictionaries can also be developed. For example, a dictionary of technical jargon or frequently used non-English words could be established so that these types of words would not be considered misspelled. Dictionaries can be shared, if desired, or used as private versions. Dictionaries can also be added to or deleted from as desired.

Change bars can be generated on documents undergoing review. A complete list of changes (line-by-line) can also be generated in a separate file of notes. Additionally, text comparison programs allow review of new documents against older versions.

LIST PROCESSING

The generation of personalized form letters and billing statements can be accomplished using WORDPRO's list processing capabilities. Lists of mailing addresses can be used concurrently by multiple terminal operators to create these types of documents with key pieces of information inserted at various points. These list processing tools can also be used in conjunction with other 68/DPS facilities (such as the data base manager) to supply current up-to-date account information for monthly statements, billings, etc. Mailing labels or pre-addressed envelopes can also be generated.

ARTWORK

WORDPRO can be used to generate diagrams, organization and flow charts, and logos. These figures can be included as part of the final, printed copy of a document.

OUTPUT CONTROL

Numerous devices can be used to print review copies or final documents. A terminal operator may request that a segment of text be output on a local or remote printer, thereby freeing the input terminal for other work. Line printers, plotting terminals, and CRT devices can all be used for output.

High-speed offline printers, such as the Honeywell Page Printing System, can be used in conjunction with WORDPRO to generate multiple copies in multiple colors with preprinted forms. Special forms control capabilities for line printers and plotting terminals permit documents to be printed on multipart, tear-away, or peel-off forms.

Computer output microfilm/microfiche interfaces are available for generating, distributing, or archiving documents on micro media. Documents can also be stored in files online, or maintained offline on tape, cards, etc.

PHOTOCOMPOSITION

The WORDPRO design incorporates a photocomposition interface to allow the generation of control information for automatic typesetting devices. This photocomposition facility is table-driven so that a variety of devices can be easily supported by simply modifying control tables.

ELECTRONIC MAIL

WORDPRO provides its users with a comprehensive, secure electronic mail facility. This facility allows message-switching networks to be established. Users can send and receive mail ranging from short, immediate messages to lengthy memos or documents, over these networks. Security is provided via a personal or shared mailbox area which is subject to extensive access control checking. Electronic mail can be the basis for the automation of a business's entire in-house mail operation, eliminating the need for couriers.

ACCESSIBILITY OF OTHER FACILITIES

Because WORDPRO is an integrated part of Level 68/DPS, its users can access other facilities. For example, financial data stored in a 68/DPS data base can be selectively inserted into a WORDPRO-generated document to produce an up-to-date monthly or quarterly statement. The 68/DPS Interactive Graphics package can be used to generate artwork for WORDPRO documents, and the Multics file management and manipulation tools can be used with online WORDPRO files.

ADMINISTRATIVE TOOLS

Implemented on a project-oriented basis, WORDPRO can easily control the use of processing resources, access to documents and tools, billing for usage, etc. Users can be restricted to certain functions, or allowed access to full capabilities of WORDPRO. All the administrative controls provided by Level 68/DPS are applicable to WORDPRO administration.

Specifications may change as design improvements are introduced.

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

20633, 2578, Printed in U.S.A.

Honeywell

Multics Graphics System

LEVEL 68

The Multics Graphics System provides a general purpose interface through which user or application programs can create, edit, store, display, and animate graphic material.

FEATURES

- High degree of terminal independence
- Ability to define graphic objects that may be used repeatedly in higher-level objects
- Powerful editing facilities for graphic objects
- Ability to store graphic objects permanently

TERMINAL INDEPENDENCE

The Multics Graphics System is organized into two distinct parts: the terminal-independent portion and the terminal interfaces.

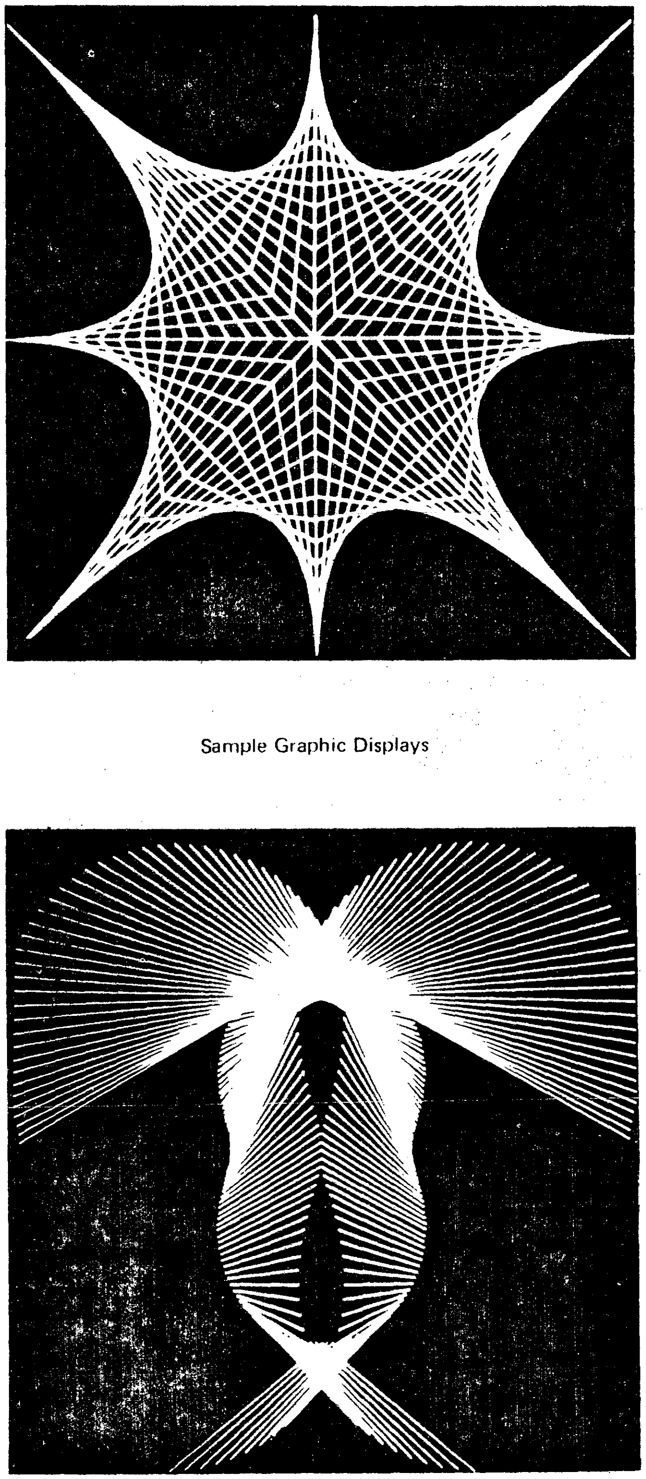
User and applications programs communicate exclusively with the terminal-independent portion of the system. This ensures that:

- User programs and applications routines are not restricted to one particular terminal type, but can use whatever graphic terminal is available.
- Users are not isolated from each other because of the types of terminals they use, but may freely use each others' programs on their own terminals.
- Graphic applications may easily be transferred as new and improved terminals become available.

The Multics Graphics System can accept new types of graphic terminals with a minimum of coding. In most cases, the user need only specify the special characteristics of his terminal in a table and construct a program to perform any code conversion necessary. No special I/O programming is required. He may then use any existing program or graphic file and obtain comparable results on his own device.

STRUCTURED GRAPHIC OBJECTS

Rather than treat graphic data as an unstructured collection of graphic elements (much as a sketch could be considered an unstructured collection of lines and points), the Multics Graphics System deals with structured descriptions of objects.



Sample Graphic Displays

This organization has three advantages:

- Natural representation of most objects can be made in terms of their own inherent organization. For example, a piston, a complex object in its own right, may be treated as an elemental object within a graphic description of an engine.
- Subpictures can be shared, thereby eliminating redundancy.
- Powerful global picture editing capabilities are possible.

PERMANENT GRAPHIC STORAGE

Facilities are provided so that the user can attach a name to any graphic object and store it in a Multics segment. Such objects may be used at any time by any user authorized to access the segment.

TERMINAL-INDEPENDENT GRAPHIC TRANSMISSION

Graphic information is transmitted in a well-defined terminal-independent code. This code may be interpreted by a Multics program and converted to the appropriate codes to drive a graphic terminal; or it may be transmitted directly to an intelligent graphic device that performs its own interpretation, with a corresponding increase in efficiency. It may also be directed to a Multics file and "played back" on any graphic device to form background scenes or standard "canned" pictures. Graphic input sent to a Multics system is converted from its original format into this code before being forwarded to the terminal-independent portion of the system.

SYSTEM COMPATIBILITY

Programs originally written on other computers that make use of the most widely used set of graphic subroutines may, with minimal conversion, interface in the same way with the Multics Graphics with System. Interfaces to mimic other popular graphics systems are easily constructed.

DYNAMIC AND INTERACTIVE GRAPHICS

When used with a terminal of sufficient intelligence, the Multics Graphics System can perform real-time graphic operations, such as dynamic animation, incremental picture update, local picture editing under control of the terminal, and sophisticated graphic input.

SYSTEM REQUIREMENTS

The Multics Graphics System is applicable to any Level 68 Multics configuration.

Specifications may change as design improvements are introduced.

Honeywell

Honeywell Information Systems
In the U.S.A.. 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada. 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.
15602, 3476, Printed in U.S.A.

Honeywell

SERIES 60 (LEVEL 68/DPS)

Multics Electronic Mail Facility

The Level 68/DPS (Multics) electronic mail facility offers its users direct, online, person-to-person distribution of text. It handles mail ranging from brief memos to multivolume documents and delivers that mail immediately to data terminals or online mailboxes. It is one of the many productive personal computing features of the Level 68/DPS.

Honeywell's electronic mail facility operates in conjunction with WORDPRO, the Level 68/DPS word processing system, and automates the creation, distribution, and updating of text information. Electronic mail can significantly reduce the volume of paperwork generated in a typical large organization, dramatically reduce the cost of disseminating documents, and avoid the delays normally associated with physically generating and transporting documents.

The Level 68/DPS electronic mail facility offers definite advantages:

- Each user can create a private "mailbox" – a special storage segment for mail delivery, or mailboxes may be shared for group use.
- Any terminal recognized by the Level 68/DPS can send or receive electronic mail – no special devices are required.
- A user can access his mailbox from any Level 68/DPS recognized terminal at any location.

Electronic mail provides such distinctive functions as:

- Immediate or deferred delivery of messages, text, and other mail
- Broadcast delivery to groups of users
- Selective delivery only to persons with a "need to know"
- A secure mailbox facility, protected by the full range of stringent security controls in Multics system software and hardware, to guard mail from unauthorized access
- Simple commands to prescribe the mailbox access and mail functions allowed for specified users
- Many document management tools for text manipulation and storage

VARIETY OF USES

The Level 68/DPS electronic mail facility has a myriad of business uses. For instance, a large organization typically functions from a number of widely separated locations – corporate headquarters, research and engineering facilities, manufacturing plants, sales and service branches, and test facilities. Because of these widely separated facilities, the slow delivery of vital documents by conventional mail often results in communication problems. Also, the cost of document reproduction and delivery can be considerable. Electronic mail can help solve these communication problems in the following ways.

- After creation, editing, and formatting via WORDPRO, text is distributed instantly to recipients at many locations, thereby reducing distribution delays and costs.
- Online, two-way messages can be exchanged between Level 68/DPS users.
- Mail can be delivered for immediate display at a recipient's terminal or stored in his mailbox, when the recipient's terminal is occupied with an urgent task or not logged onto the system.
- Automatic acknowledgment-of-mail receipt can be requested by the sender.
- A single, online copy of a large document can be maintained for remote perusal, avoiding the problems of reproducing and distributing printed copies.

In addition, management of messages and documents is aided through a variety of tools in Multics. Indexed, ordered archives of past messages can be created, searched, updated, appended, or deleted as desired. The archives can be searched for key sentences or phrases, extra copies can be printed, and old messages and mail can be periodically and automatically deleted when no longer needed.

Text can be examined and updated from several locations at once. For example, during a contract negotiation, a master copy of the contract can be maintained online via WORDPRO. All interested parties – the customer, along with marketing, engineering, legal, finance, etc. can simultaneously review and comment via electronic mail without the delays involved in the conventional mailing of contract review copies.

SECURE MAILBOX

The Level 68/DPS electronic mail facility utilizes a secure online mailbox approach. Each user can have a private mailbox under his home directory, or a group of users can share a common mailbox to receive mail. The contents of each user's mailbox and archive is protected by the extensive security controls of the Multics virtual memory and file management functions. When mail arrives in a user's mailbox, the recipient is immediately notified, if his terminal is online to the Level 68/DPS system. The user can then selectively display the mail with simple commands, save it in mail archive files, or delete it.

TYPES OF ACCESS

Any combination of six different types of mailbox access can be prescribed for a single user or a group of users:

- *add* – allows the addition of new messages to a mailbox. All users are usually given add permission so they can send mail.
- *delete* -- allows the deletion of any message in a mailbox; usually retained for use only by the mailbox owner.
- *read* – allows the user to select and read any messages in a mailbox; usually retained for use only by the mailbox owner.
- *own* – allows the sender to read, modify, or delete messages he originally sent to a mailbox, but doesn't allow access to other messages in the mailbox.
- *status* – allows a mailbox owner to check on the number of his messages, their sender, length, date, etc., in his mailbox, without actually reading them.
- *wakeup* – allows a message to be sent to a mailbox for immediate display, when the mailbox owner is accepting messages. Users can be given this permission to allow direct, user-to-user communication.

SIMPLE COMMANDS

The electronic mail facility utilizes standard Level 68/DPS system commands to send messages and to deliver and receive mail. Other standard file manipulation tools allow maintenance of online message and mail files. These commands can be issued either in full or abbreviated format. Representative Multics commands include:

- *mail (ml)* - sends mail to another system user or prints the mail in the owner's mailbox. A user's mailbox is created automatically under the home directory the first time this command is invoked. Optional arguments allow selective display or deletion of mail.
- *accept_message (am)* – restores the immediate display of messages as received; cancels the defer message command.
- *immediate_message* – requests immediate display of any messages in an owner's mailbox or of any messages received while the owner's terminal is online, unless the defer message is in effect.
- *defer_message (dm)* – prevents any messages received while a user's terminal is online from immediate display and stores them in the user's mailbox.
- *print message (pm)* – displays any messages received in a user's mailbox during a period when messages are not being accepted.
- *send message (sm)* sends a message which is designated by the person and project identities that follow the command.
- *send_message_acknowledge* sends a message to the system user(s), designated by person and project identifiers that follow the command. When the message is displayed at the recipient's terminal, an acknowledgment is returned to the sender.

SYSTEM REQUIREMENTS

The electronic mail facility is integrated with the other facilities of the Level 68/DPS, including data base management, text entry and editing, document formatting, list processing, artwork generation, photocomposition, output control, archive storage tools, and administrative tools. Electronic mail functions on all performance levels of the Level 68/DPS system.

Specifications may change as design improvements are introduced.

Honeywell

Honeywell Information Systems
In the U S A 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D F

20363, 3378, Printed in U.S.A.

APPENDIX C

ARTICLES OF INTEREST

This page has intentionally

been left blank.

In 1964 planning began on the development of a prototype of a computer utility. The aspirations for this system, named Multics (for ***Multi****plexed* ***I****nformation and* ***C****omputing* ***S****ervice), were described in papers presented at the 1965 Fall Joint Computer Conference. Implicit in those papers was the expectation of a later examination of the development effort. From the present vantage point, however, it is clear that a definitive examination is beyond possibility in a single paper; only some of the possible topics can be discussed. First we will review the goals, history and current status of the appearance of the Multics system to its various classes of users. Finally we will describe some of the insights which have come out of the development activities.*

MULTICS: The first seven years

F. J. Corbató
Massachusetts Institute of Technology
Cambridge, MA

C. T. Clingen
Honeywell Information Systems Inc.
Cambridge, MA

J. H. Saltzer
Massachusetts Institute of Technology
Cambridge, MA

FROM THE EDITOR

This overview and history of Multics will be of exceptional interest because it fills most of the requirements for being used as a "software factory", a term that I coined in 1968. It is in fact used for that purpose within Honeywell Information Systems -- in the construction of software systems.

It is particularly pleasing to note that the punch card has disappeared from the view of Multics programmers, although not from everywhere else. As Dr. J. Rabinow (of Optical Character Recognition fame) has observed, if it does it will be the first product he has seen die while on an upward curve of usage!

This article appeared in the Proceedings of the 1972 AFIPS Spring Joint Computer Conference, and is republished (not reprinted) here with the permission of AFIPS and the authors. The work reported herein was sponsored in part by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number N00014-70-A-0362-0001. Reproduction is permitted for any purpose of the United States Government.

INTRODUCTION

In 1964, following implementation of the Compatible Time-sharing System (CTSS) [1,2] serious planning began on the development of a new computer system specifically organized as a prototype of a computer utility. The plans and aspirations for this system, called Multics (for *Multi*plexed *I*nformation and *C*omputing *S*ervice), were described in a set of six papers presented at the 1965 Fall Joint Computer Conference [3-8]. The development of the system was undertaken as a cooperative effort involving the Bell Telephone Laboratories (from 1965 to 1969), the Computer Department of the General Electric Company (subsequently acquired by Honeywell Information Systems Inc.), and Project MAC of M.I.T.

Implicit in the 1965 papers was the expectation that there should be a later examination of the development effort. From the present vantage point, however, it is clear that a definitive examination cannot be presented in a single paper. As a result, the present paper discusses only some of the many possible topics. First we review the goals, history and current status of the Multics project. This review is followed by a brief description of the appearance of the Multics system to its various classes of users. Finally several topics are given which represent some of the research insights which have come out of the development activities. This organization has been chosen in order to emphasize those aspects of software systems having the goals of a computer utility which we feel to be of special interest. We do not attempt detailed discussion of the organization of Multics; that is the purpose of specialized technical books and papers (for example, the essential mechanisms for much of the Multics system are given in books by Organick [9] and Watson [10]).

GOALS

The goals of the computer utility, although stated at length in the 1965 papers, deserve a brief review. By a computer utility it was meant that one had a community computer facility with:

- Convenient remote terminal access as the normal mode of system usage.
- A view of continuous operation analogous to that of the electric power and telephone companies.
- A wide range of capacity to allow growth or contraction without either system or user reorganization.
- An internal file system so reliable that users trust their only copy of programs and data to be stored in it.
- Sufficient control of access to allow selective sharing of information.
- The ability to structure hierarchically both the logical storage of information as well as the administration of the system.
- The capability of serving large and small users without inefficiency to either.
- The ability to support different programming environments and human interfaces within a single system.
- The flexibility and generality of system organization required for evolution through successive waves of technological improvements and the inevitable growth of user expectations.

In an absolute sense the above goals are extremely difficult to achieve. Nevertheless, it is our belief that Multics, as it now exists, has made substantial progress towards achieving each of the nine goals (to the best of our knowledge, the only other attempt to comprehensively attack all of these goals simultaneously is the TSS/30 project at IBM [11,12,13]). Most importantly, none of these goals had to be compromised in any important way.

HISTORY OF THE DEVELOPMENT

As previously mentioned, the Multics project got underway in the Fall of 1964. The computer equipment to be used was a modified General Electric 635 which was later named the 645. The most significant changes made were in the processor addressing and access control logic where paging and segmentation were introduced. A completely new Generalized Input Output Controller was designed and implemented to accommodate the varied needs of devices such as disks, tapes and teletypewriters without presenting an excessive interrupt burden to the processors. To handle the expected paging traffic, a 4-million word (36-bit) high-performance drum system with hardware queueing was developed. The design specifications for these items were completed by Fall 1965, and the equipment became available for software development in early 1967.

Software preparation underwent several phases. The first phase was the development and blocking out of major ideas, followed by the writing of detailed program module specifications. The resulting 3,000 typewritten pages formed the Multics System Programmer's Manual and served as the starting point for all programming. Furthermore, the software designers were expected to implement their own designs. As a general policy PL/I was used as the system programming language wherever possible to maximize lucidity and maintainability of the system [14,15]. This policy also increased the effectiveness of system programmers by allowing each one to keep more of the system within his grasp.

The second major phase of software development, well underway by early 1967, was that of module implementation and unit checkout followed by merging into larger aggregates for integrated testing. Up to then most software and hardware difficulties had been anticipated on the basis of previous experience. But what gradually became apparent as the module integration continued was that there were gross discrepancies between actual and expected performance of the various logical execution paths throughout the software. The result was that an unanticipated phase of design iterations was necessary. These design iterations did not mean that major portions of the system were scrapped without being used. On the contrary, until their replacements could be implemented, often months later, they were crucially necessary to allow the testing and evaluation of the other portions of the system. The cause of the required redesigns was rarely "bad coding", as most of the system programmers were well above average ability. Moreover the redesigns did not mean that the goals of the project were compromised. Rather three recurrent phenomena were observed: 1) typically, specifications representing less-important features were found to be introducing much of the complexity, 2) the initial choice of modularity and interfacing between modules was sometimes awkward, and 3) it was rediscovered that the most important property of algorithms is simplicity rather than special mechanisms for unusual cases. ("In anything at all, perfection is finally attained not when there is no longer anything to add, but when there is no longer anything to take away..." -- Antoine de Saint-Exupéry, *Wind, Sand and Stars,* Quoted with permission of Harcourt Brace Jovanovich, Inc.)

The reason for bringing out in detail the above design iteration experience is that frequently the planning of large software projects still does not properly take the need for continuing iteration into account. And yet we believe that design iterations are a required activity on any large scale system which attempts to break new conceptual ground such that individual programmers cannot comprehend the entire system in detail. For when new ground is broken, it is usually impossible to deduce the consequent system behavior except by experimental operation. Simulation is not particularly effective when the system concepts and user behavior are new. Unfortunately one does not understand the system well enough to simplify it correctly and thereby obtain a manageable model which requires less effort to implement than the system itself. Instead one must develop a different view:

- The initial program version of a module should be viewed only as the first complete specification of the module and should be subject to design review *before* being debugged or checked out.

- Module design and implementation should be based upon an assumption of periodic evaluation, redesign, and evolution.

In retrospect, the design iteration effect was apparent even in the development of the earlier Compatible Timesharing System (CTSS), when a second file system with many functional improvements turned out to have poor performance when initially installed. A hasty design iteration succeeded in rectifying the matter but the episode at the time was viewed as an anomaly perhaps due to inadequate technical review of individual programming efforts.

CURRENT STATUS

In spite of the unexpected design iteration phase, the Multics system became sufficiently effective by late 1968 to allow system programmers to use the system while still developing it. By 1969 October, the system was made available for general use on a "cost-recovery" charging basis similar to that used for other major computation facilities at M.I.T. Multics is now the most widely used timesharing system at M.I.T., supporting a user community of some 500 registered subscribers. The system is currently operated for users 22 hours per day, 7 days per week. For at least eight hours each day the system operates with two processors and three memory modules containing a total of 384 K (K=1024) 36-bit words. This configuration currently is rated at a capacity of about 55 fairly demanding users such that most trivial requests obtain response in one to five seconds. (Future design iterations are expected to increase the capacity rating.) Several times a day during the off-peak usage hours the system is dynamically reconfigured into two systems: a reduced capacity service system and an independent development system. The development system is used for testing those hardware and software changes which cannot be done under normal service operation.

The reliability of the round-the-clock system operation described above has been a matter of great concern, for in any online real-time system the impact of mishaps is usually far more severe than in batch processing systems. In an online system especially important considerations are:

- the time required before the system is usable again following a mishap

- the extra precautions required for restoring possibly lost files

- the psychological stress of breaking the interactive dialogue with users who were counting on system availability

Because of the importance of these considerations, careful logs are kept of all Multics "crashes" (i.e., system service disruption for all active users) at M.I.T. in order that analysis can reveal their causes. These analyses indicate currently an average of between one and two crashes per 24 hour day: These crashes have no single cause. Some are due to hardware failures, others to operator error and still others to software bugs introduced during the course of development. At the two other sites where Multics is operated, but where active system development does not take place, there have been almost no system failures traced to software.

Currently the Multics system, including compilers, commands, and subroutine libraries, consists of about 1500 modules, averaging roughly 200 lines of PL/I apiece. These compile to produce some 1,000,000 words of procedure code. Another system measure is the size of the resident supervisor which is about 30 K words of procedure and, for a 55 user load, about 36 K words of data and buffer areas.

Because the system is so large, the most powerful maintenance tool available was chosen--the system itself. With all of the system modules stored online, it is easy to manipulate the many components of different versions of the system. Thus it has been possible to maintain steadily for the last year or so a pace of installing 5 or 10 new or modified system modules a day. Some three-quarters of these changes can be installed while the system is in operation. The remainder, pertaining to the central supervisor, are installed in batches once or twice a week. This online maintenance capability has proven indispensible to the rapid development and maintenance of Multics since it permits constant upgrading of the user interface without interrupting the service. We are just beginning to see instances of user-written applications which require this same capability so that the application users need not be interrupted while the software they are using is being modified.

The software effort which has been spent on Multics is difficult to estimate. Approximately 150 man-years were applied directly to design and system programming during the "development-only" period of Table I. Since then we estimate that another 50 man-years have been devoted to improving and extending the system. But the actual cost of a single successful system is misleading, for if one starts afresh to build a similar system, one must compensate for the non-zero probability of failure.

THE APPEARANCE OF MULTICS TO ITS USERS

Having reviewed the background of the project, we may now ask who are the users of the Multics system and what do the facilities that Multics provides mean to these users. Before answering, it is worth describing the generic user as "viewed" by Multics. Although from the system's point of view all users have the same general characteristics and interface with it uniformly, no single human interface represents the Multics machine. That machine is determined by each user's initial procedure coupled with those functions accessible to him. Thus there exists the potential to present each Multics user with a unique external interface.

However, Multics does provide a native internal program environment consisting of a stack-oriented, pure-procedure, collection of PL/I procedures imbedded in a segmented virtual memory containing all procedures and data stored online. The extent to which some, all, or none of this internal environment is visible to the various users is an administrative choice.

The implications of these two views--both the external interface and the internal programming environment--are discussed in terms of the following categories of users:

- System and user application programmers responsible for writing system and user software.
- Administrative personnel responsible for the management of system resources and privileges.
- The ultimate users of application systems.
- Operations and hardware maintenance personnel responsible, respectively, for running the machine room and maintaining the hardware.

Multics as Viewed by System and Subsystem Programmers

The machine presented to both the Multics system programmer and the application system programmer is the one with which we have the most experience; it is the raw material from which one constructs other environments. It is worth re-emphasizing that the only differentiation between Multics system programmers and user programmers is embodied in the access control mechanism which determines what online information can be referenced; therefore, what are apparently two groups of users can be discussed as one.

Major interfaces presented to programmers on the Multics system can be classified as the program preparation and documentation facilities and the program execution and debugging environment. They will be touched upon briefly, in the order used for program preparation.

■ Program Preparation and Documentation

The facilities for program preparation on Multics are typical of those found on other timesharing systems, with some shifts in emphasis. (See the Appendix.) For example, programmers consider the file system sufficiently invulnerable to physical loss that it is used casually and routinely to save all information. Thus, the punched card has vanished from the work routine of Multics programmers and access to one's programs and the ability to work on them are provided by the closest terminal.

System	Development Only	Development + Use	Use Only
CTSS	1960-1963	1963-1965	1965-present
Multics	1964-1969	1969-present	

Table I. A comparison of the system development and use periods of CTSS and Multics. The Multics development period is not significantly longer than that for CTSS despite the development of about 10 times as much code for Multics as for CTSS and a geographically distributed staff. Although reasons for this similarity in time span include the use of a higher-level programming language and a somewhat larger staff, the use of CTSS as a development tool for Multics was of pivotal importance.

As another example, the full ASCII character set is employed in preparing programs, data, and documentation, thereby eliminating the need for multiple text editors, several varieties of text formatting and comparison programs, and multiple facilities for printing information both online and offline. This generalization of user interfaces facilitates the learning and subsequent use of the system by reducing the number of conventions which must be mastered.

Finally, because the PL/I compiler is a large set of programs, considerable attention was given to shielding the user from the size of the compiler and to aiding him in mastering the complexities of the language. As in many other timesharing systems, the compiler is invoked by issuing a simple command line from a terminal exactly as for the less ambitious commands. No knowledge is required of the user regarding the various phases of compilation, temporary files required, and optional capabilities for the specialist; explanatory "sermons" diagnosing syntactic errors are delivered to the terminal to effect a self-teaching session during each compilation. To the programmer, the PL/I compiler is just another command.

■ Program Execution Environment

Another set of interfaces is embodied in the implementation environment seen by PL/I programmers. This environment consists of a directly addressable virtual memory containing the entire hierarchy of online information, a dynamic linking facility which searches this hierarchy to bind procedure references, a device-independent input/output [16] system (the Michigan Terminal System [17] has a similar device-independent input/output system) and program debugging and metering facilities. These facilities enjoy a symbiotic relationship with the PL/I procedure environment used both to implement them and to implement user facilities co-existing with them. Of major significance is that the natural internal environment provided and required by the system is exactly that environment expected by PL/I procedures. For example, PL/I pointer variables, call and return statements, conditions, and static and automatic storage all correspond directly to mechanisms provided in the internal environment. Consequently, the system supports PL/I code as a matter of course.

The main effect of the combination of these features is to permit the implementer to spend his time concentrating on the logic of his problem; for the most part he is freed from the usual mechanical problems of storage management and overlays, input/output device quirks, and machine-dependent features.

■ Some Implementation Experience

The Multics team began to be much more productive once the Multics system became useful for software development. A few cases are worth citing to illustrate the effectiveness of the implementation environment. A good example is the current PL/I compiler, which is the third one to be implemented for the project, and which consists of some 250 procedures and about 125 K words of object code. Four people implemented this compiler in two years, from start to first general use. The first version of the Multics program debugging system, composed of over 3,000 lines of source code, was usable after one person spent some six months of nights and weekends "bootlegging" its implementation. As a last example, a facility consisting of 50 procedures with a total of nearly 4,000 PL/I statements permitting execution of Honeywell 635 programs under Multics became operational after one person spent eight months learning about the GCOS operating system for the 635 PL/I, and Multics, and then implemented the environment. In each of these examples the implementation was accomplished from remote terminals using PL/I.

Multics users have discovered that it is possible to get their programs running very quickly in this environment. They frequently prepare "rough drafts" of programs, execute them, and then improve their overall design and operating strategy using the results of experience obtained during actual operation. As an example, again drawn from the implementation of Multics, the early designs and implementations of the programs supporting the virtual memory [18] made over-optimistic use of variable-sized storage allocation techniques. The result was a functionally correct but inadequately performing set of programs. Nevertheless, these modules were used as the foundation for subsequent work for many months. When they were finally replaced with modules using simplified fixed-size storage techniques, performance improvements of over an order of magnitude were realized. This technique emphasizes two points: first, it is frequently possible to provide a practical, usable facility containing temporary versions of programs; second, often the insight required to significantly improve the behavior of a program comes only after it is studied in operation. As implied in the earlier discussion of design iteration, our experience has been that structural and strategic changes rather than "polishing" (or recoding in assembly language) produce the most significant performance improvements.

In general, we have noticed a significant "amplifier" or "leverage" effect with the use of an effective online environment as a system programming facility. Major implementation projects on the Multics system seldom involve more than a few programmers, thereby easing the management and communications problems usually entailed by complex system implementations. As would be expected, the amplification effect is most apparent with the best project personnel.

Administration of Multics Facilities and Resources

The problem of managing the capabilities of a computer utility with geographically dispersed subscribers leads to a requirement of decentralized administration. At the apex of an administrative pyramid resides a system administrator with the ability to register new users, confer resource quotas, and generate periodic bills for services rendered. The system administrator deals with user groups called projects. Each group can in turn designate a project administrator who is delegated the authority to manage a budget of system resources on behalf of the project. The project administrator is then free to deal directly with project members without further intervention from the system administrator, thereby greatly reducing the bottlenecks inherent in a completely centralized administrative structure.

■ Environment Shaping

In addition to having immediate control of such resources as secondary storage, port access, and rate of processor usage, the project administrator is also able to define or shape the environment seen by the members of his project when they log into the system. He does this by defining those procedures that can be accessed by members of his project and by specifying the initial procedure executed by each member of his project when he logs in. This environment shaping facility has led to the notion of a private project subsystem on Multics. It combines the administrative and programming facilities of Multics so that a project administrator and a few project implementers can build, maintain, and evolve environments entirely on their own. Thus, some subsystems bear no internal resemblance to the standard Multics procedure environment.

For example, the Dartmouth BASIC [19] compiler executes in a closed subsystem implemented by an M.I.T. student group for use by undergraduate students. The compiler, its object code, and all support routines execute in a simulation of the native environment provided at Dartmouth. The users of this subsystem need little, if any, knowledge of Multics and are able to behave as if logged into the Dartmouth system proper. Other examples of controlled environment subsystems include one to permit many programs which normally run under the GCOS operating system to also run unmodified in Multics. Finally, an APL [20] subsystem allows the user to behave for the most part as if he were logged into an APL machine. The significance of these subsystems is that their implementers did not need to interact with the system administrator or to modify already existing Multics capabilities. The administrative facilities permit each such subsystem to be offered by its supporters as a private service with its own group of users, each effectively having its own private computer system.

Other Multics Users

Finally, we observe that the roles of the application user, the system operators and the hardware maintainers as seen by the system are simply those of ordinary Multics users with specialized access to the online procedures and data. The effect of this uniformity of treatment is to reduce greatly the maintenance burden of the system control software. One example, of great practical importance, has been the ease with which system performance measurement tools have been prepared for use by the operating staff.

INSIGHTS

So far, we have discussed the status and appearance of the Multics system. A further question is what has been learned in the construction of Multics which is of use to the designers of other systems. Having a bright idea which clearly solves a problem is not sufficient cause to claim a contribution if the idea is to be part of a complex system. In order to establish the real feasibility of an idea, all of its implications and consequences must be followed out. Much of the work on Multics since 1965 has involved following out implications and consequences of the many ideas then proposed for the prototype computer utility. That following out is an essential part of proof of ideas is attested by the difficulties which have been encountered in other engineering efforts such as the development of nuclear fusion power plants and the electric automobile. Not all proposals work out; for example, extended attempts to engineer an atomic powered airplane suggest infeasibility.

Perhaps Multics' most significant single contribution to the state of the art of computer system construction is the demonstration of a large set of fully implemented ideas in a working system. Further, most of these ideas have been integrated without straining the overall design; most additional proposals would not topple the structure. Ideas such as virtual memory access to online storage, parallel process organization, routine but controlled information sharing, dynamic linking of procedures, and high-level language implementation have proven remarkably compatible and complementary.

To illustrate some of the areas of progress in understanding of system organization and construction which have been achieved in Multics, we consider here the following five topics:

- Modular division of responsibility
- Dynamic reconfiguration
- Automatically managed multilevel memory
- Protection of programs and data
- System programming language

Modular Division of Responsibility

Early in the design of Multics a decision had to be made whether or not to treat the segmented virtual memory as a separately usable "feature", independent of a traditionally organized read/write type file system. The alternative, to use the segmented virtual memory as the file system itself, providing the illusion of direct "in-core" access to all online storage, was certainly the less conservative approach (Fig. 1). The second approach, which was the one chosen, led to a strong test of the ability of a computing system to support an apparent one-level memory for an arbitrarily large information base. It is interesting that the resulting almost total decoupling between physical storage allocation and data movement on the one hand and directory structure, naming, and file organization on the other led to a remarkably simple and functionally modular structure for that part of the system [18] (Fig. 2).

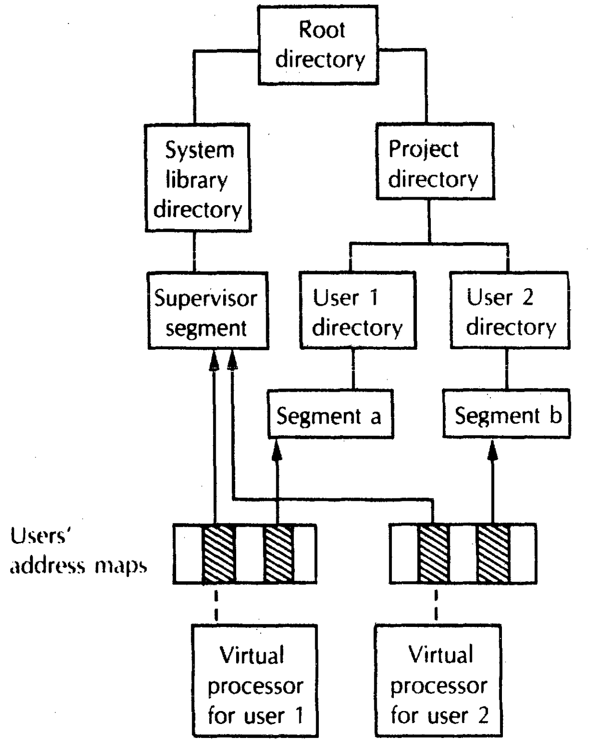


Figure 1. The entire storage hierarchy may be mapped into individual user process address spaces (see arrows) as if contained in a primary memory. Illustrated are the sharing of a supervisor segment by two users and private access to segments a and b. The necessary primary storage is simulated by a demand paging technique which moves information between the real primary memory and secondary storage.

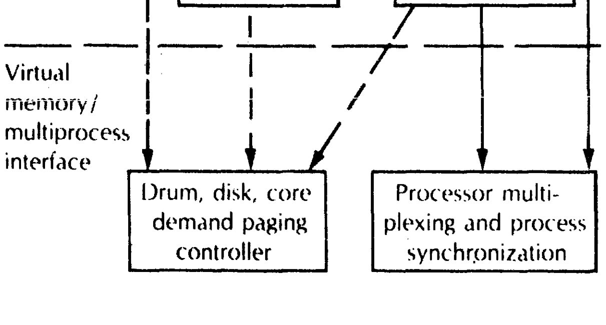

User programs and command/subroutine library
General user interface
Directory address space management
User I/O device control and buffering
Virtual memory/ multiprocess interface
Drum, disk, core demand paging controller
Processor multiplexing and process synchronization

Figure 2. Major lines of modular division in Multics. Solid lines indicate calls for services. Dotted lines indicate implicit use of the virtual memory.

Another area of Multics in which a high degree of functional modularity was achieved was in scheduling, multiprogramming, and processor management. Because harnessing of multiple processors was an objective from the beginning, a careful and methodical approach to multiplexing processors, handling interrupts, and providing interprocess synchronizing primitives was developed. The resulting design, known as the Multics traffic controller, absorbed into a single, simple module a set of responsibilities often diffused among a scheduling algorithm, the input/output controlling system, the online file management system, and special purpose inter-user communication mechanisms [21].

Finally, with processor management and online storage management uncoupled into well-isolated modules, the Multics input/output system was left with the similarly isolatable function of managing streams of data flowing from and to source and sink type devices [16]. Thus, this section of the system concentrates only on switching of the streams, allocation of data buffering areas, and device control strategies.

Each of the divisions of labor described above represents an interesting result primarily because it is so difficult to discover appropriate divisions of complex systems. (See Dijkstra [22] for a further discussion of this point). Establishing that a certain proposed division results in simplicity, creates an uncluttered interface, and does not interfere with performance, is generally cause for a minor celebration.

Dynamic Reconfiguration

If the computer utility is ever to become as much a reality as the electric power utility or the telephone communication service, its continued operation must not be dependent upon any single physical component, since individual components will eventually require maintenance. This observation leads an electric power utility to provide procedures whereby an idle generator may be dynamically added to the utility's generating capacity, while another is removed for maintenance, all without any disruption of service to customers. A similar scenario has long been proposed for multiprocessor, multimemory computer systems, in which one would dynamically switch processors and memory boxes in and out of the operating configuration as needed. Unfortunately, though there have been demonstrated a few "special purpose" designs (an outstanding example is the American Airlines SABRE system [23]) it has not been apparent how to provide for such operations in a general purpose system. A recent thesis [24] proposed a general model for the dynamic binding and unbinding of computation and memory structures to and from ongoing computations. Using this model as a basis, the thesis also proposed a specific implementation for a typical multiprocessor, multimemory computing system. One of the results of this work was the addition to the operating Multics system of the capability of dynamically adding and removing central processors and memory modules as in Figure 3. The usefulness of the idea may be gauged by observing that at M.I.T. five to ten such reconfigurations are performed in a typical 24-hour operating day. Most of the reconfigurations are used to provide a secondary system for Multics development.

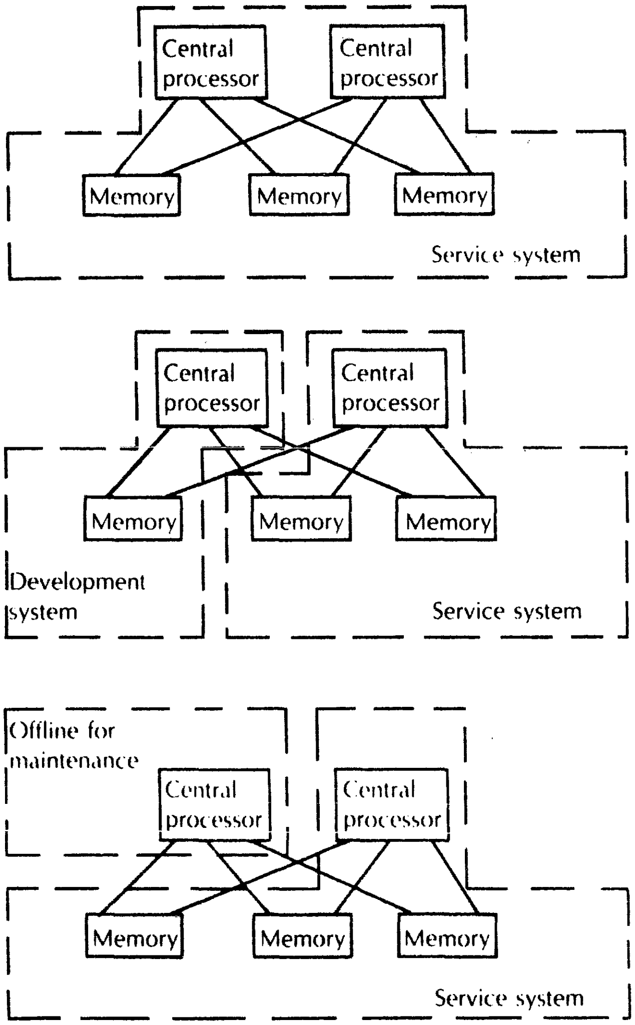


Figure 3. Dynamic reconfiguration permits switching among the three typical operating configurations shown here, without currently logged-in users being aware that a change has taken place.

Automatically Managed Multilevel Memory

By now it has become accepted lore in the computer system field that the use of automatic management algorithms for memory systems constructed of several levels with different access times can provide a significant reduction of user programming effort. Examples of such automatic management strategies include the buffer memories of the IBM system 370 models 155, 165, and 195 [25] and the demand paging virtual memories of Multics, IBMs CP-67 [26] and the Michigan Terminal System [17]. Unfortunately, behind the mask of acceptance hides a worrisome lack of knowledge about how to engineer a multilevel memory system with appropriate strategy algorithms which are matched to the load and hardware characteristics. One of the goals of the Multics project has been to instrument and experiment with the multilevel memory system of Multics, in order to learn better how to predict in advance the performance of proposed new automatically managed multilevel memory systems. Several specific aspects of this goal have been explored:

- A strategy to treat core memory, drum, and disk as a three-level system has been proposed, including a "least-recently-used" algorithm for moving information from drum to disk. Such an algorithm has been used for some time to determine which pages should be removed from core memory [27]. The dynamics of interaction among two such algorithms operating at different levels are weakly understood, and some experimental work should provide much insight. The proposed strategy will be implemented, and then compared with the simpler present strategy which never moves things from drum to disk, but instead makes educated "guesses" as to which device is most appropriate for the permanent residence of a given page. If the automatic algorithm is at least as good as the older, static one, it would represent an improvement in overall design by itself, since it would automatically track changes in user behavior, while the static algorithm requires attention to the validity of its guesses.

- A scheme to permit experimentation with predictive paging algorithms was devised. The scheme provides for each process a list of pages to be preloaded whenever the process is run, and a second list to be immediately purged whenever the process stops. The updating of these lists is controlled by a decision table exercised every time the process stops running. As every page of the Multics virtual memory is potentially shared, the decision table represents a set of heuristics designed to separate out those which are probably not being shared at the moment.

- A series of measurements was made to establish the effectiveness of a small hardware associative memory used to hold recently accessed page descriptors. These measurements established a profile of hit ratio (probablity of finding a page descriptor in the associative memory) versus associative memory size which should be useful to the designer of virtual memory systems [28].

- A set of models, both analytic and simulation, was constructed to try to understand program behavior in a virtual memory. So far, two results have been obtained. One is the finding that a single program characteristic (the mean execution time before encountering a "missing" page in the virutal memory as a function of memory size) suffices to provide a quite accurate prediction of paging and idle overhead times. The second is a direct calculation of the distribution of response times under multiprogramming. Having available the entire response time distribution, rather than just averages, permits estimation of the variance and 90-percentile points of the distribution, which may be more meaningful than just the average. A doctoral thesis is in progress on this topic.

Although the immediate effect of each of these investigations is to improve the understanding or performance of the current version of Multics, the long-range payoff in methodical engineering using better understood memory structures is also evident.

Protection of Programs and Data

A long-standing objective of the public computer utility has been to provide facilities for the protection of executing programs from one another, so that users may with confidence place appropriate control on the release of their private information. In 1967, a mechanism was proposed [29] and implemented in software which generalized the usual supervisor-user protection relationship. This mechanism, named "rings of protection", provides user-written subsystems with the same protection from other users that the supervisor has, yet does not require that the user-written subsystem be incorporated into the supervisor. Recently, this approach was brought under intense review, with two results:

- A hardware architecture which implements the mechanism was proposed [30]. One of the chief features of the proposed architecture is that subroutine calls from one protection ring to another use exactly the same mechanisms as do subroutine calls among procedures within a protection area. The proposal appears sufficiently promising that it is included in the specifications for the next generation of hardware to be used for Multics.

- As an experiment in the feasibility of a multilayered supervisor, several supervisor procedures which required protection, but not all supervisor privileges, were moved into a ring of protection intermediate between the users and the main supervisor. The success of this experiment established that such layering is a practical way to reduce the quantity of supervisor code which must be given all privileges.

Both of these results are viewed as steps toward first, a more complete exploitation and understanding of rings of protection, and later, a less constrained organization of the type suggested by Evans and LeClerc [31] and by Lampson [32]. But more importantly, rings of protection appear applicable to any computer system using a segmented virtual memory. Two doctoral theses are underway in this area.

System Programming Language

Another technique of system engineering methodology being explored within the Multics project is that of higher level programming language for system implementation. The initial step in this direction (which proved to be a very big step) was the choice of the PL/I language for the implementation of Multics. By now, Multics offers an extensive case study in the viability of this strategy. Not only has the cost of using a higher level language been acceptable, but increased maintainability of the software has permitted more rapid evolution of the system in response to development ideas as well as user needs. Three specific aspects of this experience have now been completed:

- The transition from an early PL/I subset compiler [14] to a newer compiler which handles almost the entire language was completed. This transition was carried out with performance improvement in practically every module converted in spite of the larger language involved. The significance of the transition is the demonstration that it is not necessary to narrow one's sights to a "simple" subset language for system programming. If the language is thoroughly understood, even a language as complex as the full PL/I can be effectively used. As a result, the same language and compiler provided for users can also be used for system implementation, thereby minimizing maintenance, confusion, and specialization.

- Notwithstanding the observation just made, the time required to implement a full PL/I compiler is still too great for many situations in which the compiler implementation cannot be started far enough in advance of system coding. For this reason, there is considerable interest in defining a smaller language which is easily compilable, yet retains the features most important for system implementation. On the basis of the experience of programming Multics in a subset of PL/I, such a language was defined but not implemented, since it was not needed [33].

- A census of Multics system modules reveals how much of the system was actually coded in PL/I, and reasons for use of other languages. Roughly, of the 1500 system modules, about 250 were written in machine language. Most of the machine language modules represent data bases or small subroutines which execute a single privileged instruction. (No attempt was made to provide either a data base compiler or PL/I built-in functions for specialized hardware needs.) Significantly, only a half dozen areas (primarily in the traffic controller, the central page fault path, and interrupt handlers) which were originally written in PL/I have been recoded in machine language for reasons of squeezing out the utmost in performance. Several programs, originally in machine language, have been recoded in PL/I to increase their maintainability.

As with the earlier topics, the implications of this work with PL/I should be felt far beyond the Multics system. Most implementers, when faced with the economic uncertainties of a higher-level language, have chosen machine language for their central operating systems. The experience of using PL/I for Multics, when added to the expanding collection of experience elsewhere [34], should help reduce the uncertainty.

In a research project as large, long, and complex as Multics, any paper such as this must necessarily omit many equally significant ideas, and touch only a few which may happen to have wide current interest. It is the purpose of individual and detailed technical papers to explain these and other ideas more fully. The bibliography found in reference [35] contains over twenty such technical papers.

IMMEDIATE FUTURE PLANS

The Multics software is continuing to evolve in response to user needs and improved understanding of its organization. In 1972 a new hardware base for Multics will be installed by the Information Processing Center at M.I.T. for use by the M.I.T. computing community. This program compatible hardware base contains small but significant architectural extensions to the current hardware. The circuit technology used will be that of the Honeywell 6080 computer. The substantial changes include:

- Replacement of the paging drum initially with bulk core and, when available, LSI memory.

- Implementation of rings of protection as part of the paging and segmentation hardware.

Wherever possible the strategy of using off-the-shelf standard equipment rather than specially engineered units for Multics has been followed. This strategy is intended to simplify maintenance.

CONCLUSIONS

There are many conclusions which could possibly be drawn from the experience of the Multics project. Of these, we consider four to be major and worthy of note. *First*, we feel it is clear that it is possible to achieve the goals of a prototype computer utility. The current implementation of Multics provides a measure of the mechanisms required. Moreover, the specific implementation of the system, because it has been written in PL/I, forms a model for other system designers to draw upon when constructing similar systems.

Second, the question of whether or not the specific software features and mechanisms which were postulated for effective computer utility operation are desirable has now been tested with specific user experience. Although the specific mechanisms implemented subsequently may be superseded by better ones, it is certainly clear that the improvement of the user environment which was wanted has been achieved.

Third, systems of the computer utility class must evolve indefinitely since the cost of starting over is usually prohibitive and the many-year lead time required may be equally unacceptable. The requirement of evolvability places stringent demands on design, maintainability, and implementation techniques.

Fourth and finally, the very act of creating a system which solves many of the problems posed in 1965 has opened up many new directions of research and development. It would appear almost a certainty that increased user aspirations will continue to require intensive work in the areas of computer system principles and techniques.

In closing, perhaps we should take note that in the seven years since Multics was proposed, a great many other systems have also been proposed and constructed; many of these have developed similar ideas. Some examples which have not already been mentioned include:

- the TENEX system of Bolt, Beranek and Newman
- the VENUS system of Mitre Corp.
- the MU5 at Manchester University
- RC-4000 of Regnecentralen
- 5020 TSS of Hitachi Corp.
- DIPS-1 of Nippon Telephone
- the Japanese National Computer Project
- the PDP-10/50 TSS of Digital Equipment Corp.
- the CBB-500 of Berkeley Computer Corp.
- I.T.S. of the M.I.T. Artificial Intelligence Laboratory
- Exec-8 of Univac
- System 3 and 7 and the SPECTRA 70/46 of RCA
- Star-100 of CDC
- UTS of Xerox Data Systems
- the 6700 system of Burroughs
- the Dartmouth Timesharing System

In most cases, their designers have developed effective implementations which are directed to a different interpretation of the goals, or to a smaller set of goals than those required for the complete computer utility. This diversity is valuable, and probably necessary, to accomplish a thorough exploration of many individually complex ideas, and thereby to meet a future which holds increasing demand for systems which embrace the totality of computer utility requirements.

ACKNOWLEDGEMENT

It is impossible to acknowledge accurately the contributions of all the individuals or even the several organizations which have given various forms of support to the development of Multics over the past seven years. As would be expected of any multiorganization project spanning several years, there has been a turnover in the personnel involved. As the individual contributors now number in the hundreds, proper recognition cannot be given here. Instead, since the development of significant features and designs of Multics has occurred in specific areas and disciplines such as input/output, virtual memory design, languages, and resource multiplexing, a more accurate delineation of achievements should be made in specialized papers. So in the end we must defer to the authors of individual papers, past and future, to acknowledge the efforts of some of the many contributors who have made the evolution of Multics possible.

APPENDIX: A CHECKLIST OF MULTICS FEATURES

Following is a checklist of currently available features and facilities of Multics. Although many of the features are described in cryptic and untranslated local jargon, one can at least obtain a feel for the range of facilities now provided. Further information on most of these features may be found in the Multics Programmers' Manual [35].

CHECKLIST OF MULTICS FEATURES

Interactive Time-Sharing Facilities
- file editors
- file manipulation (rename/move/delete)
- personal command abbreviations
- recursive command language
- source language debugging with breakpoints
- subroutine call tracer
- can stop any running command or program

Programming Languages
- PL/I
- FORTRAN
- BASIC (The BASIC system and the Dartmouth environment were developed at Dartmouth College. Used at M.I.T. by permission of Dartmouth College.)
- APL
- LISP
- BCPL
- ALM (assembly language/Multics)

Information Storage System
- configuration independent
- accessed through virtual memory (segments)
- access control lists by user and project
- links to segments of other users
- hierarchical directory (catalog) arrangements
- public library facilities
- sharing at all levels
- multiple segment names (synonyms)
- separate control of read, write, and execute

Programming Environment
- segmented virtual memory
- dynamic linking of procedures and data, or prelinking
- interprocess communication
- independent of configuration
- uniform error handling mechanism
- user definable protection rings
- microsecond calendar clock with interrupt
- program interrupt signal from console

Input and Output
- standard typewriter interface for device independence
- ASCII character set used throughout
- input characters converted to canonical form
- erase and kill editing on typed input
- I/O streams switchable during execution
- magnetic tape, printer, card punch, card reader
- typewriter terminals (IBM 2741, 1050; Teletype 37, 33, 35; Dura; Datel; Execuport; Terminet-300)
- graphic support library (devices: ARDS, IMLAC, DEC 338)
- ARPA network
- interfaces at three levels (formatted data coversion; bit stream control; full device control)

Management Facilities
- passwords required for login
- project may interpose authentication procedure
- decentralized projects
- accounting, billing, and quotas
- online probing and account adjustment
- operator or system initiated logout of users
- unlisted and anonymous users
- limited service system
- dynamic reconfiguration of memories and processors
- system performance metering for parameter adjustment
- project-imposed starting procedure

Communication Facilities
- interuser mail
- help command, help files
- message of the day
- online error reporting and consultation service
- online user graffiti board
- operations message broadcast to logged-in users

Absentee Facilities
- priority/defer queues for printer, card punch
- queued translator facility
- general absentee job facility

Reliability Measures
- weekly file copies onto tape
- daily disk/drum copy onto tape
- incremental file copies onto tape, 1/2 hour behind use
- salvager to clean up files after system crash
- emergency shutdown entry to system

Maintenance Features
- online library change, no disruption of current users
- entire system source online, maintenance tools
- system checkout on small hardware configuration
- online performance monitoring of multiprogramming (paging traffic; drum/disk usage; typewriter traffic)
- user performance feedback (cpu time and paging load on each command; page trace always operating; subroutine call counters)

Private Project Subsystems
- project-providable command interface
- Dartmouth environment (The BASIC system and the Dartmouth environment were developed at Dartmouth College. Used at M.I.T. by permission)
- student environment

Miscellaneous Facilities
- desk calculators
- sort command
- memorandum formatting and typing subsystem
- user-provided list of programs to be automatically executed when user logs in
- GCOS environment

1. F.J.Corbató, M.M.Daggett and R.C.Daley, "An Experimental Timesharing System", *AFIPS Conf. Proc.* **21**, Spartan Books, Washington, DC, 1962, pp. 335-344.

2. P.A.Crisman, "The Compatible Timesharing System: A Programmer's Guide", 2nd Ed., M.I.T. Pr., Cambridge, MA, 1965.

3. F.J.Corbató, and V.A.Vyssotsky, "Introduction and Overview of the Multics System", *Proc. AFIPS Fall Joint Comput. Conf.* **27**, Spartan Books, Washington, DC, 1965, pp. 185-196.

4. E.L.Glaser, et al, "System Design of a Computer for Timesharing Application", *Proc. AFIPS Fall Joint Comput. Conf.*, **27**, Spartan Books, Washington, DC, 1965, pp. 197-202

5. V.A.Vyssotsky, et al, "Structure of the Multics Supervisor", *Proc. AFIPS Fall Joint Comput. Conf.* **27**, Spartan Books, Washington, DC, 1965, pp. 203-212.

6. R.C.Daley and P.G.Neumann, "A General-Purpose File System for Secondary Storage", *Proc. AFIPS Fall Joint Comput. Conf.* **27**, Spartan Books, Washington, DC, 1965, pp. 213-229.

7. J.F.Ossanna, et al, "Communication and Input/Output Switching in a Multiplex Computing System", *Proc. AFIPS Fall Joint Comput. Conf.* **27**, Spartan Books, Washington, DC, 1965, pp. 230-242.

8. E.E.David, Jr. and R.M.Fano, "Some Thoughts About the Social Implications of Accessible Computing", *Proc. AFIPS Fall Joint Comput. Conf.* **27**, Spartan Books, Washington, DC, 1965, pp. 243-247.

9. E.I.Organick, *The Multics System: An Examination of its Structure*, M.I.T. Pr., Cambridge, MA and London, England.

10. R.W.Watson, *Timesharing System Design Concepts*, McGraw, NY, 1970.

11. W.T.Comfort, "A Computing System Design for User Service", *Proc. AFIPS Fall Joint Comput. Conf.* **27**, Spartan Books, Washington, DC, 1965, pp. 619-626.

12. A.S.Lett and W.L.Konigsford, "TSS/360: A Timeshare Operating System", *Proc. AFIPS Fall Joint Comput. Conf.* **33**, Thompson Books, 1968, pp. 14-28.

13. R.E.Schwemm, "Experience Gained in the Development and Use of TSS/360", *Proc. AFIPS Spring Joint Comput. Conf.* **40**, AFIPS Pr., Montvale, NJ, (1972).

14. F.J.Corbató, "PL/I as a Tool for System Programming", *Datamation* **15**, No. 6, 68-76 (1969).

15. R.A.Freiburghouse, "The Multics PL/I Compiler", *Proc. AFIPS Fall Joint Comput. Conf.* **35**, AFIPS Pr., 1969, pp. 187-199.

16. R.J.Feiertag and E.I.Organick, "The Multics Input-Output System", *ACM Third Symp. on Operating Syst. Principles.*, 35-41 (1971).

17. M.T.Alexander, "Organization and Features of the Michigan Terminal System", *Proc. AFIPS Spring Joint Comput. Conf.* **40**, AFIPS Pr., Montvale, NJ, (1972)

18. A.Bensoussan, C.T.Clingen and R.C.Deley, "The Multics Virtual Memory", *ACM Second Symp. on Operating Syst. Principles,* Princeton Univ., 30-42 (1969).

19. *BASIC,* 5th Ed., Kiewit Computation Center, Dartmouth College (1970).

20. *APL/360 User's Manual,* IBM form GH20-0683-1 (1970).

21. J.H.Salzer, "Traffic Control in a Multiplexed Computer System", Sc.D. Thesis, M.I.T. Dept. of Electrical Engineering, (1966). Also available as Proj. MAC Tech. Report TR-30.

22. E.W.Dijkstra, "The Structure of the 'THE' Multiprogramming System", *Commun. ACM* **11**, No. 5, 341-346 (1968).

23. R.W.Parker, "The Sabre System", *Datamation* **11**, No. 9, 49-52 (1965).

24. R.R.Schell, "Dynamic Reconfiguration in a Modular Computer System", Ph.D. Thesis, M.I.T. Department of Electrical Engineering (1971). Available as Proj. MAC Tech. Report TR-86.

25. C.J.Conti, "Concepts for Buffer Storage", *IEEE Comput. Group News.,* 9-13 (1969).

26. R.A.Meyer and L.H.Seawright, "A Virtual Machine Timesharing System", *IBM Syst. J.* **9**, No. 3, 199-218 (1970).

27. F.J.Corbató, "A Paging Experiment with the Multics System", In Honor of P.M.Morse, M.I.T. Pr., Cambridge, MA, 1969, pp. 217-228.

28. M.D.Schroeder, "Performance of the GE-645 Associative Memory While Multics is in Operation", *ACM Workshop on Syst. Performance Evaluation,* 227-245 (1971).

29. R.M.Graham, "Protection in an Information Processing Utility", *Commun. ACM* **11**, No. 5, 365-369 (1968).

30. M.D.Schroeder and J.H.Saltzer, "A Hardware Architecture for Implementing Protection Rings", *ACM Third Symp. on Operating Syst. Principles,* 42-54 (1971).

31. D.C.Evans and J.Y.LeClerc, "Address Mapping and the Control of Access in an Interactive Computer", *Proc. AFIPS Spring Joint Comput. Conf.* **30**, Thompson Books, 1967, pp. 23-30

32. B.W.Lampson, "An Overview of the CAL Timesharing System", Computer Ctr., Univ. of Calif., Berkeley, CA, (1969).

33. D.D.Clark, R.M.Graham, J.H.Saltzer and M.D.Schroeder, "Classroom Information and Computing Service", Project MAC Tech. Report TR80 (1971).

34. J.E.Sammet, "Brief Survey of Languages Used for Systems Implementation", *SIGPLAN Notices* **6**, No. 9, (1971).

35. *The Multiplexed Information and Computing Service: Programmers' Manual,* M.I.T. Proj. MAC, Rev. 10 (1972). Available from the M.I.T. Information Proc. Center.

HIGHLIGHTS of the MULTICS SYSTEM

FROM THE EDITOR

This is the information that a new Multics programmer receives when he becomes a user at M.I.T. (or elsewhere). This should be of interest in view of impending upgrading of the system.

The next issue of the Honeywell Computer Journal will carry a special microfiche devoted to a historical potpourri of the major papers on Multics.

Introduction

Multics (from **Multi**plexed **I**nformation and **C**omputing **S**ervice) is the name of a new, general-purpose computer system developed by the Computer System Research group at M.I.T. Project MAC, in cooperation with Honeywell Information Systems (formerly the General Electric Company Computer Department) and the Bell Telephone Laboratories. This system is designed to be a "computer utility", extending the basic concepts and philosophy of the Compatible Timesharing System (CTSS, operating now on the IBM 7094 computer) in many directions. Multics is implemented initially on the Honeywell 645 computer system, an enhanced relative of the Honeywell 635 computer.

One of the over-all design goals of Multics is to create a computing system which is capable of meeting almost all of the present and near future requirements of a large computer utility. Such systems must run continuously and reliably 7 days a week, 24 hours a day, in a way similar to telephone or power systems, and must be capable of meeting wide service demands: from multiple man-machine interaction to the sequential processing of absentee user jobs; from the use of the system with dedicated languages and subsystems to the programming of the system itself; and from centralized bulk card, tape, and printer facilities to remotely located terminals. Such information processing and communication systems are believed to be essential for the future growth of computer use in business, in industry, in government and in scientific laboratories, as well as stimulating applications which would otherwise be untried.

Because the system must ultimately be comprehensive and able to adapt to unknown future requirements, its framework must be general, and capable of evolving with time. As brought out in the sequel, this need for an evolutionary framework influences and contributes to much of the system design and is a major reason why most of the programming of the system has been done in a subset of the PL/I language. Because the PL/I language is largely machine-independent (e.g., data descriptions refer to logical items, not physical words), the system should also be. Specifically, it is hoped that future hardware improvements will not make system and user programs obsolete and that implementation of the entire system on other suitable computers will require only a moderate amount of additional programming.

As computers have matured during the last two decades from curiosities to calculating machines to information processors, access to them by users has not improved, and, in the case of most large machines, has retrogressed. Princi pally for economic reasons, batch processing of computer jobs has been developed and is currently practiced by most large computer installations, and the concomitant isolation of the user from elementary cause-and-effect relationships has been either reluctantly endured or rationalized. For several years a solution has been proposed to the access problem. This solution, usually called timesharing, is basically the rapid time-division multiplexing of a central processor unit among the jobs of several users, each online at a typewriter-like terminal. The rapid switching of the processor unit among user programs is, of course, nothing but a particular form of multiprogramming.

The impetus for timesharing first arose from professional programmers because of their constant frustration in debugging programs at batch processing installations. Thus, the original goal was to timeshare computers to allow simultaneous access by several persons while giving to each of them the illusion of having the whole machine at his disposal. This goal led to the development of the Compatible Timesharing System (CTSS) at M.I.T. Project MAC. However, at Project MAC it has turned out that simultaneous access to the machine, while obviously necessary to the objective, has not been the major ensuing benefit. Rather, it is the availability at one's fingertips of facilities for editing, compiling, debugging, and running programs in one continuous interactive session that has had the greatest effect on programming. Professional programmers are encouraged to be more imaginative in their work and to investigate new programming techniques and new problem approaches because of the much smaller penalty for failure. But, the most significant effect that CTSS has had on the M.I.T. community is seen in the achievements of persons for whom computers are tools for other objectives. The availability of CTSS not only has changed the way problems are attacked, but has caused important research to be undertaken that otherwise would not have been done. As a consequence, the objective of the current and future development of timesharing extends beyond the improvement of computational facilities with respect to traditional computer applications. Rather, it is the online use of computers for new purposes and in new fields which provides the challenge and the motivation to the system designer. In other words, the major goal is to provide suitable tools for what is currently being called machine-aided cognition.

More specifically, the importance of a multiple-access system operated as a computer utility is that it allows a vast enlargement of the scope of computer-based activities, which can, in turn, stimulate a corresponding enrichment of many areas of our society. Over six years of experience indicates that continuous operation in a utility-like manner, with flexible remote access, encourages users to view the system as a thinking tool in their daily intellectual work. Mechanistically, the qualitative change from the past results from the drastic improvement in access time and convenience. Objectively, the change lies in the user's ability to control and affect interactively the course of a process whether it involves numerical computation or manipulation of symbols. Thus, parameter studies are more intelligently guided; new problem-oriented languages and subsystems are developed to exploit the interactive capability; many complex analytical problems, as in magnetohydrodynamics, which have been too cumbersome to be tackled in the past, are now being successfully pursued; even more, new, imaginative approaches to basic research have been developed, as in the decoding of protein structures. These are examples taken from an academic environment; the effect of multiple-access systems on business and industrial organizations can be equally dramatic. It is with such new applications in mind that the Multics system has been developed. Not that the traditional uses of computers are being disregarded: rather, these traditional needs are viewed as a subset of the broader, more demanding, new requirements.

To meet the above objectives, issues such as response time, convenience of manipulating data and programs, ease of controlling processes during execution, and, above all, protection of private information and isolation of independent processes, become of critical importance. These issues demand departures from traditional computer systems. While these departures are deemed to be desirable with respect to traditional computer applications, they are essential for rapid man-machine interaction.

System Requirements

In the early days of computer design, there was the concept of a single program on which a single processor computed for long periods of time with almost no interaction with the outside world. Today such a view is considered incomplete. The effective boundaries of an information processing system extend beyond the processor, beyond the card reader and printer, and even beyond the typing of input and the printing of output. In fact, they encompass the goals of many people. To better understand the effect of this broadened design scope, it is helpful to examine several phenomena characteristic of large, service-oriented computer installations.

First, there are incentives for any organization to have the biggest possible computer system that it can afford. It is usually only on the biggest computers that there are elaborate programming systems, compilers, and features which make a computer "powerful". This results partly because it is more difficult to prepare system programs for smaller computers when limited by speed or memory size, and partly because large systems involve more persons and, hence, permit more attention to be given to system programs. Moreover, by combining resources in a single computer system rather than in several, bulk economies and therefore lower computing costs can be achieved. Finally, as a practical matter, considerations of floor space, management efficiency, and operating personnel provide a strong incentive for centralizing computer facilities in a single large installation.

Second, the capacity of a contemporary computer installation, regardless of the sector of applications it serves, must be capable of growing to meet continuously increasing demand. A doubling of demand every two years is not uncommon. Multiple-access computers promise to accelerate this growth further since they allow a man-machine interaction rate which is faster by at least two orders of magnitude than other types of computing systems. Present indications are that multiple-access systems for only a few hundred users can generate a demand for computation exceeding the capacity of the fastest existing single processor system. Since the speed of light, the physical sizes of computer components, and the speeds of memories are intrinsic limitations on the speed of any single processor, it is clear that systems with multiple processors and multiple memory units are needed to provide greater capacity. This is not to say that fast processor units are undesirable, but that extreme system complexity to enhance this single parameter among many appears neither wise nor economic.

Third, computers are no longer a luxury used when and if available, but are primary working tools in business, government, and research laboratories. The more reliable computers become, the more their availability is depended upon. A system structure including pools of functionally identical units (processors, memory modules, input/output controllers, etc.) can provide continuous service without significant interruption for equipment maintenance, as well as provide growth capability through the addition of appropriate units.

Fourth, user programs, especially in a timesharing system, interact frequently with secondary storage devices and terminals. This communication traffic produces a need for multiprogramming to avoid wasting main processor time while an input/output request is being completed. It is important to note that an individual user is ordinarily not is a position to do an adequate job of multiprogramming since his program lacks proper balance, and he probably lacks the necessary dynamic information, ingenuity, or patience.

Finally, as noted earlier, the value of a timesharing system lies not only in providing, in effect, a private computer to a number of people simultaneously, but, above all, in the services that the system places at the fingertips of the users. Moreover, the effectiveness of a system increases as user-developed facilities are shared by other users. This increased effectiveness because of sharing is due not only to the reduced demands for core and secondary memory, but also to the cross-fertilization of user ideas. Thus, a major goal of the present effort is to provide multiple access to a growing and potentially vast structure of shared data and shared program procedures. In fact, the achievement of multiple access to the computer processors should be viewed as but a necessary subgoal of this broader objective. Thus, the primary and secondary memories where programs reside play a central role in the hardware organization, and the presence of independent communication paths between memories, processors, and terminals is of critical importance.

From the above it can be seen that the system requirements of a computer installation are not for a single program on a single computer, but rather for a large system of many components serving a community of users. Moreover, each user of the system asynchronously initiates jobs of arbitrary and indeterminate duration which subdivide into sequences of processor and input/output tasks. It is out of this seemingly chaotic, random environment that one arrives at a utility-like view of a computing system. For instead of chaos, one can average over the different user requests to achieve high utilization of all resources. The task of multiprogramming required to do this need only be organized once in a central supervisor program. Each user thus enjoys the benefit of efficiency without having to average the demands of his own particular program.

With the above view of computer use, where tasks start and stop every few milliseconds, and where the memory requirements of tasks grow and shrink, it is apparent that one of the major jobs of the supervisor program (i.e., monitor, executive, etc.) is the allocation and scheduling of computer resources. The general strategy is clear. Each user's job is subdivided into tasks, usually as the job proceeds, each of which is placed in an appropriate queue (i.e., for a processor or an input/output controller). Processors or input/output controllers are, in turn, assigned new tasks as they either complete or are removed from old tasks. All processors are treated equivalently in an anonymous pool and are assigned to tasks as needed. In particular, the supervisor does not have a special processor. Further, processors can be added or deleted without significant change in either the user or system programs. Similarly, input/output controllers are directed from queues independently of any particular processor. Again, as with the processors, one can add or delete input/output capacity according to system load without significant reprogramming required.

The Multics System

The over-all design goal of the Multics system is to create a computing system which is capable of comprehensively meeting almost all of the present and near future requirements of a large computer service installation. It is not expected that the initial system, although useful, will reach the objective; rather, the system will evolve with time in a general framework which permits continual growth to meet unknown future requirements. The use of the PL/I language will allow major system software changes to be developed on a schedule separate from that of hardware changes. Since most organizations can no longer afford to overlap old and new equipment during changes, and since software development is at best difficult to schedule, this relative machine-independence should be a major asset.

It is expected that the Multics system will be published and will therefore be available for implementation on any equipment with suitable characteristics. Such publication is desirable for two reasons: first, the system should withstand public scrutiny and criticism; second, in an age of increasing complexity, there is an obligation to present and future system designers to make the inner operating system as lucid as possible so as to reveal the basic system issues.

An ability to share data contained within the framework of a general-purpose timesharing system is a unique feature of Multics, and is directly applicable to administrative problems, research requiring a multi-user accessible database, and general application of the computer to very complicated research problems. The attention paid to mechanisms to provide and control privacy is of direct interest for several of the same applications as well as, for example, medical data. Multics can thus be a valuable tool which provides opportunities for important new research in these areas.

The Hardware System

The Honeywell 645 computer system is a large-scale information processing system with most of the features currently found in such systems. If one attempted to classify systems, it would fall in the same general category of size as the Honeywell 635, the Univac 1108, and the IBM Systems 360/65 and 67.

The configuration at M.I.T., shown in Figure 1, currently contains 384 K (K = 1024) 36-bit words of core memory (1 microsecond access to 36 bits or 1.3 microseconds access to 72 bits), two central processors (330,000 instructions per second when running Multics), a high-performance paging drum (it moves 1024 words in 2 milliseconds, 16 milliseconds average latency with a queue-driven channel controller), 78 million words of disk storage, and a Generalized I/O Controller which handles magnetic tapes, card equipment, and high-speed full ASCII printers, as well as all telecommunication channels. The central processor is built on the Honeywell 635 instruction set with augmentation to permit control of paging and segmentation hardware.

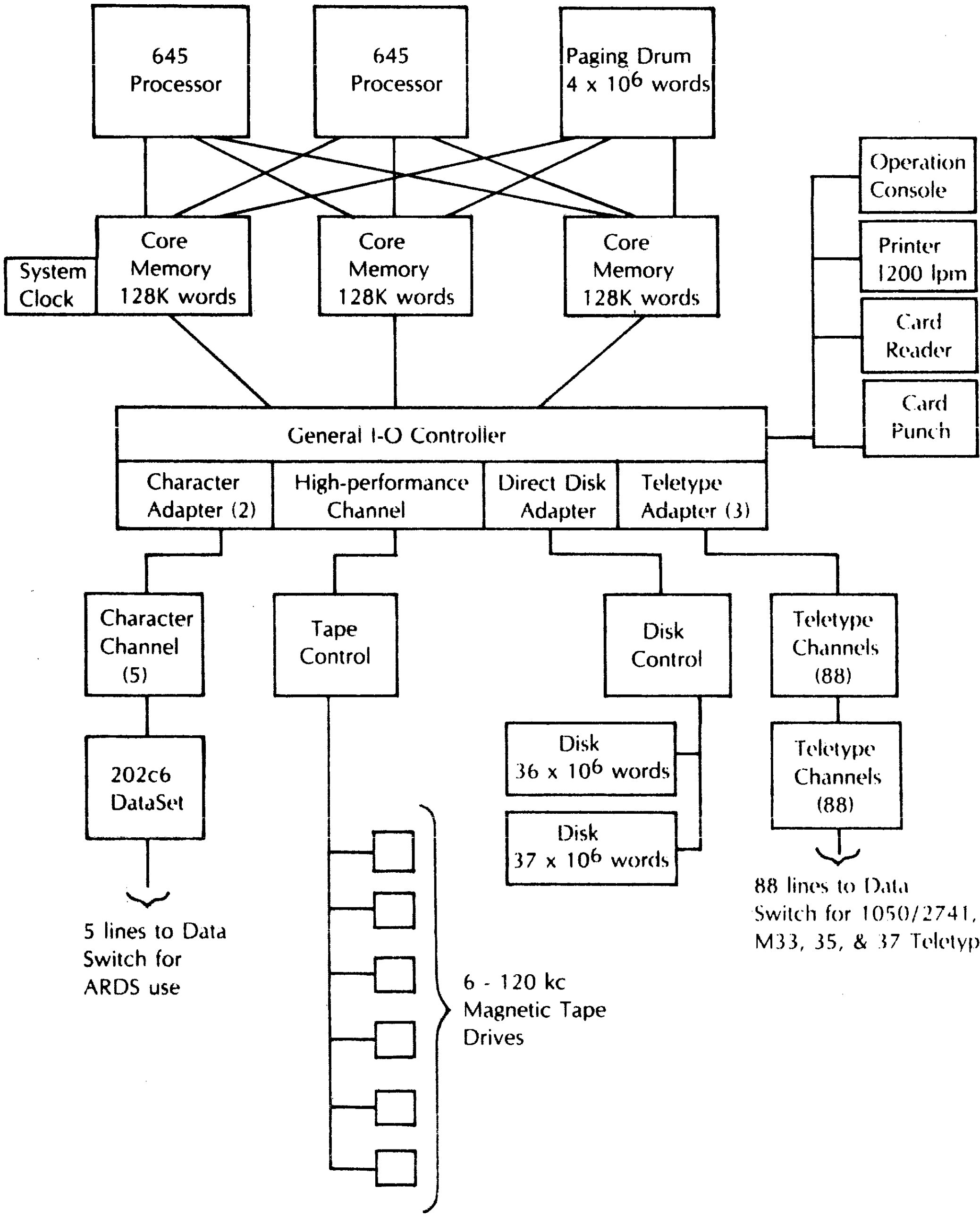


Figure 1.
Honeywell 645 Hardware Configuration at M.I.T.

Overview of Multics Capabilities

Multics offers a number of capabilities which go well beyond those provided by many other systems. Those which are most significant from the user's point of view are described here. Perhaps the most interesting aspect of all is that a single system encompasses all of these capabilities simultaneously.

- The ability to be a small user of Multics.

  An underlying consideration throughout the Multics design has been that the simple user should not pay a noticeable extra price for a system which also accommodates the sophisticated user. For example, a student can be handed a limited set of tools, can do limited work (perhaps debugging and running small FORTRAN programs), and expect to receive a bill for resource usage which is equivalent to the limited work done. If all users are small, then of course the number of users can be increased in proportion to their smallness. As an administrative aid, facilities are provided so that one can restrict any particular user to a specific set of tools and thereby limit his ability to use up resources.

- The ability to control sharing of information.

  There are a variety of applications for a computer system which involve building up a base of information which is to be shared among several individuals. Multics provides facilities in two directions.

  Sharing:

  Links to other users' programs and data.

  Ability to move one's base of operation into another user's directory (with his permission).

  Direct access with uniform conventions to any information stored in the system.

  Ability for two or more users to share a single copy of a program or data in core memory.

  Control:

  Ability to specify precisely to whom, and with what access mode (e.g., read, write, and execute permissions are separate and per-user) a piece of data or the entire contents of a subdirectory are available.

  Ability to revoke access at any time.

  Ability, using the Multics protection ring structure, to force access to a database to be only via a program supplied by the database owner. This facility may be used to allow access to aggregate information, such as averages or counts, or specified data entries, without simultaneously giving access to the entire file of raw data, which may be confidential. There are a large number of potential administrative applications of this feature, and as far as is known, Multics is the only general-purpose system which provides it.

- The virtual memory approach

  In the opposite direction of the little user is the person with a difficult research problem requiring a very large addressable memory. The Multics storage system, with the aid of a high-performance paging drum, provides this facility in what is often called a virtual memory of an extent limited only by the total of secondary storage devices (drums, disks, etc.) attached to the system. An interesting property of the Multics implementation is that a procedure may be written to operate in a very large virtual memory, but core resources are used only for those parts of the virtual memory actually touched by the program on that execution, and disk and drum resources are used only for those parts of the memory which actually contain data. Another very useful property from a programmer's point of view is that information stored in the storage system is directly accessible to his program by a virtual memory address. This property eliminates the need for explicitly programmed overlays, chain links, or core loads, and also reduces the number of explicitly programmed input and output operations. The Multics storage system takes on the responsibility for safekeeping of all information placed there by the user. It therefore automatically maintains tape copies of all information which has remained in the system for more than an hour. These tapes can be used to reload any user information lost or damaged as a result of hardware or software failures, and may also be used to retrieve individual items damaged by a user's own blunder.

  Each user has an administratively set quota of space which limits the amount of storage he can use, although he may purchase as large an amount of space as he would like. Additional disk storage can be added to the 645 computer in large quantities if necessary.

- The option of dynamic linking.

  In constructing a program or system of programs, it is frequently convenient to begin testing certain features of one program before having written another program which is needed for some cases. Dynamic linking allows the execution of the first program to begin, and a search for the second program is undertaken only if and when it is actually called by the first one. This feature also allows a user to freely include in his program a conditional call out to a large and sophisticated error diagnostic program, secure in the knowledge that in all those executions of his program which do not encounter the error, he will not pay the cost of locating, linking, and mapping the error diagnosis package into his virtual memory. It also allows a user who is borrowing a program to provide a substitute for any subroutine called by that program when he uses it, since he has control over where the system looks to find missing subroutines. In those cases where subroutine A calls subroutine B every time, there is, of course, no need to use dynamic linking (and the implied library search), and so facilities are therefore provided to bind A and B together prior to execution.

- Configuration Flexibility.

  An important aspect of the Multics design is that it is actually difficult for a user to write a program which will stop working correctly if the hardware configuration is changed. In response to changing system-wide needs, the amount of core memory, the number of central processors, the amount and nature of secondary storage (disks, drums, etc.), and the type of interactive typewriter terminals may change with time over a range of 2 or 3 to 1, but users do not normally need to change their programs to keep up with the hardware. The system itself adapts to changes in the number of processor or memory boxes dynamically, that is, while users are logged in. Most other configuration changes (e.g., the addition of disk storage units) require that the system be reinitialized, an operation which takes a few minutes.

- The human interface.

  Experience has proven that ease of use of a timesharing system is considerably more sensitive to human engineering than is a batch processing system. The Multics command language has been designed with this in mind. Features such as universal use of a character set with both upper and lower case letters in it, and allowing names of files to be 32 characters long, are examples of the little things which allow the nonspecialist to feel that he does not have to discover a secret in order to be an effective user of the system. In a similar vein, a hierarchial file system provides a very useful organization and bookkeeping aid, so that a user need keep immediately at hand only those things he is working with at the moment. Such a facility is of great assistance when attacking complicated or intricately structured problems.

Languages

Multics provides two primary user languages: FORTRAN IV and PL/I. The FORTRAN compiler is fairly standard. It is supported by the usual library of math routines and formatted input/output facilities. Its primary use is for translation of already written programs which have been imported from other computer systems.

The Multics PL/I compiler is quite interesting because it offers a very full selection of language facilities, over 300 helpful error diagnostics, and the ability to get at the advanced features of Multics, all at a reasonable cost. On a "seconds to translate a source language page" basis, the PL/I compiler currently takes about twice as long as does the FORTRAN compiler; on the other hand, a page of PL/I program can express considerably more than a page of FORTRAN program. For these reasons, as well as the anticipated wide availability of PL/I on other computer systems, it is the recommended language for subsystem implementers and general research users needing an expressive language.

Other languages are:

BASIC A translator and editor subsystem for the BASIC language, developed at Dartmouth College. A limited Multics service is available which restricts the user to just this subsystem, if desired. The BASIC subsystem is also available to regular Multics users.

APL A powerful and popular interpretive language developed by Kenneth Iverson.

LISP List Processing language, version 1.5. Both an interpreter and a compiler for this popular language for "artificial intelligence" problems are available. An interesting feature of the Multics implementation is the very large structure space provided by the virtual memory.

ALM A machine language assembler for the Honeywell 645 computer. (It is not recommended for general use; it is slow and the language is very difficult.)

QEDX A programmable editor which qualifies as a minor interpretive language.

All of the above languages translate a source program which has been previously placed in the storage system. Input and editing of source text is done with one of the available text editors, EDM or QEDX. Although interactive, line-by-line syntax checking languages are easily implemented in the Multics environment, none are yet available.

A source language debugging system, named DEBUG, provides the ability to inspect variables and set breakpoints in terms of the PL/I program being debugged. It also has a variety of features to allow inspection of all aspects of the Multics execution environment.

Reliability and Performance

An initial version of Multics began operating on a scheduled daily basis for system programming use in 1968 September. It has been scheduled to run on a 24-hour-a-day basis since 1969 May 1. Since that time, almost three years of operational experience has been obtained. During this time, reliability, functional capabilities, and performance have been brought to the point that, as of 1972 January 1, a two processor system serves 55 simultaneous users, with good interactive response.

The full configuration of Figure 1 is used regularly, and should ultimately handle about 90 average PL/I programmers. Both smaller and larger users are also runnable on the system, in increased and reduced numbers, respectively.

As an offering of the M.I.T. Information Processing Center, Multics has attracted a community of about 600 registered users, and an equal number of unregistered student users. These users are organized around approximately 100 projects, thus making Multics the primary source of timesharing services at M.I.T. (As with all I.P.C. computer systems, the use of Multics is charged to its users at rates adjusted to return full hardware and running costs when the system is operating at about two-thirds capacity.)

A MULTICS BIBLIOGRAPHY

Manuals that are Generally Available

1. *Multics Programmers' Manual.* An updateable reference manual giving calling sequences and reference information for all user-callable subroutines and commands. Includes an introduction to the Multics programming environment and a guide to typical ways of using the system. Approximately 800 pages.

2. *The Multics System: An Examination of Its Structure,* by E. I. Organick. A hard cover book describing in some detail how Multics works. The description is from the point of view of a programmer developing a large program or subsystem, who wishes to gain the extra insight to help him intelligently choose among available alternatives of his implementation. M.I.T. Press, Cambridge, MA, and London, England. 392 pp. (1972).

3. *A User's Guide to the Multics FORTRAN Implementation,* by R.A. Freiburghouse. A document which provides the prospective Multics FORTRAN user with sufficient information to enable him to create and execute FORTRAN programs on Multics. It contains a complete definition of the Multics FORTRAN language as well as a description of the FORTRAN command and error messages. It also describes how to communicate with non-FORTRAN programs, and discusses some of the fundamental characteristics of Multics which affect the FORTRAN user. 68 pages.

4. *Multics PL/I Language Specification.* A reference manual which specifies precisely the subset of the PL/I language used on Multics. 174 pages.

5. *User's Guide to the Multics PL/I Implementation,* by R.A.Freiburghouse, et al. Provides detailed information about how the PL/I language is embedded in the Multics programming environment. 53 pages.

6. *Graphic Users' Supplement to the Multics Programmers' Manual.* In the same format as the Multics Programmers' Manual, this supplement gathers in one place descriptions of the Multics Graphics System and the commands and subroutines needed to use it. Approximately 55 pages, illustrated.

Manuals that may be examined in the Project MAC or I. P. C. Document Rooms

1. *Multics System Programmers' Manual.* In principle, a complete reference manual describing how the system works inside. In fact, this document contains many sections which are inconsistent, inaccurate, or obsolete; it is in need of much upgrading. However, its overview sections are generally accurate and valuable if insight into the internal organization is desired. Approximately 3,500 pages.

2. *System Programmers' Supplement to the Multics Programmers' Manual.* This updateable reference manual, in the same format as the Multics Programmers' Manual, provides calling sequences of every system module. Approximately 850 pages.

3. *EPLBSA Programmer's Reference Handbook,* by D.J.Riesenberg. A manual describing the assembly (machine) language for the Honeywell 645 computer. The language has been renamed ALM since the publication of this manual. (Needed only by programmers with some special reason to use 645 machine language.) 85 pages.

4. *Honeywell 645 Processor Manual.* A hardware description, including opcodes, addressing modifiers, etc. Of interest only to dedicated machine language programmers. 175 pages.

5. *Subsystem Writers' Supplement to the Multics Programmers' Manual.* A manual giving calling sequences of internal interfaces of the system which are user-accessible. For the sophisticated subsystem writer who feels that it is important to bypass some standard Multics facility, this manual provides some help in using interfaces one level deeper into the system. This manual is definitely not for the casual user. Approximately 50 pages.

Technical Papers about Multics

1. F.J.Corbató and V.A.Vyssotsky, "Introduction and Overview of the Multics System", *AFIPS Conf. Proc., Fall Joint Comput. Conf.* **27,** Spartan Books, Washington, DC, 1965, pp. 185-196.

2. E.L.Glaser, et al., "System Design of a Computer for Timesharing Application", *AFIPS Conf. Proc., Fall Joint Comput. Conf.* **27,** Spartan Books, Washington, DC, 1965, pp. 197-202.

3. V.A.Vyssotski, et al., "Structure of the Multics Supervisor", *AFIPS Conf. Proc., Fall Joint Comput. Conf.* **27,** Spartan Books, Washington, DC, 1965, 203-212.

4. R.C.Daley and P.G.Newmann, "A General-Purpose File System for Secondary Storage", *AFIPS Conf. Proc., Fall Joint Comput. Conf.* **27,** Spartan Books, Washington, DC, 1965, pp. 213-229.

5. J.F.Ossanna, et al., "Communication and Input/Output Switching in a Multiplex Computing System", *AFIPS Conf. Proc., Fall Joint Comput. Conf.* **27,** Spartan Books, Washington, DC, 1965, pp. 231-241.

6. E.E.David, Jr. and R.M.Fano, "Some Thoughts About the Social Implications of Accessible Computing", *AFIPS Conf. Proc., Fall Joint Comput. Conf.* **27,** Spartan Books, Washington, DC, 1965, pp. 243-247.

7. A.Bensoussan, C.T.Clingen and R.C.Daley, "The Multics Virtual Memory", *ACM Second Symp. on Operating Syst. Principles,* Princeton University, 30-42 (1969).

8. C.T.Clingen, "Program Naming Problems in a Shared Tree-Structured Hierarchy", *NATO Sci. Committee Conf. on Techniques in Software Engg.* **1,** Rome, Italy (1969).

9. R.M.Graham, "Protection in an Information Processing Utility", *Commun. ACM* **11,** No. 5, 306-312 (1968).

10. F.J.Corbató and J.H.Saltzer, "Some Considerations of Supervisor Program Design for Multiplexed Computer Systems", *IFIP Conf. Proc., Invited Papers, 66-72 (1968).*

11. R.C.Daley and J.B.Dennis, "Virtual Memory, Processors, and Sharing in MULTICS", *Commun. ACM* **11,** No. 5, 365-369 (1968).

12. F.J.Corbató, "PL/I as a Tool for System Programming", *Datamation* **15,** No. 6, 68-76 (1969).

13. F.J.Corbató, "A Paging Experiment with the Multics System", *In Honor of P.M. Morse,* M.I.T. Pr., Cambridge, MA, 217-228 (1969).

14. J.H.Saltzer and J.W.Gintell, "The Instrumentation of Multics", *ACM Second Symp. on Operating Syst. Principles,* Princeton University, 167-174 (1969), also in *Commun. ACM* **13,** No. 8, 495-500 (1970).

15. M.J.Spier and E.I.Organick, "The Multics Inter-Process Communication Facility", *ACM Second Symp. on Operating Syst. Principles,* Princeton University, 83-91 (1969).

16. R.A.Freiburghouse, "The Multics PL/I Compiler", *AFIPS Conf. Proc.* **35,** AFIPS Pr., 187-199 (1969).

17. J.M.Grochow, "Real-Time Graphic Display of Timesharing System Operating Characteristics", *AFIPS Conf. Proc., Fall Joint Comput. Conf.* **35,** AFIPS Pr., 379-385 (1969).

18. J.H.Saltzer and J.F.Ossanna, "Remote Terminal Character Stream Processing in Multics", *AFIPS Conf. Proc., Spring Joint Comput. Conf.* **36,** AFIPS Pr., 621-627 (1970).

19. J.F.Ossanna and J.H.Saltzer, "Technical and Human Engineering Problems in Connecting Terminals to a Timesharing System", *AFIPS Conf. Proc., Fall Joint Comput. Conf.* 37, AFIPS Pr., 355-362 (1970).

20. D.D.Clark, R.M.Graham, J.H.Saltzer and M.D.Schroeder, "Classroom Information and Computing Service", M.I.T. Project MAC Technical Report TR-80, (1971).

21. M.D.Schroeder, "Performance of the GE-645 Associative Memory While Multics is in Operation", *ACM Workshop on Syst. Performance Evaluation,* 227-245 (1971).

22. M.D.Schroeder and J.H.Saltzer, "A Hardware Architecture for Implementing Protection Rings", *ACM Third Symp. on Operating Syst. Principles,* Palo Alto, CA, (1971).

23. R.J.Reiertag and E.I.Organick, "The Multics Input/Output System", *ACM Third Symp. on Operating Syst. Principles,* Palo Alto, CA, (1971).

M.I.T. Theses related to Multics

1. J.H.Saltzer, "Traffic Control in a Multiplexed Computer System", Sc.D., MAC-TR-30, (1966).

2. R.Rappaport, "Implementing Multi-Process Primitives in a Multiplexed Computer System", S.M., MAC-TR-55, (1968).

3. H.Deitel, "Absentee Computations in a Multiple-Access Computer System", S.M., MAC-TR-52, (1968).

4. J.Greenbaum, "A Simulator of Multiple Interactive Users to Drive a Timeshared Computer System", S.M., MAC-TR-58, (1968).

5. J.M.Grochow, "The Graphic Display as an Aid in the Monitoring of a Timeshared Computer System", S.M., MAC-TR-54, (1968).

6. R.I.Ancona, "A Compiler for MAD-Based Language on Multics", S.M., (1968).

7. D.Clark, "A Reduction Analysis System for Parsing PL/I", S.M., (1968).

8. M.D.Schroeder, "Classroom Model of an Information and Computing Service", S.M., (1969).

9. C.M.Vogt, "Suspension of Processes' in a Multiprocessing Computer System", S.M., (1970).

10. R.Frankston, "A Limited Service System on Multics", S.B., (1970).

11. R.R.Schell, "Dynamic Reconfiguration in a Modular Computer System", Ph.D., (1971).

APPENDIX D

MULTICS COURSES

This page has intentionally been left blank.

LEVEL 68 CURRICULUM
HONEYWELL MARKETING EDUCATION

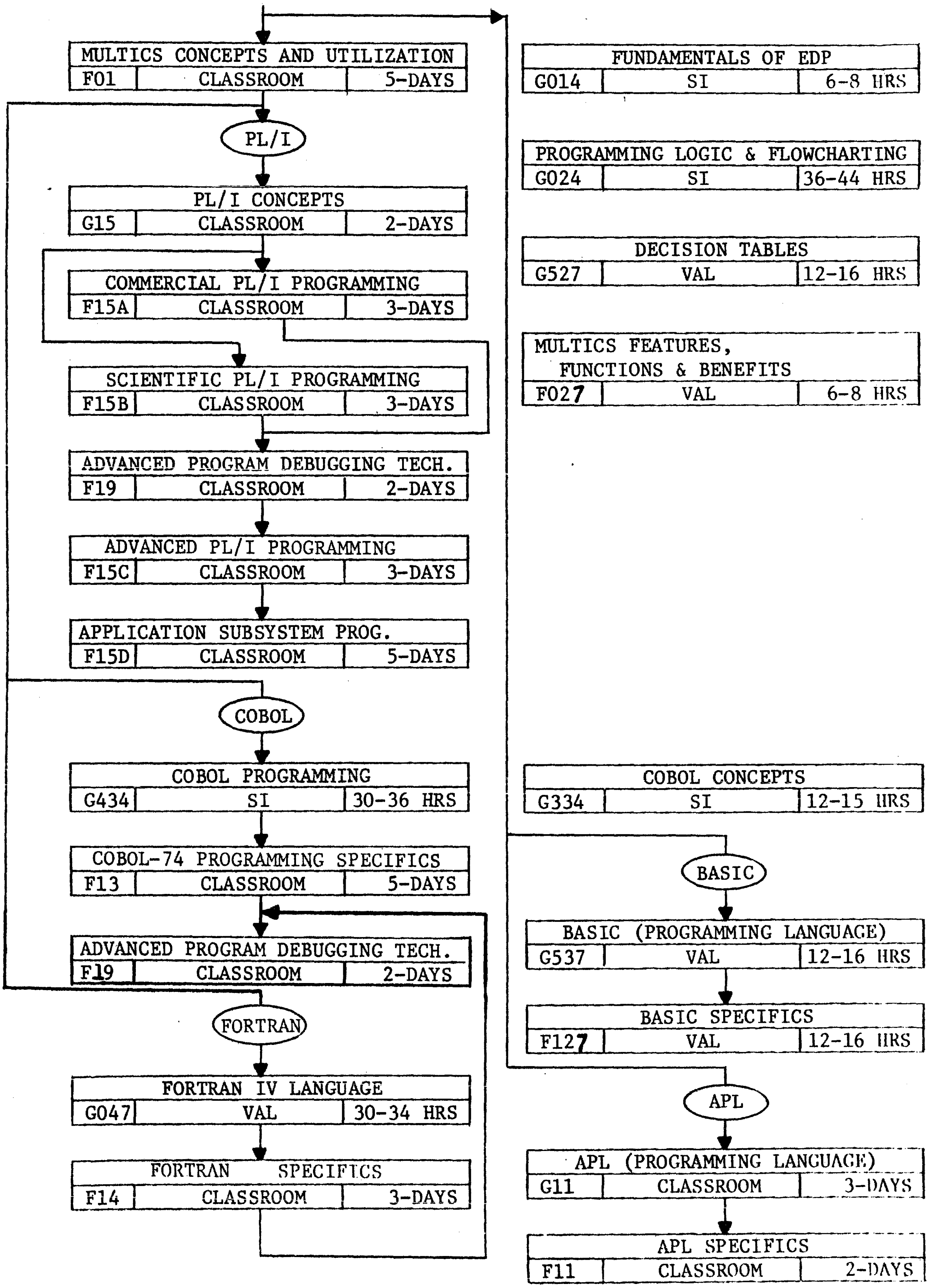
General		
F01	5-Days	Multics Concepts and Utilization
F02	1-Day	Multics Features, Functions, and Benefits
Languages		
G11	3-Days	APL
+F11	2-Days	APL Specifics
+F12	2-Days	BASIC Specifics
F13	5-Days	COBOL-74 Specifics
F14	3-Days	FORTRAN Specifics
F15	5-Days	Multics PL/I Programming
F15C	3-Days	Advanced PL/I Programming
F15D	5-Days	Application Subsystem Programming Techniques
F19	2-Days	Advanced Program Debugging Techniques
Data Base		
F32	3-Days	LINUS Utilization
F31	5-Days	MRDS Utilization
F30	2-Days	MRPG Utilization
Word Processing		
*F41	1-Day	WORDPRO Basic Utilization
*F42	3-Days	WORDPRO Advanced Utilization
F43	5-Days	WORDPRO for Document Administrators
Administration and Operation		
F60	5-Days	System Administration
F617	6-8 Hour VAL	Project Administration
F68	5-Days	Operator Training
Applications		
+F70	5-Days	Graphics Programming
Operating Supervisor		
F80	5-Days	Multics Process Management Analysis
+F81	5-Days	Multics Failure Analysis and Recovery
+F86	5-Days	MCS Concepts and Implementation
Miscellaneous		
+F90	3-Days	GCOS Environment Simulator Utilization

+ Not currently available (December, 78)

* Conducted on-site by special request only

LANGUAGES

COURSE SELECTION GUIDE



RECOMMENDED

MULTICS CONCEPTS AND UTILIZATION	F01	CLASSROOM	5-DAYS

PL/I

PL/I CONCEPTS	G15	CLASSROOM	2-DAYS
COMMERCIAL PL/I PROGRAMMING	F15A	CLASSROOM	3-DAYS
SCIENTIFIC PL/I PROGRAMMING	F15B	CLASSROOM	3-DAYS
ADVANCED PROGRAM DEBUGGING TECH.	F19	CLASSROOM	2-DAYS
ADVANCED PL/I PROGRAMMING	F15C	CLASSROOM	3-DAYS
APPLICATION SUBSYSTEM PROG.	F15D	CLASSROOM	5-DAYS

COBOL

COBOL PROGRAMMING	G434	SI	30-36 HRS
COBOL-74 PROGRAMMING SPECIFICS	F13	CLASSROOM	5-DAYS
ADVANCED PROGRAM DEBUGGING TECH.	F19	CLASSROOM	2-DAYS

FORTRAN

FORTRAN IV LANGUAGE	G047	VAL	30-34 HRS
FORTRAN SPECIFICS	F14	CLASSROOM	3-DAYS

SUPPLEMENTAL/OPTIONAL

FUNDAMENTALS OF EDP	G014	SI	6-8 HRS
PROGRAMMING LOGIC & FLOWCHARTING	G024	SI	36-44 HRS
DECISION TABLES	G527	VAL	12-16 HRS
MULTICS FEATURES, FUNCTIONS & BENEFITS	F027	VAL	6-8 HRS
COBOL CONCEPTS	G334	SI	12-15 HRS

BASIC

BASIC (PROGRAMMING LANGUAGE)	G537	VAL	12-16 HRS
BASIC SPECIFICS	F127	VAL	12-16 HRS

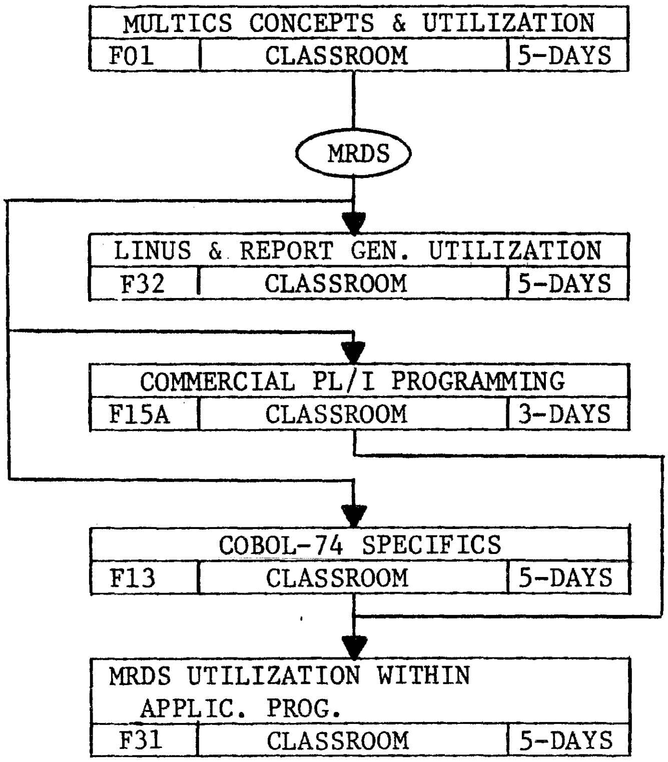
APL

APL (PROGRAMMING LANGUAGE)	G11	CLASSROOM	3-DAYS
APL SPECIFICS	F11	CLASSROOM	2-DAYS

DATA BASE

COURSE SELECTION GUIDE

RECOMMENDED



MULTICS CONCEPTS & UTILIZATION		
F01	CLASSROOM	5-DAYS
LINUS & REPORT GEN. UTILIZATION		
F32	CLASSROOM	5-DAYS
COMMERCIAL PL/I PROGRAMMING		
F15A	CLASSROOM	3-DAYS
COBOL-74 SPECIFICS		
F13	CLASSROOM	5-DAYS
MRDS UTILIZATION WITHIN APPLIC. PROG.		
F31	CLASSROOM	5-DAYS

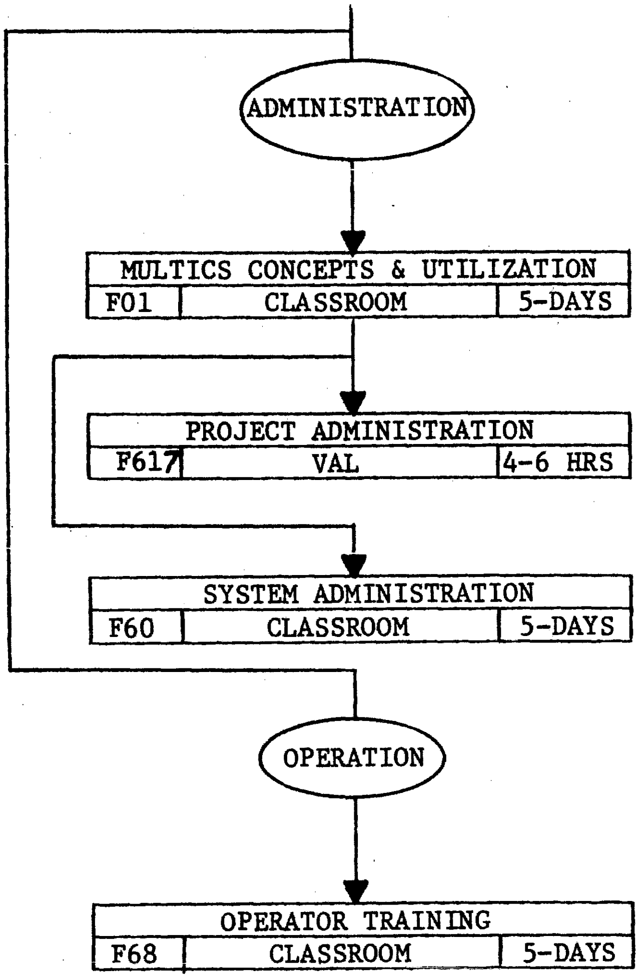
SUPPLEMENTAL/OPTIONAL

FUNDAMENTALS OF EDP		
G014	CLASSROOM	6-8 HRS
MULTICS FEATURES, FUNCTIONS & BENEFITS		
F027	VAL	6-8 HRS
DATA BASE CONCEPTS		
G078	1BR	1-HR

ADMINISTRATION AND OPERATION

COURSE SELECTION GUIDE

RECOMMENDED



MULTICS CONCEPTS & UTILIZATION		
F01	CLASSROOM	5-DAYS

PROJECT ADMINISTRATION		
F617	VAL	4-6 HRS

SYSTEM ADMINISTRATION		
F60	CLASSROOM	5-DAYS

OPERATOR TRAINING		
F68	CLASSROOM	5-DAYS

SUPPLEMENTAL/OPTIONAL

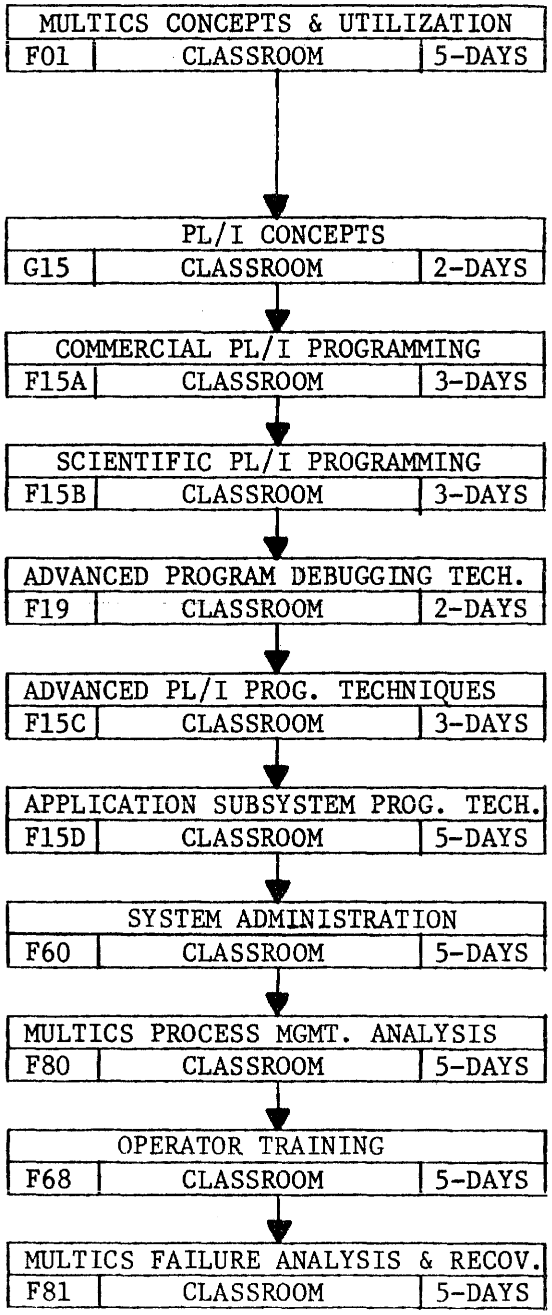
FUNDAMENTALS OF EDP		
G014	SI	6-8 HRS

MULTICS FEATURES, FUNCTIONS & BENEFITS		
F027	VAL	6-8 HRS

OPERATING SUPERVISOR

COURSE SELECTION GUIDE

RECOMMENDED



MULTICS CONCEPTS & UTILIZATION	F01	CLASSROOM	5-DAYS
PL/I CONCEPTS	G15	CLASSROOM	2-DAYS
COMMERCIAL PL/I PROGRAMMING	F15A	CLASSROOM	3-DAYS
SCIENTIFIC PL/I PROGRAMMING	F15B	CLASSROOM	3-DAYS
ADVANCED PROGRAM DEBUGGING TECH.	F19	CLASSROOM	2-DAYS
ADVANCED PL/I PROG. TECHNIQUES	F15C	CLASSROOM	3-DAYS
APPLICATION SUBSYSTEM PROG. TECH.	F15D	CLASSROOM	5-DAYS
SYSTEM ADMINISTRATION	F60	CLASSROOM	5-DAYS
MULTICS PROCESS MGMT. ANALYSIS	F80	CLASSROOM	5-DAYS
OPERATOR TRAINING	F68	CLASSROOM	5-DAYS
MULTICS FAILURE ANALYSIS & RECOV.	F81	CLASSROOM	5-DAYS

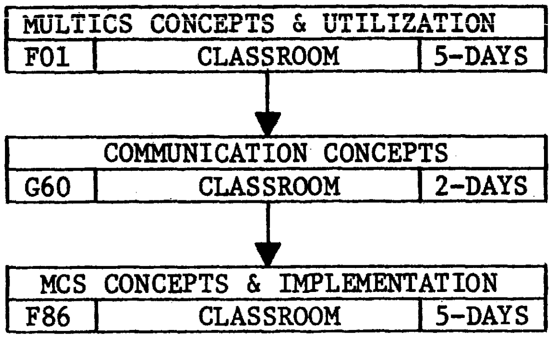
SUPPLEMENTAL/OPTIONAL

FUNDAMENTALS OF EDP	G014	SI	6-8 HRS
PROGRAMMING LOGIC & FLOWCHARTING	G024	SI	36-44 HRS
DECISION TABLES	G527	VAL	12-16 HRS
MULTICS FEATURES, FUNCTIONS & BENEFITS	F027	VAL	6-8 HRS

COMMUNICATIONS

COURSE SELECTION GUIDE

RECOMMENDED



MULTICS CONCEPTS & UTILIZATION

F01 CLASSROOM 5-DAYS

COMMUNICATION CONCEPTS

G60 CLASSROOM 2-DAYS

MCS CONCEPTS & IMPLEMENTATION

F86 CLASSROOM 5-DAYS

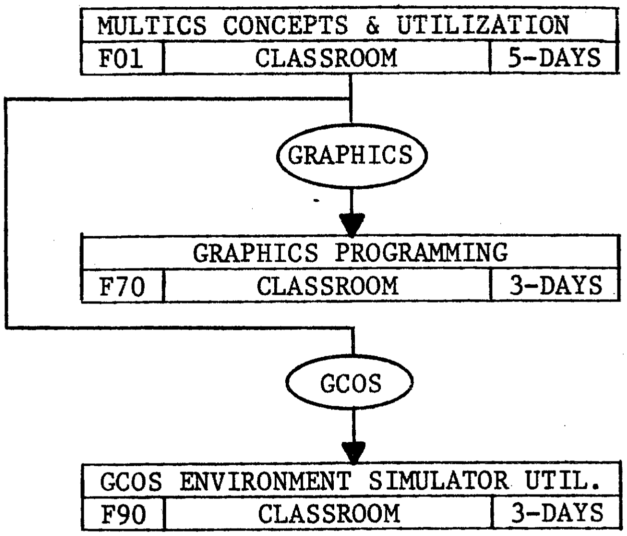
SUPPLEMENTAL/OPTIONAL

FUNDAMENTALS OF EDP	G014	SI	6-8 HRS
COMMUNICATIONS CONCEPTS (BASIC)	G064	SI	6-8 HRS
COMM. SYS. ANALYSIS & DESIGN	G86	CLASSROOM	5-DAYS
THE DISTRIBUTED ENVIRONMENT	G80	CLASSROOM	2-DAYS
MULTICS FEATURES, FUNCTIONS, & BENEFITS	F027	VAL	6-8 HRS

APPLICATIONS AND MISCELLANEOUS

COURSE SELECTION GUIDE

RECOMMENDED



SUPPLEMENTAL/OPTIONAL

MULTICS FEATURES, FUNCTIONS & BENEFITS		
F027	VAL	6-8 HRS

This page has intentionally

been left blank

APPENDIX E

COMMANDS/MANUALS CROSS-REFERENCE

This page has intentionally

been left blank

COMMANDS/MANUALS CROSS REFERENCE

AG92/r2	029and963
AG92/r2	MAP
AR97/aA	TEST (BOS)
AR97/aA	TSTCHN (BOS)
AK92/aB	aa
AG92/r2	ab
AG92/r2	abbrev
AZ03/r0	abbrev_
AZ03/r0	abbrev_$abbrev_
AZ03/r0	abbrev_$expanded_line
AZ03/r0	abbrev_$set_cp_
AG92/r2	abc
AG93/aD	absolute_pathname_
AG92/r2	ac
AG92/r2	accept_messages
AN52/r0	acm
AK92/aB	active_fnc_err_
AS68/r0	add_anon
AG92/r2	add_name
AG92/r2	add_search_rules
AG92/r2	adjust_bit_count
AG93/aD	adjust_bit_count_
AK50/r0	admin_util
AK92/aB	aim_check_
AK92/aB	aim_check_$equal
AK92/aB	aim_check_$greater
AK92/aB	aim_check_$greater_or_equal
AN52/r0	alarm_clock_meters
AK92/aB	alm
AK92/aB	alm_abs
AG92/r2	alv
AG92/r2	am
AG92/r2	an
AG92/r2	and (AF)
AG92/r2	ans
AG92/r2	answer
AG92/r2	apl
AG92/r2	ar
AG92/r2	archive
AK92/aB	archive_sort
AK92/aB	area_info_
AK92/aB	area_status
AK92/aB	as
AK51/r1	as_who
AK92/aB	ascii_to_ebcdic_
AK92/aB	ascii_to_ebcdic_$table
AZ03/r0	ask_
AZ03/r0	ask_$ask_
AZ03/r0	ask_$ask_c
AZ03/r0	ask_$ask_cflo
AZ03/r0	ask_$ask_cint
AZ03/r0	ask_$ask_cline
AZ03/r0	ask_$ask_clr
AZ03/r0	ask_$ask_flo
AZ03/r0	ask_$ask_int
AZ03/r0	ask_$ask_line
AZ03/r0	ask_$ask_n
AZ03/r0	ask_$ask_nflo
AZ03/r0	ask_$ask_nint
AZ03/r0	ask_$ask_nline
AZ03/r0	ask_$ask_prompt
AZ03/r0	ask_$ask_setline
AG92/r2	asr
AK92/aB	assign_
AG92/r2	assign_resource
AG92/r2	attach_lv
AG92/r2	basic
AG92/r2	bd
AS68/r0	bill
AG92/r2	bind
AG92/r2	branches (AF)
AG92/r2	calc
AG92/r2	cancel_abs_request
AG92/r2	cancel_cobol_program
AG92/r2	cancel_daemon_request
AG92/r2	car
AG92/r2	ccp
AG92/r2	cd
AG92/r2	cdr
AG92/r2	cdwd
AG92/r2	ceil (AF)
AG92/r2	cem
AG92/r2	cf
AS68/r0	chaddr
AS68/r0	chalias
AS68/r0	change
AG92/r2	change_default_wdir
AG92/r2	change_error_mode
AZ03/r0	change_kst_attributes
AN52/r0	change_tuning_parameters
AG92/r2	change_wdir
AG93/aD	change_wdir_
AS68/r0	charge
AK50/r0	charge_disk
AS68/r0	chdf_proj
AK50/r0	check_dir
AG92/r2	check_iacl

COMMANDS/MANUALS CROSS-REFERENCE

AG92/r2	check_info_segs
AS68/r0	check_log
AZ03/r0	check_mdcs
AK50/r0	check_mst
AZ03/r0	check_mst
AN53/r0	check_sst
AK92/aB	check_star_name_
AK92/aB	check_star_name_$entry
AK92/aB	check_star_name_$path
AS68/r0	chname
AS68/r0	chpass
AS68/r0	chprog
AG92/r2	cis
AZ03/r0	cka
AK50/r0	ckm
AZ03/r0	ckm
AK50/r0	clean_card_pool
AK50/r0	clear_projfile
AK50/r0	clear_reqfile
AG93/aD	clock_
AG92/r2	close_file
AG92/r2	co
AK50/r0	cob
AZ03/r0	cob
AG92/r2	cobol
AG93/aD	com_err_
AG93/aD	com_err_
AG93/aD	com_err_$suppress_name
AG93/aD	command_query_
AK50/r0	comp_dir_info
AZ03/r0	comp_dir_info
AG92/r2	compare
AG92/r2	compare_ascii
AK50/r0	compare_mst
AZ03/r0	compare_mst
AK50/r0	compare_object
AZ03/r0	compare_object
AK50/r0	compute_bill
AK92/aB	condition_interpreter_
AG92/r2	console_output
AK92/aB	continue_to_signal_
AK92/aB	convert_aim_attributes_
AG93/aD	convert_authorization_
AG93/aD	convert_authorization_$decode
AG93/aD	convert_authorization_$encode
AG93/aD	convert_authorization_$from_string
AG93/aD	convert_authorization_$minimum
AG93/aD	convert_authorization_$to_string
AG93/aD	convert_authorization_$to_string_short
AG93/aD	convert_date_to_binary_
AG93/aD	convert_date_to_binary_$relative
AK92/aB	convert_dial_message_
AK92/aB	convert_status_code_
AG92/r2	copy
AG92/r2	copy_acl
AK50/r0	copy_as_meters
AG92/r2	copy_cards
AN53/r0	copy_dump
AN53/r0	copy_dump$set_fdump num
AN53/r0	copy_dump$sfdn
AN53/r0	copy_dump_seg_
AG92/r2	copy_file
AG92/r2	copy_iacl_dir
AG92/r2	copy_iacl_seg
AZ03/r0	copy_mst
AK92/aB	copy_names
AN53/r0	copy_out
AK50/r0	copy_pnt
AN53/r0	copy_salvager_output
AG92/r2	cp
AG92/r2	cpa
AG92/r2	cpf
AZ03/r0	cpm
AG92/r2	cpt
AG93/aD	cpu_time_and_paging_
AG92/r2	cr
AG92/r2	create
AK92/aB	create_area
AK50/r0	create_cmf
AK50/r0	create_daemon_queues
AG92/r2	create_dir
AZ03/r0	create_ips_mask_
AS68/r0	credit
AK50/r0	cref
AZ03/r0	cref
AK50/r0	cross_reference
AZ03/r0	cross_reference
AN52/r0	ctp
AS68/r0	cu
AG93/aD	cu_

COMMANDS/MANUALS CROSS-REFERENCE

AK92/aB	cu_
AK92/aB	cu_$af_arg_count
AK92/aB	cu_$af_arg_ptr
AK92/aB	cu_$af_return_arg
AG93/aD	cu_$arg_count
AK92/aB	cu_$arg_list_ptr
AG93/aD	cu_$arg_ptr
AK92/aB	cu_$arg_ptr_rel
AK92/aB	cu_$cl
AG93/aD	cu_$cp
AK92/aB	cu_$decode_entry_value
AK92/aB	cu_$generate_call
AK92/aB	cu_$get_cl_intermediary
AK92/aB	cu_$get_command_processor
AK92/aB	cu_$get_ready_mode
AK92/aB	cu_$get_ready_procedure
AK92/aB	cu_$level_get
AK92/aB	cu_$level_set
AK92/aB	cu_$ready_proc
AK92/aB	cu_$set_cl_intermediary
AK92/aB	cu_$set_command_processor
AK92/aB	cu_$set_ready_mode
AK92/aB	cu_$set_ready_procedure
AK92/aB	cu_$stack_frame_ptr
AK92/aB	cu_$stack_frame_size
AG92/r2	cumulative_page_trace
AK92/aB	cv_bin_
AK92/aB	cv_bin_$dec
AK92/aB	cv_bin_$oct
AK50/r0	cv_cmf
AK92/aB	cv_entry_
AK92/aB	cv_hex_
AK92/aB	cv_oct_
AK92/aB	cv_oct_check
AK51/r1	cv_pmf
AK92/aB	cv_ptr_
AG92/r2	cwd
AG92/r2	d
AG92/r2	da
AK50/r0	daily_log_process
AK50/r0	daily_summary
AK50/r0	daily_syserr_process
AR97/aA	daily_syserr_process
AG92/r2	date (AF)
AZ03/r0	date_deleter
AG92/r2	date_time (AF)
AG93/aD	date_time_
AG93/aD	date_time_$fstime
AZ03/r0	datebin_
AZ03/r0	datebin_$clockathr
AZ03/r0	datebin_$datebin
AZ03/r0	datebin_$datofirst
AZ03/r0	datebin_$dayr_clk
AZ03/r0	datebin_$dayr_mc
AZ03/r0	datebin_$following_midnight
AZ03/r0	datebin_$last_midnight
AZ03/r0	datebin_$next_shift_change
AZ03/r0	datebin_$preceding_midnight
AZ03/r0	datebin_$revert
AZ03/r0	datebin_$revertabs
AZ03/r0	datebin_$shift
AZ03/r0	datebin_$this_midnight
AZ03/r0	datebin_$time
AZ03/r0	datebin_$wkday
AS68/r0	day
AG92/r2	day (AF)
AG92/r2	day_name (AF)
AG92/r2	db
AK92/aB	dcn
AG92/r2	dcr
AG92/r2	dd
AZ03/r0	deactivate_seg
AG92/r2	debug
AG92/r2	decode
AG93/aD	decode_clock_value_
AZ03/r0	decode_definition_
AZ03/r0	decode_definition_$full
AZ03/r0	decode_definition_$init
AK92/aB	decode_descriptor_
AG92/r2	default (AF)
AG92/r2	defer_messages
AK92/aB	define_area_
AS68/r0	delegate
AG92/r2	delete
AG93/aD	delete_
AG93/aD	delete_$path
AG93/aD	delete_$ptr
AG92/r2	delete_acl
AG92/r2	delete_dir
AK92/aB	delete_external_variables
AG92/r2	delete_force
AG92/r2	delete_iacl_dir

COMMANDS/MANUALS CROSS-REFERENCE

AG92/r2	delete_iacl_seg
AG92/r2	delete_message
AG92/r2	delete_name
AK50/r0	delete_proj
AG92/r2	delete_search_rules
AG92/r2	detach_lv
AN52/r0	device_meters
AR97/aA	device_meters
AG92/r2	df
AG92/r2	dial
AK92/aB	dial_manager_
AG92/r2	did
AG92/r2	directories (AF)
AG92/r2	directory (AF)
AG92/r2	dirs (AF)
AG92/r2	dis
AG93/aD	discard_
AN52/r0	disk_queue
AS68/r0	disk_report
AK50/r0	disk_stat_print
AK51/r1	disk_stat_print
AK50/r0	disk_usage_stat
AK50/r0	disklow
AZ03/r0	display_branch
AG92/r2	display_cobol_run_unit
AK92/aB	display_component_name
AZ03/r0	display_kst_entry
AG92/r2	display_pl1io_err
AG92/r2	divide (AF)
AG92/r2	dl
AG92/r2	dlv
AG92/r2	dm
AS68/r0	dmisc
AG92/r2	dn
AG92/r2	do
AG92/r2	dp
AG92/r2	dpe
AS68/r0	dpmf
AG92/r2	dpn
AG92/r2	dprint
AK92/aB	dprint_
AS68/r0	dproj
AG92/r2	dpunch
AN52/r0	dq
AG92/r2	ds
AK50/r0	dsp
AG92/r2	dsr
AK50/r0	dump_cdt
AR97/aA	dump_firmware
AR97/aA	dump_mpc
AN53/r0	dump_pdmap
AG92/r2	dump_segment
AN52/r0	dvm
AG92/r2	e
AG92/r2	ear
AK92/aB	ebcdic_to_ascii_
AK92/aB	ebcdic_to_ascii_$table
AG92/r2	ec
AK50/r0	ed_installation_parms
AK50/r0	ed_mgt
AK50/r0	edit_proj
AS68/r0	edit_proj
AK50/r0	edit_proj$change_all
AG92/r2	edm
AK50/r0	edurf
AG92/r2	encode
AG93/aD	encode_clock_value_
AG92/r2	enm (AF)
AG92/r2	enter
AG92/r2	enter_abs_request
AG92/r2	enterp
AG92/r2	entry (AF)
AG92/r2	ep
AK50/r0	epro
AG92/r2	equal (AF)
AG92/r2	equal_name (AF)
AK50/r0	erf
AK92/aB	error_table_compiler
AR97/aA	et
AK92/aB	etc
AZ03/r0	excerpt_mst
AG92/r2	exec_com
AG92/r2	exists (AF)
AZ03/r0	expand
AG93/aD	expand_path_
AG93/aD	expand_pathname_
AN53/r0	extract
AG92/r2	fa
AG92/r2	fast
AG92/r2	fcs
AG92/r2	file_output
AN52/r0	file_system_meters
AG92/r2	files (AF)
AK92/aB	find_condition_info_
AZ03/r0	find_include_file
AZ03/r0	find_include_file_$get_search_rules
AZ03/r0	find_include_file_$initiate_count

COMMANDS/MANUALS CROSS-REFERENCE

AZ03/r0	find_include_file_$set_search_rules
AG92/r2	fl (AF)
AG92/r2	floor (AF)
AN52/r0	flush
AG92/r2	fo
AN53/r0	format_355_dump_line_
AN53/r0	format_355_dump_line_$line
AG92/r2	format_cobol_source
AG92/r2	format_line (AF)
AG92/r2	fortran
AG92/r2	fortran_abs
AG92/r2	fs_chname
AN52/r0	fsm
AG92/r2	ft
AG92/r2	gc
AG92/r2	gcl
AG92/r2	gcos
AG92/r2	general_ready
AK50/r0	generate_mst
AZ03/r0	generate_mst
AN53/r0	get_ast_name_
AG93/aD	get_authorization_
AZ03/r0	get_bound_seg_info_
AG92/r2	get_com_line
AK92/aB	get_default_wdir_
AK92/aB	get_definition_
AK51/r1	get_dir_quota
AN53/r0	get_dump_ptrs_
AK92/aB	get_entry_name_
AK92/aB	get_equal_name_
AG93/aD	get_group_id_
AG93/aD	get_group_id_$tag_star
AZ03/r0	get_initial_ring_
AZ03/r0	get_library_segment
AG93/aD	get_line_length_
AG93/aD	get_max_authorization_
AG92/r2	get_pathname (AF)
AG93/aD	get_pdir_
AK92/aB	get_privileges_
AG93/aD	get_process_id_
AG92/r2	get_quota
AK92/aB	get_ring_
AK92/aB	get_system_free_area_
AG92/r2	get_system_search_rules
AG93/aD	get_temp_segments_
AK50/r0	get_uid_with_lastname
AG93/aD	get_wdir_
AZ03/r0	gls
AK50/r0	gm
AZ03/r0	gm
AG92/r2	gpn (AF)
AG92/r2	gq
AG92/r2	gr
AG92/r2	greater (AF)
AG92/r2	gssr
AG92/r2	have_mail (AF)
AG93/aD	hcs_$add_acl_entries
AG93/aD	hcs_$add_dir_acl_entries
AK92/aB	hcs_$add_dir_inacl_entries
AK92/aB	hcs_$add_inacl_entries
AG93/aD	hcs_$append_branch
AG93/aD	hcs_$append_branchx
AG93/aD	hcs_$append_link
AG93/aD	hcs_$chname_file
AG93/aD	hcs_$chname_seg
AG93/aD	hcs_$create_branch_
AK92/aB	hcs_$del_dir_tree
AG93/aD	hcs_$delentry_file
AG93/aD	hcs_$delentry_seg
AG93/aD	hcs_$delete_acl_entries
AG93/aD	hcs_$delete_dir_acl_entries
AK92/aB	hcs_$delete_dir_inacl_entries
AK92/aB	hcs_$delete_inacl_entries
AG93/aD	hcs_$fs_get_mode
AG93/aD	hcs_$fs_get_path_name
AG93/aD	hcs_$fs_get_ref_name
AG93/aD	hcs_$fs_get_seg_ptr
AG93/aD	hcs_$fs_move_file
AG93/aD	hcs_$fs_move_seg
AG93/aD	hcs_$get_access_class
AG93/aD	hcs_$get_access_class_seg
AK92/aB	hcs_$get_author
AK92/aB	hcs_$get_bc_author
AK92/aB	hcs_$get_dir_ring_brackets
AK92/aB	hcs_$get_max_length
AK92/aB	hcs_$get_max_length_seg
AZ03/r0	hcs_$get_page_trace

COMMANDS/MANUALS CROSS-REFERENCE

AK92/aB	hcs_$get_ring_brackets
AK92/aB	hcs_$get_safety_sw
AK92/aB	hcs_$get_safety_sw_seg
AK92/aB	hcs_$get_search_rules
AK92/aB	hcs_$get_system_search_rules
AG93/aD	hcs_$initiate
AG93/aD	hcs_$initiate_count
AK92/aB	hcs_$initiate_search_rules
AG93/aD	hcs_$list_acl
AG93/aD	hcs_$list_dir_acl
AK92/aB	hcs_$list_dir_inacl
AK92/aB	hcs_$list_inacl
AG93/aD	hcs_$make_entry
AG93/aD	hcs_$make_ptr
AG93/aD	hcs_$make_seg
AK92/aB	hcs_$quota_move
AK92/aB	hcs_$quota_read
AG93/aD	hcs_$replace_acl
AG93/aD	hcs_$replace_dir_acl
AK92/aB	hcs_$replace_dir_inacl
AK92/aB	hcs_$replace_inacl
AG93/aD	hcs_$set_bc
AG93/aD	hcs_$set_bc_seg
AK92/aB	hcs_$set_dir_ring_brackets
AK92/aB	hcs_$set_entry_bound
AK92/aB	hcs_$set_entry_bound_seg
AK92/aB	hcs_$set_max_length
AK92/aB	hcs_$set_max_length_seg
AK92/aB	hcs_$set_ring_brackets
AK92/aB	hcs_$set_safety_sw
AK92/aB	hcs_$set_safety_sw_seg
AK92/aB	hcs_$star_
AK92/aB	hcs_$star_dir_list_
AK92/aB	hcs_$star_list_
AG93/aD	hcs_$status_
AG93/aD	hcs_$status_long
AG93/aD	hcs_$status_minf
AG93/aD	hcs_$status_mins
AG93/aD	hcs_$terminate_file
AG93/aD	hcs_$terminate_name
AG93/aD	hcs_$terminate_noname
AG93/aD	hcs_$terminate_seg
AG93/aD	hcs_$truncate_file
AG93/aD	hcs_$truncate_seg
AK92/aB	hcs_$wakeup
AR97/aA	heals_report
AG92/r2	hello
AG92/r2	help
AG92/r2	hmu
AG92/r2	home_dir (AF)
AG92/r2	hour (AF)
AG92/r2	how_many_users
AK50/r0	hp_delete_acl
AK50/r0	hp_set_acl
AK50/r0	hpda
AN71/r1	hphcs_$add_cpu
AN71/r1	hphcs_$add_main
AN71/r1	hphcs_$add_mem
AN71/r1	hphcs_$add_pd_records
AN71/r1	hphcs_$del_cpu
AN71/r1	hphcs_$del_main
AN71/r1	hphcs_$del_mem
AN71/r1	hphcs_$delete_pd_records
AN71/r1	hphcs_$rc_force_unlock
AN71/r1	hphcs_$reconfig_info
AK50/r0	hpsa
AK50/r0	hpset_ring_brackets
AK50/r0	hpsrb
AZ03/r0	hunt
AK50/r0	icref
AZ03/r0	icref
AG92/r2	if
AG92/r2	im
AG92/r2	immediate_messages
AG92/r2	in
AK50/r0	include_cross_reference
AZ03/r0	include_cross_reference
AG92/r2	ind
AG92/r2	indent
AG92/r2	index (AF)
AG92/r2	index_set (AF)
AG92/r2	initiate
AK50/r0	install
AK51/r1	install
AS68/r0	install
AN52/r0	instr_speed
AN52/r0	interrupt_meters
AG92/r2	io
AG92/r2	io_call
AR97/aA	io_error_summary
AG93/aD	ioa_
AG93/aD	ioa_

COMMANDS/MANUALS CROSS-REFERENCE

AG93/aD	ioa_$general_rs
AG93/aD	ioa_$ioa_stream
AG93/aD	ioa_$ioa_stream_nnl
AG93/aD	ioa_$ioa_switch
AG93/aD	ioa_$ioa_switch_nnl
AG93/aD	ioa_$nnl
AG93/aD	ioa_$rs
AG93/aD	ioa_$rsnnl
AG93/aD	ioa_$rsnp
AG93/aD	ioa_$rsnpnnl
AK92/aB	iod_info_
AK92/aB	iod_info_$ driver_access_name
AK92/aB	iod_info_$generic_type
AK50/r0	iod_tables_compiler
AG93/aD	iox_
AK92/aB	iox_
AG93/aD	iox_$attach_iocb
AG93/aD	iox_$attach_ioname
AG93/aD	iox_$close
AG93/aD	iox_$control
AG93/aD	iox_$delete_record
AK92/aB	iox_$destroy_iocb
AG93/aD	iox_$detach_iocb
AK92/aB	iox_$err_no_operation
AK92/aB	iox_$err_not_attached
AK92/aB	iox_$err_not_closed
AK92/aB	iox_$err_not_open
AG93/aD	iox_$find_iocb
AK92/aB	iox_$find_iocb_n
AG93/aD	iox_$get_chars
AG93/aD	iox_$get_line
AK92/aB	iox_$look_iocb
AG93/aD	iox_$modes
AG93/aD	iox_$move_attach
AG93/aD	iox_$open
AG93/aD	iox_$position
AK92/aB	iox_$propagate
AG93/aD	iox_$put_chars
AG93/aD	iox_$read_key
AG93/aD	iox_$read_length
AG93/aD	iox_$read_record
AG93/aD	iox_$rewrite_record
AG93/aD	iox_$seek_key
AG93/aD	iox_$write_record
AK92/aB	ipc_
AK92/aB	ipc_$block
AK92/aB	ipc_$create_ev_chn
AK92/aB	ipc_$cutoff

AK92/aB	ipc_$ dcl_event_call_channel
AK92/aB	ipc_$decl_ev_wait_chn
AK92/aB	ipc_$delete_ev_chn
AK92/aB	ipc_$drain_chn
AK92/aB	ipc_$mask_ev_calls
AK92/aB	ipc_$read_ev_chn
AK92/aB	ipc_$reconnect
AK92/aB	ipc_$set_call_prior
AK92/aB	ipc_$set_wait_prior
AK92/aB	ipc_$unmask_ev_calls
AK50/r0	is_he_user
AK50/r0	is_legal_proj
AS68/r0	ison
AG92/r2	l
AG92/r2	la
AG92/r2	lac
AG92/r2	lar
AG92/r2	last_message (AF)
AG92/r2	last_message_sender (AF)
AG92/r2	last_message_time (AF)
AG92/r2	ldr
AG92/r2	length (AF)
AG92/r2	less (AF)
AZ03/r0	lex_error_
AZ03/r0	lex_string_
AZ03/r0	lex_string_$ init_lex_delims
AZ03/r0	lex_string_$lex
AG92/r2	lid
AG92/r2	line_length
AG92/r2	link
AN52/r0	link_meters
AZ03/r0	link_unsnap_
AG92/r2	links (AF)
AG92/r2	lis
AG92/r2	list
AG92/r2	list_abs_requests
AG92/r2	list_accessible
AG92/r2	list_acl
AG92/r2	list_daemon_requests
AK50/r0	list_delegated_projects
AZ03/r0	list_dir_info
AZ03/r0	list_dir_info_
AK92/aB	list_external_variables
AK50/r0	list_extra_personids
AG92/r2	list_iacl_dir
AG92/r2	list_iacl_seg
AK51/r1	list_mdir

AZ03/r0	list_mst
AG92/r2	list_not_accessible
AG92/r2	list_ref_names
AG92/r2	list_resources
AK92/aB	list_temp_segments
AG92/r2	lk
AN52/r0	lkm
AG92/r2	ll
AG92/r2	lm (AF)
AG92/r2	lms (AF)
AG92/r2	lmt (AF)
AG92/r2	lnac
AK51/r1	load_ctl_status
AG92/r2	login
AG92/r2	logout
AG92/r2	long_date (AF)
AG92/r2	lr
AG92/r2	lrn
AG92/r2	ls
AG92/r2	lv_attached (AF)
AG92/r2	mail
AN69/r0	mailbox_$add_index
AN69/r0	mailbox_$check_salv_bit_index
AN69/r0	mailbox_$chname
AN69/r0	mailbox_$close
AN69/r0	mailbox_$create
AN69/r0	mailbox_$delete
AN69/r0	mailbox_$delete_index
AN69/r0	mailbox_$get_message_count_index
AN69/r0	mailbox_$get_mode_index
AN69/r0	mailbox_$incremental_read_index
AN69/r0	mailbox_$open
AN69/r0	mailbox_$own_incremental_read_index
AN69/r0	mailbox_$own_read_index
AN69/r0	mailbox_$read_delete_index
AN69/r0	mailbox_$read_index
AK92/aB	match_star_name_
AG92/r2	max (AF)
AK92/aB	mban
AK92/aB	mbcr
AK92/aB	mbda
AK92/aB	mbdl
AK92/aB	mbdn
AK92/aB	mbla
AK92/aB	mbrn
AK92/aB	mbsa
AK92/aB	mbsml
AN69/r0	mbx_acl_add
AN69/r0	mbx_acl_delete
AN69/r0	mbx_acl_list
AN69/r0	mbx_acl_replace
AK92/aB	mbx_add_name
AK92/aB	mbx_create
AK92/aB	mbx_delete
AK92/aB	mbx_delete_acl
AK92/aB	mbx_delete_name
AK92/aB	mbx_list_acl
AK92/aB	mbx_rename
AK92/aB	mbx_set_acl
AK92/aB	mbx_set_max_length
AZ03/r0	mcs_version
AG92/r2	memo
AG92/r2	merge
AZ03/r0	merge_mst
AN69/r0	message_segment_$add_file
AN69/r0	message_segment_$add_index
AN69/r0	message_segment_$check_salv_bit_file
AN69/r0	message_segment_$check_salv_bit_index
AN69/r0	message_segment_$chname_file
AN69/r0	message_segment_$close
AN69/r0	message_segment_$create
AN69/r0	message_segment_$delete
AN69/r0	message_segment_$delete_file
AN69/r0	message_segment_$delete_index
AN69/r0	message_segment_$get_message_count_file
AN69/r0	message_segment_$get_message_count_index
AN69/r0	message_segment_$get_mode_file
AN69/r0	message_segment_$get_mode_index
AN69/r0	message_segment_$incremental_file_index

COMMANDS/MANUALS CROSS-REFERENCE

AN69/r0	message_segment_$ incremental_read_index
AN69/r0	message_segment_$ ms_acl_add
AN69/ro	message_segment_$ ms_acl_delete
AN69/r0	message_segment_$ ms_acl_list
AN69/r0	message_segment_$ ms_acl_replace
AN69/r0	message_segment_$ open
AN69/r0	message_segment_$ own_incremental_ read_file
AN69/r0	message_segment_$ own_incremental_ read_index
AN69/r0	message_segment_$ own_read_file
AN69/r0	message_segment_$ own_read_index
AN69/r0	message_segment_$ read_delete_file
AN69/r0	message_segment_$ read_delete_index
AN69/r0	message_segment_$ read_file
AN69/r0	message_segment_$ read_index
AN69/r0	message_segment_$ update_message_file
AN69/r0	message_segment_$ update_message_index
AN52/r0	meter_fim
AN52/r0	meter_gate
AN52/r0	meter_gate_
AN52/r0	meter_signal
AN52/r0	meter_util_
AN52/r0	mg
AG92/r2	min (AF)
AG92/r2	minus (AF)
AG92/r2	minute (AF)
AS68/r0	misc
AG92/r2	ml
AG92/r2	mod (AF)
AG92/r2	month (AF)
AG92/r2	month_name (AF)

AG92/r2	move
AK51/r1	move_dir_quota
AK92/aB	move_names
AG92/r2	move_quota
AG92/r2	mq
AK50/r0	ms_add_name
AN69/r0	ms_add_name
AK50/r0	ms_create
AN69/r0	ms_create
AK50/r0	ms_delete
AN69/r0	ms_delete
AK50/r0	ms_delete_acl
AN69/r0	ms_delete_acl
AK50/r0	ms_delete_name
AN69/r0	ms_delete_name
AK50/r0	ms_list_acl
AN69/r0	ms_list_acl
AK50/r0	ms_rename
AN69/r0	ms_rename
AN69/r0	ms_salv_util_vN_
AN69/r0	ms_salvager_vN_
AK50/r0	ms_set_acl
AN69/r0	ms_set_acl
AK50/r0	ms_set_max_length
AK50/r0	msan
AN69/r0	msan
AK50/r0	mscr
AN69/r0	mscr
AK50/r0	msda
AN69/r0	msda
AK50/r0	msdl
AN69/r0	msdl
AK50/r0	msdn
AN69/r0	msdn
AN69/r0	mseg_
AN69/r0	mseg_access_
AN69/r0	mseg_add_
AN69/r0	mseg_convert_
AN69/r0	mseg_data_
AN69/r0	mseg_index_
AN69/r0	mseg_own_
AN69/r0	mseg_util_
AN69/r0	mseg_util_vN_
AK92/aB	msf_manager_
AK92/aB	msf_manager_$acl_add
AK92/aB	msf_manager_$acl_delete
AK92/aB	msf_manager_$acl_list
AK92/aB	msf_manager_$ acl_replace

COMMANDS/MANUALS CROSS-REFERENCE

AK92/aB	msf_manager_$adjust
AK92/aB	msf_manager_$close
AK92/aB	msf_manager_$get_ptr
AK92/aB	msf_manager_$open
AK50/r0	msla
AN69/r0	msla
AK50/r0	msrn
AN69/r0	msrn
AK50/r0	mssa
AN69/r0	mssa
AK50/r0	mssml
AN69/r0	msu_
AG92/r2	nequal (AF)
AG92/r2	new_fortran
AG92/r2	new_proc
AS68/r0	new_proj
AS68/r0	new_smf
AK50/r0	new_user
AK50/r0	new_user$cg
AK50/r0	new_user$cga
AK50/r0	new_user$change
AK50/r0	new_user$install
AK50/r0	new_user$new_user
AK50/r0	new_user$nu
AK50/r0	new_user$nua
AG92/r2	ngreater (AF)
AG92/r2	nless (AF)
AG92/r2	nondirectories (AF)
AG92/r2	nondirs (AF)
AG92/r2	nonlinks (AF)
AG92/r2	nonsegments (AF)
AG92/r2	nonsegs (AF)
AG92/r2	not (AF)
AZ03/r0	nothing
AZ03/r0	nt
AK92/aB	object_info_
AK92/aB	object_info_$brief
AK92/aB	object_info_$display
AK92/aB	object_info_$long
AN53/r0	od_355
AN53/r0	od_cleanup
AN53/r0	od_print_
AN53/r0	od_print_$op_finish
AN53/r0	od_print_$op_fmt_line
AN53/r0	od_print_$op_init
AN53/r0	od_print_$op_new_page
AN53/r0	od_print_$op_new_seg
AN53/r0	od_stack_
AN53/r0	ol_dump_
AG92/r2	old_fortran
AN53/r0	online_355_dump_
AN53/r0	online_dump
AN53/r0	online_dump_355
AG92/r2	or (AF)
AG92/r2	pa
AN52/r0	page_multilevel_meters
AG92/r2	page_trace
AG92/r2	pan
AZ03/r0	parse_file_
AZ03/r0	parse_file_$parse_file_
AZ03/r0	parse_file_$parse_file_cur_line
AZ03/r0	parse_file_$parse_file_init_name
AZ03/r0	parse_file_$parse_file_init_ptr
AZ03/r0	parse_file_$parse_file_line_no
AZ03/r0	parse_file_$parse_file_ptr
AZ03/r0	parse_file_$parse_file_set_break
AZ03/r0	parse_file_$parse_file_unset_break
AK50/r0	pass_util
AG92/r2	pat
AR97/aA	patch_firmware
AN53/r0	patch_ring_zero
AG92/r2	path (AF)
AZ03/r0	pause
AG92/r2	pb
AK92/aB	pbm
AR97/aA	pcd
AZ03/r0	pcd
AG92/r2	pd (AF)
AK50/r0	pdt_copy
AG92/r2	pdwd
AZ03/r0	pel
AZ03/r0	pem
AZ03/r0	peo
AZ03/r0	peol
AG92/r2	pg
AG92/r2	pgt
AG92/r2	pi
AG92/r2	pl1
AG92/r2	pl1_abs
AK92/aB	pli_

AK92/aB	plu
AG92/r2	plus (AF)
AG92/r2	pm
AS68/r0	pmf
AS68/r0	pmisc
AN52/r0	pmlm
AG92/r2	pmotd
AR97/aA	poll_mos_memory
AG92/r2	ppa
AN52/r0	pph
AG92/r2	pr
AZ03/r0	prelink
AK92/aB	prepare_mc_restart_
AK92/aB	prepare_mc_restart_$replace
AK92/aB	prepare_mc_restart_$retry
AK92/aB	prepare_mc_restart_$tra
AZ03/r0	pri
AG92/r2	print
AN53/r0	print_apt_entry
AN53/r0	print_aste_ptp
AG92/r2	print_attach_table
AG92/r2	print_auth_names
AK92/aB	print_bind_map
AG93/aD	print_cobol_error_
AG93/aD	print_cobol_error_$switch
AR97/aA	print_configuration_deck
AZ03/r0	print_configuration_deck
AG92/r2	print_default_wdir
AK50/r0	print_devices
AK50/r0	print_disk
AN53/r0	print_dump_seg_name_
AN53/r0	print_dump_seg_name_$get_ptr
AN53/r0	print_dump_seg_name_$hard
AN53/r0	print_dump_tape
AZ03/r0	print_error_message
AN52/r0	print_gen_info_
AR97/aA	print_heals_message
AK50/r0	print_iod_tables
AK92/aB	print_link_info
AK92/aB	print_linkage_usage
AK50/r0	print_log
AG92/r2	print_messages
AK50/r0	print_meters
AG92/r2	print_motd
AN52/r0	print_paging_histogram
AK51/r1	print_pdt
AK50/r0	print_pnt
AG92/r2	print_proc_auth
AK50/r0	print_projfile
AZ03/r0	print_relocation_info
AK50/r0	print_reqfile
AG92/r2	print_request_types
AK50/r0	print_sat
AG92/r2	print_search_rules
AK50/r0	print_syserr_log
AR97/aA	print_syserr_log
AG92/r2	print_translator_search_rules
AN52/r0	print_tuning_parameters
AK50/r0	print_urf
AG92/r2	print_wdir
AK50/r0	priv_move_quota
AZ03/r0	privileged_prelink
AG92/r2	probe
AG92/r2	profile
AG92/r2	program_interrupt
AG92/r2	progress
AS68/r0	proj_mtd
AK51/r1	proj_usage_report
AG92/r2	prt
AG92/r2	psr
AN52/r0	ptp
AG92/r2	ptsr
AG92/r2	pwd
AG92/r2	qedx
AG92/r2	query (AF)
AG92/r2	quotient (AF)
AG92/r2	qx
AG93/aD	random_
AG93/aD	random_$exponential
AG93/aD	random_$exponential_seq
AG93/aD	random_$get_seed
AG93/aD	random_$normal
AG93/aD	random_$normal_ant
AG93/aD	random_$normal_ant_seq
AG93/aD	random_$normal_seq
AG93/aD	random_$set_seed
AG93/aD	random_$uniform
AG93/aD	random_$uniform_ant
AG93/aD	random_$uniform_ant_seq
AG93/aD	random_$uniform_seq
AG92/r2	rc
AZ03/r0	rdc

AG92/r2	rdf
AG92/r2	rdn
AG92/r2	rdy
AG92/r2	re
AK92/aB	read_allowed_
AK92/aB	read_write_allowed_
AG92/r2	ready
AG92/r2	ready_off
AG92/r2	ready_on
AK50/r0	rebuild_dir
AZ03/r0	rebuild_dir
AK50/r0	reclassify_dir
AK50/r0	reclassify_seg
AK50/r0	reclassify_sys_seg
AG93/aD	record_stream_
AS68/r0	recov
AZ03/r0	reduction_compiler
AS68/r0	register
AK50/r0	register_mdir
AG92/r2	release
AK92/aB	release_area_
AG93/aD	release_temp_segments_
AK50/r0	remove_user
AK50/r0	remove_user$persnt
AK50/r0	remove_user$urf
AG92/r2	rename
AS68/r0	rename_proj
AK92/aB	reorder_archive
AZ03/r0	repeat_line
AG92/r2	repeat_query
AG92/r2	reprint_error
AK50/r0	reset_cdt_meters
AK50/r0	reset_disk_meters
AK92/aB	reset_external_variables
AK50/r0	reset_soos
AZ03/r0	reset_tpd
AK50/r0	reset_usage
AK50/r0	reset_use_totals
AZ03/r0	resetcopysw
AG92/r2	resource_usage
AG92/r2	response (AF)
AK50/r0	restore_pdt_access
AG92/r2	rf
AG92/r2	rfa
AZ03/r0	ring0_get_
AZ03/r0	ring0_get_$definition
AZ03/r0	ring0_get_$definition_given_slt
AZ03/r0	ring0_get_$name
AZ03/r0	ring0_get_$names
AZ03/r0	ring0_get_$segptr
AN69/r0	ring_1_lock_
AN53/r0	ring_zero_dump
AG92/r2	rl
AG92/r2	rn
AZ03/r0	rpl
AS68/r0	rqm
AG92/r2	ru
AG92/r2	run_cobol
AG92/r2	runoff
AG92/r2	runoff_abs
AG92/r2	sa
AK50/r0	sac
AG92/r2	safety_sw_off
AG92/r2	safety_sw_on
AK50/r0	save_dir_info
AZ03/r0	save_dir_info
AG92/r2	sbc
AG92/r2	scl
AG92/r2	scr
AG92/r2	search (AF)
AG92/r2	segments (AF)
AG92/r2	segs (AF)
AK50/r0	send_admin_command
AG93/aD	send_mail_
AG92/r2	send_message
AG92/r2	send_message_acknowledge
AG92/r2	send_message_express
AG92/r2	send_message_silent
AG92/r2	set_acl
AG92/r2	set_bit_count
AG92/r2	set_cc
AG92/r2	set_com_line
AK50/r0	set_dir_quota
AG92/r2	set_iacl_dir
AG92/r2	set_iacl_seg
AG93/aD	set_lock_
AG93/aD	set_lock_$lock
AG93/aD	set_lock_$unlock
AK92/aB	set_max_length
AK51/r1	set_mdir_account
AK51/r1	set_mdir_owner
AK51/r1	set_mdir_quota
AR97/aA	set_mos_polling_time
AK50/r0	set_pnt_audit
AK50/r0	set_pnt_auth
AK50/r0	set_pnt_gpw
AR97/aA	set_proc_required

COMMANDS/MANUALS CROSS-REFERENCE

AK92/aB	set_ring_brackets
AK50/r0	set_sat_audit
AK50/r0	set_sat_auth
AG92/r2	set_search_rules
AK50/r0	set_sons_volume
AK50/r0	set_system_priv
AK50/aA	set_system_search_rules
AK92/aB	set_system_storage
AZ03/r0	set_timax
AZ03/r0	set_tpd
AK50/r0	set_tpp
AG92/r2	set_translator_search_rules
AG92/r2	set_tty
AK92/aB	set_user_storage
AK51/r1	set_volume_quota
AZ03/r0	setcopysw
AS68/r0	setcrank
AS68/r0	setdisk
AG92/r2	sid
AK92/aB	signal_
AG92/r2	sis
AG92/r2	slave
AN52/r0	slm
AG92/r2	sm
AG92/r2	sma
AK92/aB	sml
AG92/r2	sms
AG92/r2	smx
AG92/r2	sort
AZ03/r0	sort_items_
AZ03/r0	sort_items_$bit
AZ03/r0	sort_items_$fixed_bin
AZ03/r0	sort_items_$float_bin
AZ03/r0	sort_items_$general
AZ03/r0	sort_items_$varying_char
AZ03/r0	sort_items_indirect_
AZ03/r0	sort_items_indirect_$adj_char
AZ03/r0	sort_items_indirect_$bit
AZ03/r0	sort_items_indirect_$fixed_bin
AZ03/r0	sort_items_indirect_$float_bin
AZ03/r0	sort_items_indirect_$general
AZ03/r0	sort_items_indirect_$varying-char
AK50/r0	sort_projfile
AK50/r0	sort_reqfile
AG92/r2	sort_seg
AG92/r2	spe (AF)
AN52/r0	spg
AN52/r0	spg_ring_0_info_
AN52/r0	spg_util_
AR97/aA	sprq
AG92/r2	sr
AK92/aB	srb
AG92/r2	ss
AG92/r2	ssf
AG92/r2	ssn
AG92/r2	ssr
AG92/r2	st
AG92/r2	start
AG92/r2	status
AZ03/r0	stm
AG92/r2	stop_cobol_run
AK50/r0	stpp
AG92/r2	string (AF)
AG92/r2	strip (AF)
AG92/r2	strip_entry (AF)
AG92/r2	stsr
AG92/r2	stty
AK92/aB	stu_
AK92/aB	stu_$decode_runtime_value
AK92/aB	stu_$find_block
AK92/aB	stu_$find_header
AK92/aB	stu_$find_runtime_symbol
AK92/aB	stu_$get_implicit_qualifier
AK92/aB	stu_$get_line
AK92/aB	stu_$get_line_no
AK92/aB	stu_$get_location
AK92/aB	stu_$get_runtime_address
AK92/aB	stu_$get_runtime_block
AK92/aB	stu_$get_runtime_line_no
AK92/aB	stu_$get_runtime_location
AK92/aB	stu_$get_statement_map
AK92/aB	stu_$offset_to_pointer
AK92/aB	stu_$pointer_to_offset
AK92/aB	stu_$remote_format

COMMANDS/MANUALS CROSS-REFERENCE

AK92/aB	sub_err_
AG92/r2	substr (AF)
AG92/r2	suffix (AF)
AK50/r0	sweep
AK51/r1	sweep
AZ03/r0	sweep_disk_
AG93/aD	syn_
AK50/r0	sys_full_report
AK92/aB	sys_info_
AG92/r2	system (AF)
AK50/r0	system_daily_report
AK92/aB	system_info_
AK92/aB	system_info_$abs_prices
AK92/aB	system_info_$access_ceiling
AK92/aB	system_info_$category_names
AK92/aB	system_info_$device_prices
AK92/aB	system_info_$installation_id
AK92/aB	system_info_$io_prices
AK92/aB	system_info_$last_shutdown
AK92/aB	system_info_$level_names
AK92/aB	system_info_$next_shutdown
AK92/aB	system_info_$prices
AK92/aB	system_info_$shift_table
AK92/aB	system_info_$sysid
AK92/aB	system_info_$timeup
AK92/aB	system_info_$titles
AK92/aB	system_info_$users
AN52/r0	system_link_meters
AK50/r0	system_monthly_report
AN52/r0	system_performance_graph
AK50/r0	system_total
AG92/r2	tc
AN52/r0	tcm
AN52/r0	tcq
AZ03/r0	teco
AG93/aD	term_
AG93/aD	term_$refname
AG93/aD	term_$seg_ptr
AG93/aD	term_$single_refname
AG93/aD	term_$unsnap
AG92/r2	terminate
AG92/r2	terminate_refname
AG92/r2	terminate_refname
AG92/r2	terminate_segno
AG92/r2	terminate_segno
AG92/r2	terminate_single_refname
AG92/r2	terminate_single_refname
AR97/aA	test_cpu
AR97/aA	test_tape
AG92/r2	time (AF)
AK92/aB	timer_manager_
AK92/aB	timer_manager_$alarm_call
AK92/aB	timer_manager_$alarm_call_inhibit
AK92/aB	timer_manager_$alarm_wakeup
AK92/aB	timer_manager_$cpu_call
AK92/aB	timer_manager_$cpu_call_inhibit
AK92/aB	timer_manager_$cpu_wakeup
AK92/aB	timer_manager_$reset_alarm_call
AK92/aB	timer_manager_$reset_alarm_wakeup
AK92/aB	timer_manager_$reset_cpu_call
AK92/aB	timer_manager_$reset_cpu_wakeup
AK92/aB	timer_manager_$sleep
AG92/r2	times (AF)
AN52/r0	tln
AG92/r2	tm
AG92/r2	tmr
AG92/r2	tmr
AG92/r2	tms
AG92/r2	tms
AG92/r2	tmsr
AG92/r2	tmsr
AN52/r0	total_time_meters
AG92/r2	trace_
AG92/r2	trace_stack
AN52/r0	traffic_control_meters
AN52/r0	traffic_control_queue
AZ03/r0	translator_info_

AZ03/r0	translator_info_$get_source_info
AK50/r0	trim_syserr_log
AG92/r2	trunc (AF)
AG92/r2	truncate
AR97/aA	truncate_heals_log
AG92/r2	ts
AK92/aB	tssi_
AK92/aB	tssi_$clean_up_file
AK92/aB	tssi_$clean_up_segment
AK92/aB	tssi_$finish_file
AK92/aB	tssi_$finish_segment
AK92/aB	tssi_$get_file
AK92/aB	tssi_$get_segment
AN52/r0	ttm
AG93/aD	tty_
AK92/aB	tty_
AK50/r0	tty_lines
AN52/r0	tty_lines
AN52/r0	tty_meters
AN52/r0	ttym
AG92/r2	ul
AG92/r2	unassign_resource
AS68/r0	undelegate
AG92/r2	underline (AF)
AG92/r2	unique (AF)
AG93/aD	unique_bits_
AG93/aD	unique_chars_
AG92/r2	unlink
AK92/aB	unwinder_
AK50/r0	up_ctr
AR97/aA	update_heals_log
AS68/r0	upmf
AG92/r2	ur
AK50/r0	urfp
AK50/aA	usage_and_revenue
AK50/r0	usage_total
AG92/r2	user (AF)
AG93/aD	user_info_
AG93/aD	user_info_
AG93/aD	user_info_$absentee_queue
AG93/aD	user_info_$absin
AG93/aD	user_info_$absout
AG93/aD	user_info_$attributes
AG93/aD	user_info_$homedir
AG93/aD	user_info_$limits
AG93/aD	user_info_$load_ctl_info
AG93/aD	user_info_$login_data
AG93/aD	user_info_$logout_data
AG93/aD	user_info_$outer_module
AG93/aD	user_info_$responder
AG93/aD	user_info_$tty_data
AG93/aD	user_info_$usage_data
AG93/aD	user_info_$whoami
AK50/r0	value
AS68/r0	value
AS68/r0	value$dump
AS68/r0	value$set
AK50/r0	value$set_seg
AG92/r2	verify (AF)
AG92/r2	vfa
AG93/aD	vfile_
AK92/aB	vfile_
AG92/r2	vfile_adjust
AG92/r2	vfile_status
AG93/aD	vfile_status_
AG92/r2	vfs
AG93/aD	virtual_cpu_time_
AZ03/r0	vtoc_pathname
AG92/r2	walk_subtree
AG92/r2	wd (AF)
AG92/r2	wh
AG92/r2	where
AG92/r2	who
AS68/r0	who_delg
AZ03/r0	whotab (data base)
AK51/r1	work_class_meters
AK50/r0	write_acct_bill
AK92/aB	write_allowed_
AK50/r0	write_billing_summary
AZ03/r0	write_mst
AK50/r0	write_user_usage_report
AG92/r2	ws
AG92/r2	year (AF)

This page has intentionally

been left blank

APPENDIX F

ERROR MESSAGES

CONTENTS (cont)

SECTION I

INTRODUCTION

The purpose of this document is to aid the user in understanding errors that may occur during use of Multics. The standard Multics error reporting mechanism is introduced, and the meaning of error messages is explained. Possible causes of many errors are discussed, and when applicable, methods for recovering from, or circumventing an error are presented.

HOW TO READ THIS DOCUMENT

This document is intended for use by novice users or applications programmers unfamiliar with the Multics programming environment. As a result, the kinds of errors covered and the examples given have been tailored to this audience.

The discussion of error conditions is divided into four sections. The first explains command processor errors. This includes all errors that arise in the interpretation of commands, and the formation of command names and command arguments. The second section, entitled "Command Errors", deals with errors that are detected by commands themselves. For example, an editor will report an error when a request is issued to read a nonexistant file. The third section discusses execution errors: hardware (simulated) conditions that arise during the execution of programs. Examples of this class of errors include zero divide, subscript range, and segment fault. Such errors may occur in user or system programs and generally indicate the presence of a program bug. As a result, this section is of the greatest use to programmers creating and debugging a new program. The fourth section covers fatal process errors. These are similar in origin to execution errors, but cause the user's process to be aborted. Language-specific errors such as input/output, BASIC run time errors, etc. are not covered by this memo.

Each section describes the most commonly occurring errors of the particular class. Console scripts, along with descriptive commentaries, are used to present typical examples of the error, the methods used to determine the actual cause of the error, and means for recovering from the error. The memo is partially tutorial; however, for complex explanations, the user will be referred to the MPM.

This document may be used as a reference for individual errors. However, it is felt that the reader would benefit by at least one reading from cover to cover. This should provide experience with errors that otherwise could only only be obtained by a long period of actual use. Furthermore, many errors have related causes and consequently related methods for analysis, and since, in general, a topic is explained in full detail only once, a thorough reading will help the user see how a specific technique or error falls into the overall scheme. Finally, the reader should find a number of worthwhile hints that enable him or her to avoid problems in the first place.

User feedback is solicited concerning errors in this document, awkward or incomprehensible explanations, common problems that have been left out, etc.

THE PROGRAMMING ENVIRONMENT

To obtain a good understanding of errors and their causes, it is helpful to understand the environment in which programs run. (Often, too, it is simply a misunderstanding that is the cause of the error.) Below is a brief and simplified discussion of the basic features and terminology of the Multics programming environment. Refer to the MPM Reference Guide (Order No. AG91) for more detailed information.

A process is a computation or execution activity much like a job on other computer systems. The term refers, collectively, to the program or group of programs in execution, the current value of program variables, the address space (the temporary and permanent segments of memory that have been referenced), the files and other input/output devices (e.g., console, tape volumes) in use, and various system-maintained data bases. What is important to the programmer is the duration of the process, since it maintains the programming environment. When the user first logs in (1) a new process is created. At this time, all per process information is put into a consistent initial state. There are no files or devices attached besides the terminal; all program variables are (in effect) set to their initial values; and so forth. A process is terminated when the user logs out, gives the new_proc command, or a fatal error occurs.

On Multics, programs and their data reside in a virtual memory or address space. This consists of (potentially) a large number of disjoint segments of storage. Each such segment may contain up to 255K (K = 1024) words of storage, and has a name by which it may be accessed. An address (pointer) designating a specific location in the

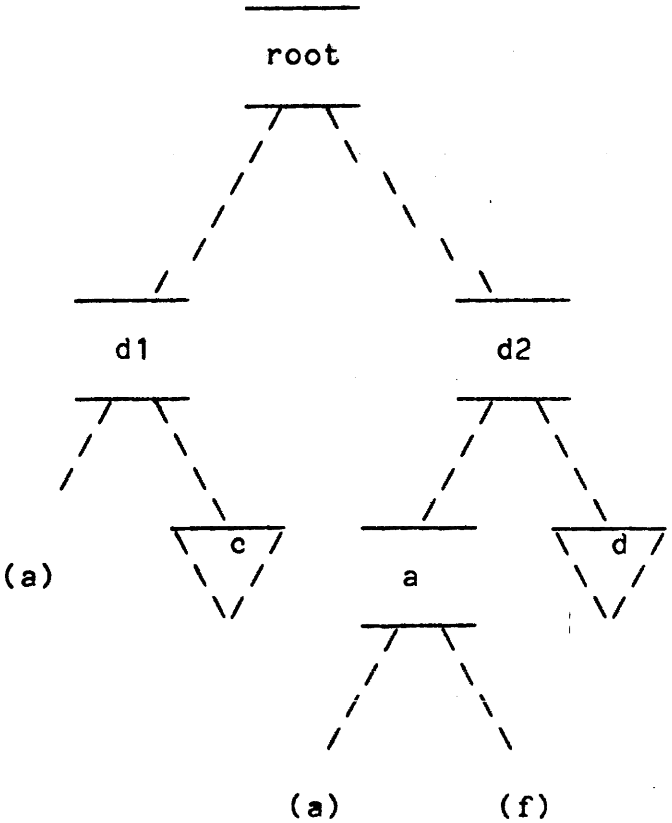
(1) Or equivalently an absentee job is logged in at the user's request by the system, in which case, the console is replaced by the absentee input segment.

virtual address space is a pair of numbers: a segment number and an offset within the segment. When a segment is to be used, it is initiated forming a mapping between the name and a segment number. This mapping remains valid for the duration of a process, or until the segment is explicitly terminated. Use of an invalid segment number, in particular a segment number generated in another process, is a common source of errors. (See the description of the segment fault error.) An offset within a segment, on the other hand, remains valid as long as the contents of the segment are not modified. This means, for example, that it is possible to save the offset part of an address across processes and reuse the offset with a segment number generated in the new process. (1) Errors occur in the use of offsets if the value of the offset is out or range: negative or greater than 255K. (A lower limit can be set by explicit programmer action, if desired.) See the discussion of out of bounds errors.

A segment may be designated by one of two naming conventions: pathnames and reference names. In general, a segment is specified by pathname when accessed by commands, and by reference name when accessed from a program as an external symbol (e.g., the name of an external subroutine).

A pathname designates an unique segment within the Multics file system. Every segment has one or more entrynames by which it is cataloged in a directory. Within a single directory, the entrynames of all entries are unique; entrynames of entries in different directories need not be distinct. In addition to segments, a directory may contain other subdirectories. The Multics file system is organized as a tree with a root directory that contains all other directories as subdirectories, subsubdirectories, etc.

[1] PL/I programmers may realize the similarity between this and the restrictions on the use of based areas and offsets on other systems. Programs are not allowed to save pointers to data allocated within the area from job to job because while the area may retain its integrity, the actual address may change. The same reasoning applies to segment numbers and offsets, and to areas on Multics saved from process to process.



A pathname, then, is an ordered list of entrynames designating the directories containing the entry and the entry itself. It gives the path from the root to the segment. In the diagram above, squares represent directories and diamonds represent segments. The following are all valid pathnames within that hierarchy:

```
>d1>a
>d2>a
>d2>a>a
```

In addition to segments and directories, a directory may also contain links. A link is a pathname pointing to another entry (usually in another directory); it allows that entry to be accessed as if it were cataloged in the directory containing the link with the name(s) of the link. In the diagram above, links are presented by triangles with dotted lines leading to the target of the link.

```
          >d1>c     and   >d2>a,
    and   >d1>c>f   and   >d2>d
```

refer to the same directory and segment respectively. If a link points to another link, then the entry pointed to by the second link becomes the ultimate target.

There are two forms of pathnames. An absolute pathname contains the entrynames of all the containing directories; that is, it fully specifies the position of something (i.e., segment, directory, link) within the directory hierarchy. A relative pathname, on the other hand, gives the position of an entry relative to the working directory. A relative pathname is expanded into an absolute pathname by logically concatenating the pathname of the working directory to the relative pathname given. For example, if the working directory were >d2>a from the the diagram above, the relative pathname "f" would be the same as the absolute pathname >d2>a>f. It is possible to move upwards in the hierachy by using "<"s in a relative pathname. Thus with the same working directory as above, <d would be the same as >d2>d. The working directory is initially the user's home directory, normally >udd>Project_ID>Person_ID; it may be changed by the user with the change_wdir (cwd) command. Absolute pathnames are distinquished from relative ones by a leading ">".

The term segment is often confused with the term file. Strictly speaking, a file is storage accessed through explicit input/output operations. In principle, there are no limitations on the size of a file; it may get as large as is necessary, far exceeding the 255K word limit of a segment. Those files smaller than 255K are placed in a single segment. Unstructured files of character data are the same as text segments. That is, they may be created or modified in an editor, printed, or sorted. Structured files (those containing nonprintable "binary" data) should not be acessed except through I/O facilities. Those files larger than 255K are implemented as multisegment files. The data is stored in several segments which should only be accessed by I/O mechanisms. Even if the file contains only printable text, there are currently no mechanisms for editing it and guaranteeing the authenticity of the file. A multisegment file is cataloged in a directory like any other entry. In this memo, the distinction between file and segment will be strictly adhered to.

A reference name is the name by which a segment is referenced (e.g., called) during the execution of a program. It consists of the entryname of the segment; the absolute pathname of a particular segment is determined by applying the search rules in effect for the process. They are as follows.

1. A list of reference names already known within the process is checked; if the entryname being searched for is found, the associated segment is used. A name becomes known when the search rules are first used to reference a segment with that entryname or when a specific segment is initiated and a reference name (which need not be an entryname of the segment) is specified for the segment.

2. The directory containing the program making the reference is checked for a segment with the entryname. If found, the segment found becomes known by the reference name.

3. Each of an ordered list of directories is checked for a segment with the correct entryname. If found, the segment becomes known by the reference name. Initially, this list

starts with the user's working directory and is followed by the list of system libraries. Alterations may be made to this list of directories by using the add_search_rules command. For example, the working directory (which is logically specified as "working_dir" and whose meaning changes as the working directory is changed) may be moved to follow the system libraries, or a specific user directory may be added to the search rules.

Once a reference name is associated with a particular segment, it remains in effect (rule 1) until the process is terminated or until the segment is terminated by using the terminate command. This can cause unexpected results for the unsuspecting user. In the sample hierarchy given above, if the user's current working directory is >d1 and he or she executes the procedure a in that directory, it will become known by the reference name a. If later, he changes to a new working directory, >d2>a, and executes the procedure f which calls a subroutine a, then the segment with reference name a found by the search rules will not be the segment >d2>a>a, but rather the segment previously known by that reference name, >d1>a.

For each process, the system creates a process directory. This directory contains temporary data segments and special segments in which program data and system control information are stored. The stack is one such segment and contains records of current procedure activations and the automatic storage used by these procedures. The combined linkage segment contains the linkage sections of procedures which hold the internal static data used by a program. The identity of these segments can be important in determining the cause of an error. When an error occurs, the segment containing the data being referenced will be given when appropriate. If the segment is the stack, it is a strong clue that the operation causing the error was making use of a datum of automatic storage. Similarly if the segment is the combined linked segment (combined_linkage_r.nn), it is likely that static data is involved. Often such a clue can be used to isolate the cause of an error in the absence of other debugging information such as the line on which the error occurred.

The list of procedure calls contained in the stack is an extremely useful source of debugging information. Consider the following example. Assume that a procedure A is executing. It then calls procedure B. B in turn calls C. When C has finished executing, it returns and B continues from the point at which it called C. B eventually returns, and A then calls the procedure C. At each step, it is possible to construct an ordered list of the procedure invocations indicating which program called which other program and the location in a program at which it called out. Graphically, this list may be represented as follows:

```
                 C
       B         B         B                   C
       A         A         A         A         A

   A calls B   B calls C   C returns   B returns   A calls C
```

In the figure above, the most recently called procedure is on top. Such a stack trace is particularly useful to understand the sequence of calls that lead up to an error. The probe and trace_stack commands may be used to obtain a stack trace. Examples of the use of the probe command are given in later sections. A block is activated when a procedure or subroutine is called or a begin block is entered. Activation causes dynamic allocation of storage for a program's automatic variables. The area allocated is placed at the end (top) of the process's stack and is called a stack frame. When the procedure returns, the stack frame is freed. This means that variables of the automatic storage class (1) only have storage assigned to them (and hence a legitimate address and value) when the block in which they are declared is active (i.e., its execution is ongoing or suspended as the result of an error condition, a quit signal, a call, or a debugger breakpoint). When a procedure is called recursively, it has frames on the stack for each activation. Each instance of the procedure on the stack is considered a separate and distinquishable activation of the procedure. The values of automatic variables and parameters can be (and usually are) different in different activations of the same procedure.

Within a process, PL/I static variables and Fortran variables have a continuity of value. That is, the value of such a variable is initialized only once during a process, when a program in which the variable is defined is first invoked; subsequent modifications to the value of the variable will remain in effect, even through new activations of the procedure, until the process is terminated. (2)

(1) PL/I variables, by default, belong to the automatic class. FORTRAN variables appearing in an automatic statement (a nonstandard Multics feature) are also of this class.

(2) PL/I internal static variables or Fortran variables that are not in common may be reinitialized by terminating the segment. External

CONVENTIONS

The examples in this document follow certain conventions. All lines typed by the user are prefixed by an exclamation point ( ). A quit signal issued by the user by pressing the QUIT, ATTN, or BREAK keys on the terminal is indicated by "(quit)" appearing to the left of the normal text of the example. When a line of output is too long to be placed on a single line, it is broken at an arbitrary point. This should not affect the user's understanding of the example. In ready messages output by the system on the completion of a command line, a dot (.) replaces information, such as the time of day, irrelevant to the example being presented. Finally, many lines of input or output whose content should be obvious from the example is replaced by ellipsis (...). In all other respects, an attempt has been made to produce examples which are accurate. Most are simply modifications of the output from test error situations.

and commoned variables may be reinitialized only by using the new_pro command to obtain a new process.

SECTION II

COMMAND PROCESSOR ERRORS

These are errors that are detected by the command processor When this kind of error is detected, a message is printed on the console, (along with a ready message), and then the system waits for the user to type another command. Once the cause of the problem has been determined, the user may retry the (corrected) command.

These errors may often be caused by a garbled telephone connection between the user's terminal and the computer. If there is no apparent cause for an error, reenter the same command line again.

SEGMENT NNNN NOT FOUND

This means that reference name NNNN did not match any reference or entryname within the user's search rules. The most common cause of the error is incorrectly specifying (through mistyping or a misconception) the command name.

For example, suppose that a user had a program called colour, but mistakenly typed "color" when calling it:

```
color
Segment color not found.
r ....

list

Segments = 3, Length = 3.

re     1  colour
r w    1  colour.pl1
rew    1  PJApple.mbx

r ....

colour
...
```

Here the user gets the error, then uses the list command to see if there was something wrong with the name. Finding the mistake, the command line is retyped with the corrected name.

This error is virtually identical to the "segment not found" case of linkage errors. See that discussion below for additional examples.

ENTRY POINT EEEE NOT FOUND IN SEGMENT

This indicates that a segment matching the reference name NNNN was found in the user's search rules, but it did not contain the entry point called EEEE. To determine what entry points are actually present in a program (or other type of object segments) the print_link_info command may be used.

```
       colour
       Entry point colour not found in segment colour.
       r ....

       pli colour -entry

                    colour    02/05/76 1540.4 est Thu

       3 Definitions:

       segname:     colour
       symb¦0       symbol_table
       text¦17      colour              Entry: text¦17

       r ....

       colour$color
       ...
```

In the above example, a segment matching the reference name colour (see previous example) was found but it did not contain an entry point "colour". The pli command gives the "segname" or the name by which the program was known when it was compiled; the only "entry" defined is one called "color". The user corrects the line by giving a command name that contains both the reference name and entry point name separated by a "$". Another way to correct the error would be to rename the program (both the source and the object segments):

```
       rename colour.** color.==
       r ....

       color
       r ....
```

In this way, the segment now has the same entryname and entry poin name and can therefore be called as a command by giving only it entryname. (1)

The problem illustrated here occurs quite often when the program contains a procedure with a different name than that given to the segment containing the text of the program.

```
edm colour.pl1
Input.
color: procedure;

   ...

end color;
.
Edit.
w
q
r ....
```

There are a number of other causes for this error. The entrypoint may have been deleted by binding.

This error is virtually identical in cause to the "external definition not found" case of linkage errors. See that discussion below for additional examples. If the meaning of reference names versus entrynames and entry point names is confusing, see the MPM Reference Guide, Section III, Constructing and Interpreting Names.

EXPANDED COMMAND LINE IS TOO LONG

This error arises when an active string has been used, and the string returned overflows the currently allocated size for the command line. The user may recover by using the set_com_line command to set the command line length to some arbitrarily large value (e.g., 3000) and reissuing the command line. Below, the active string segs is invoked to return a string containing the names of all PL/I and Fortran source segments; this string overflows the command line length.

```
archive ad saved_programs [segs *.pl1 *.fortran]
command_processor_: Expanded command line is too long.
r ....

set_com_line 3000
r ....

archive ad saved_programs [segs *.pl1 *.fortran]
r ....
```

(1) For an explanation of the star and equals convention, see the MPM Reference Guide, Section III, Constructing and Interpreting Names.

IMPROPER SYNTAX IN COMMAND NAME

This error is issued when the user has specified a command name which is not in the standard form of a reference name, optionally followed by the special character "$" and an entry point name. Examples of correctly formed command names are:

```
          ref_name              ref_name$entry_point_name
```

Examples of incorrectly formatted names are:

```
           name$          $name
```

More detailed information may be found in Section III of the MPM Reference Guide.

BAD SYNTAX IN PATHNAME

This means that the program to be invoked was specified by an incorrectly formed absolute or relative pathname. See the discussion of the error in the Command Errors section.

PARENTHESES DO NOT BALANCE

This means simply that a parenthesis beginning an iteration set wa unbalanced. For example:

```
       create >udd>Serpent>PJApple>(output1 output2
       command_processor_: Parentheses do not balance.
       r ....
```

What was intended was "(output1 output2)" to create two segments. This error is handled by reentering a command line containing a balancing parenthesis.

The problem may arise when iteration of a command is not intended. For instance, the "send_message" command which transmits its arguments to another party.

```
       send_message BDLucifer Serpent delete the files (in PJApple>old
       command_processor_: Parentheses do not balance.
       r ....
```

Here the intent is to send a message containing a parenthetical thought. If the command line were reentered with a trailing parenthesis, two messages would be sent. That is, the user named would receive

```
       from PJApple.Serpent: delete the files in

       =: delete the files PJApple>old
```

Notice that the first message contains the first string in the iteration set "(in PJApple>old)", and the second message, the second string. This problem may be avoided by enclosing the entire message in quotes.

```
send_message BDLucifer Serpent "delete the files (in PJApple>old)"
r ....
```

It is advisable to always enclose messages in quotes to avoid unintentionally sending someone repeated messages.

BRACKETS DO NOT BALANCE

An invocation of an active function (a procedure returning a string to be inserted into the command line) is enclosed in square brackets. This error simply means that the command line had an unbalanced left or right bracket.

```
list -pathname [pd
command_processor_: Brackets do not balance.
r ....
```

The correct command line would have contained "[pd]" to return the name of the user's process directory. The error may be handled by entering a corrected command line.

In a manner similar to that described for the case of unbalanced parentheses, an unintentionally balanced active function invocation in a "send_message" command line will transmit a message containing the value of the active function. The problem may be avoided by enclosing the message in quotes.

QUOTES DO NOT BALANCE

Quotation marks may be used in a command line to delimit a single string argument that contains special characters such as "(", "]", or space. The error means that the command line contained an unbalanced quote. The error may be remedied by reentering the corrected command line.

SECTION III

COMMAND ERRORS

These are errors detected in the processing of a command. They reflect not hardware conditions, but rather software conditions concerning, for example, the file system. Command errors are not restartable. A message is printed, and the system resumes what it was doing (e.g., listening for commands). The cause of the error may be fixed, and the command reissued.

BAD SYNTAX IN PATHNAME

This means that a pathname (the ordered list of entrynames identifying a segment in the storage system) has been formed incorrectly. The causes of this error are typing mistakes and an incomplete understanding of what a pathname is. (In the latter case, see the MPM Reference Guide, Section III, Constructing and Interpreting Names.)

```
print >udd>Serpent>>PJApple>a.basic
print: Bad syntax in pathname. >udd>Serpent>>PJApple>a.basic
r ....

print >udd>Serpent>PJApple>a.basic
...
```

Here the user gave a pathname with two ">"s next to each other. As this is incorrect syntax, an error message was printed. The user recovered by typing the correct pathname. This error will also occur if a "<" appears out of place in a relative pathname, that is, at any place other than the beginning of the pathname. For example:

```
<<Student<Green>old.runoff
```

INCORRECT ACCESS ON ENTRY

This means that the user, when logged in under the current project, does not have the correct access to a segment to perform a certain operation. This error can be dealt with by using the command "list_acl" (la) to determine why the user has no access, who can give him or her access, and if the user can do it him/herself, use the set_acl command to set the appropriate access to the segment.

The error may arise when trying to read a segment or file. For example, when reading a segment with an editor (edm or qedx), when printing a file (using print or dprint), or when trying to compile a source program. In this case, the user does not have "read" access to the segment. For example, the following dialogue might occur for user Green logged in under the Serpent project.

```
edm color.pl1
edm: Incorrect access on entry. >udd>Serpent>PJApple>color.basic
r ....

list_acl color.pl1
r w   BDLucifer.Serpent.*
r w   *.SysDaemon.*
r ....

set_acl color.pl1 r
r ....

edm color.pl1
...
```

Here the user has attempted to edit segment color.pl1. The edm command detects that he does not have read access to the segment, and reports an error. By using the "list_acl" command, he finds that only one other user on the Serpent project (BDLucifer.Serpent) has access to the file. The "set_acl" command is used to give Green.Serpent access to the file, and he retries the command over again. The error may also occur when writing out a segment that is being edited. In this case, the user does not have "write" access to the segment.

```
edm color.pl1
Edit.

...

w
edm: Incorrect access on entry.  >udd>Serpent>PJApple>color.pl1
E set_acl color.pl1 rw
Edit.
w
q
r ....
```

Here the user tries to save a program that he has been editing, but cannot do so because he does not have write access to the segment. He is faced with the problem of setting the access on the segment without losing the editing that he had done. The edm "E" requests allows him to execute a Multics command line, so this feature is used to let him invoke the set_acl command to recover from the error. After the access is changed, the edm write request may be reissued. If the user had been unable to change the access, he could at least save what was done by writing it out into another segment as shown below:

```
E set_acl color.pl1 rw
set_acl: Incorrect access to directory containing entry.
Edit.
w color1.pl1
q
```

INCORRECT ACCESS TO DIRECTORY CONTAINING ENTRY

This error means that the user on his or her current project does not have enough access on the directory in which a segment is (to be) cataloged to perform some operation on it. Again, this error can be dealt with by using the list_acl and set_acl commands.

This error most commonly occurs while trying to delete a segment (the user lacks modify access), while trying to set the access on a segment (lacks modify) while trying to move, create, or copy a segment (lacks modify and/or append), or while trying to find out information about the segment (lacks status permission).

```
status <BDLucifer>souls.list
status: Incorrect access to directory containing entry.
        >udd>Serpent>BDLucifer>souls.list
r ....


list_acl >udd>Serpent>BDLucifer
sma    BDLucifer.*.*
sma    *.SysDaemon.*
r ....

list_acl >udd>Serpent
sma    *.Serpent.*
sma    *.SysDaemon.*

r ....

set_acl <BDLucifer s
r ....

status <BDLucifer>souls.list
...
```

Here the user attempts to find out information about the segment. The status command requires at least "s" access to the containing directory, and not having it, an error message is issued. The user then checks the fact, and looks at her access to the parent of the directory containing the segment to see if she can set the appropriate access herself. She then gives herself the necessary access, and reissues the command.

SOME DIRECTORY IN PATH SPECIFIED DOES NOT EXIST

This means that a directory specified in the pathname of a segment does not actually exist.

The way to determine what directory is missing and/or the entryname of the directory actually intended is to use the list command.

```
print >udd>Serpent>PApple>color.pl1
print: Some directory in path specified does not exist.
        >udd>Serpent>PApple>color.pl1
r ....

list -pn >udd>Serpent -dr

Directories = 2.

     PJApple
sma  BDLucifer

r ....

print >udd>Serpent>PJApple>color.pl1
...
```

ENTRY NOT FOUND

This means that a segment specifed was not found in the directory. (All the containing directories do exist.)

This error may be dealt with by using the list command to see if the segment exists under some other entryname. The rename or addname commands can be used as desired to change the segment's entryname or give it an additional entryname.

A common cause of this error in the case of novice users is misnaming the segment. For example, a Fortran source program must have the suffix ".fortran". Thus if the segment "main" had been created containing the program, an error would ensue:

```
edm main
Input.
...
w
q
r ....

fortran main
Fortran
fortran: Entry not found. main.fortran
r ....

rename main main.fortran
r ....

fortran main
...
```

If the name identifies a link, then another possible cause of the error is that the segment pointed to by the link does not exist. This possibility can be checked by listing the link and checking whether the target exists. (Note that the link target may be another link, in which case the process must be repeated.)

INSUFFICIENT ACCESS TO RETURN ANY INFORMATION

This error arises in the cases described for the above four errors when the user does not even have enough access to determine why the operation cannot be performed. The problem is that the user does not have status permission on the directory containing a segment or, in the second case, to the directory containing the directory containing the segment.

This error can be dealt with as described above by first setting access on the containing directory. Usually, a user who receives this error will not have access to set the required access, and will have to contact the user who controls the directory in question.

ILLEGAL ENTRYNAME

This message is generated by an editor when the user tries to write from an editor buffer into a segment with a malformed name. A malformed name is one which contains special characters such as blank, tab, "/", etc., or which contains missing components. Generally, this is a name which would make it difficult to access the segment because of system conventions. Examples of illegal names are:

```
a*b
ho/whose/
c..d
prog.
```

This almost always occurs when the user has given an accidental write request. For example:

```
edm second.fortran
Edit.
...
w = a*b
edm: Illegal entryname. = a*b
...
```

If it is desired to have a segment with a name containing such special characters, the segment can be written with a normal name, and renamed to the entryname containing special characters.

SECTION IV

EXECUTION ERRORS

This class of errors includes all hardware and software detected faults and conditions. When an error of this sort occurs, a condition is signalled. The condition may be handled by a user-supplied condition handler (PL/I on unit), or if no on unit is found (as is normally the case), the system's on unit. The system's on unit prints an error message and invokes a new command level, suspending the execution of the program causing the error. This new command level is indicated by a ready message with a level number greater than one:

```
     r 1204 2.039 39.200 347 level 2,16
```

After an error has occurred, and a new command level entered, the user should eventually do one of three things:

1. Issue a release command to flush execution of the suspended program. For example, a quit signal (sent by striking the ATTN, BREAK, or QUIT key on the terminal) may be used to stop a run away program, or excessive printing.

   ```
             looper
      (quit)
             QUIT
             r .... level 3,.

             release
             r ....
   ```

   The release command need not be used immediately after the error occurs. If the cause of the error is not obvious, system supplied tools (e.g., probe) may be invoked at the new command level to determine the cause. Whether or not this is possible, the release command should be issued before doing any additional work (e.g., changing and recompiling the program) to avoid more serious and incomprehensible errors.

2. The start command may be used to restart the program that was interrupted. This is possible if the problem is correctable, or in the case of an erroneous computation where the system's on unit performs some specified action to correct the condition upon restart. Such a correction might be to set the result of the computation, 2 ** -1000, to 0 after an underflow condition has occurred. The actions taken by the system on unit are often specified in the error message; if not, the user may consult the MPM Reference Guide if he or she desires.

   Another common practice is to "quit" out of a program that appears to be looping, check the CPU time that it has used by inspecting the ready message, and if it is looping, releasing the suspended program (after debugging the cause of the loop); otherwise, resuming the execution with "start". (Note: quit/starting in this way may lose output directed to the terminal. However, under certain circumstances, this may be desirable.)

```
              count
                 1
                 2
       (quit)
              QUIT
              r .... level 2,.

              start
                 6
                 7
       (quit)
              QUIT
              r .... level 3,.

              release -all
              r ....
```

   Here a program named count has been invoked. It was then stopped by a quit signal and restarted by the start command; as a result, a few lines of output were lost. The program was then stopped a second time by a quit signal and aborted by the release command. Notice how the level numbers are effected by this sequence.

3. Issue the new_proc command to get a new process. This will reinitialize all static variables, common blocks, I/O attachments, files, etc. The use of this command is recommended when inexplicable (1) errors occur. Once a new_proc is finished, it is

(1) Errors for which you can see no apparent reason. For example, you followed a memo verbatim; exactly the same thing worked before; an so forth. In general, a new_proc should be tried if you have not seen the error before and cannot find a ready explanation.

advisable (1) to retry the program with which there is a problem. Often the problem will disappear. If it doesn't, it is likely that a program bug exists, and you should continue to look for some other cause. The thing to remember is that an erroneous program can cause other programs including system programs to go awry.

The error messages produced for most of this class of runtime errors are in a common format. For example:

```
Error: Attempt to divide by zero at >udd>Serpent>PJApple>prog¦13 (line
system handler for error returns to command level
```

The first line gives the type of error "Attempt to divide by zero" the pathname of the object segment causing the error (>udd>...>prog), the offset in the program object segment of the instruction at which the error occurred (13 octal), and, if the program were compiled with the "-table" option, the source line number. The second line gives additional information about the error. Here it states that a new command level will be created.

In general, an error which occurs in a system program can be traced to a user error. (2) In the case of an error in a system program, the user should verify that he has called it properly: that the correct number of arguments have been passed, that all documented requirements and restrictions have been met, and that all values passed as input to the system program have reasonable values.

When an error occurs in a system program, the location in the user program where the system program was called is not given in the error message. This location can be determined using the probe command.

(1) If the cost of program execution prior to occurrence of the error is acceptable.

(2) This is not to say that there are not bugs in system programs, however it is more likely that the user did something wrong.

```
prog

Error while processing in ring 0:
Segment-fault error by bound_system_faults¦24330
referencing 1777¦0
There was an attempt to use an invalid segment number.
Entry into lower ring was by
change_wdir$get_wdir_¦476
(>system_library_standard>bound_fscom1_)
referencing hcs_$fs_search_get_wdir
r .... level 2,.

probe
Condition seg_fault_error raised at get_wdir_¦16344.
stack
 11     command_processor_
 10     release_stack
  9     unclaimed_signal
  8     default_error_handler_
  7     real_sdh_
  6     get_wdir_                     seg_fault_error
  5     prog
  4     command_processor_
  3     listen_
  2     process_overseer_
  1     user_init_admin_
use prog
source
          this -> wdir = get_wdir_ ();
where source
Current line is line 9 of prog.
   ...
```

The error message may be interpreted as follows. The error was a segment fault error. The message "Error while processing in ring 0" indicates that the error occurred in a supervisor program. The supervisor program was called from the user ring by the program change_wdir$get_wdir_ (which is located in a bound segment in a system library). (1) The error message does not indicate what user program called these programs. This is determined by invoking probe, and issuing special requests to it. The stack request lists the procedure activations currently on the user stack (the called procedures). Here "get_wdir_" has been called by the user program "prog". The use request tells probe to examine "prog", and the source request then gives the text of the line at which the error occurs; "where" retrieves the line number of that line. Note that these (and most other) probe requests can only be used as shown when the program in question has been compiled with the -table option.

(1) The other information is largely irrelevant for the purposes of the casual user.

RECORD QUOTA OVERFLOW

This means that the disk storage quota for a directory has been exceeded. The problem can be dealt with only by finding or reclaiming additional disk quota. Usually, it is possible to have the storage quota on a directory increased by the project administrator. (1) Quota can also be temporarily reclaimed by deleting unneeded segments, or if possible moving a segment to a directory with unused quota.

This error will occur when a segment is being written into. There are three common contexts in which this will occur. Writing out an editor buffer, compiling a program to produce an object segment, and outputting a file from a user program using language I/O facilities. The following is an example of a record quota overflow occurring while writing an editor buffer. This is a particularly dangerous problem for if it can not be corrected, the changes made to the text or source file will be lost.

```
edm prog.pl1
Edit.

...

w

Error while processing in ring 0:
record_quota_overflow condition by bound_file_system|3627
referencing >udd>Serpent>PJApple>prog.pl1|30046

Entry into lower ring was by
>system_library_standard>edm|5047
referencing hcs_$truncate_seg
r .... level 2,.

get_quota -wd
quota = 2; used = 2
r .... level 2,.

list

Segments = 3, Lengths = 2.

re      1  xxxx
r w     0  prog.pl1
r w     1  xxxx.pl1

r .... level 2,.
```

(1) This is normally the person or group through which funding has been arranged. For most users, this will be the User Accounts Office. Students in courses should contact their instructor. Users on the Student project, should contact the SIPB.

```
        delete xxxx
        r .... level 2,.

        get_quota -wd
        quota = 2; used = 1
        r .... level 2,.

        start
        q
        r ....
```

When the user tries to save his changes to the file "prog.pl1", the error occurs. The error message indicates that the error occurred in a supervisor routine called from the editor program.

The get_quota command gives the current value of the quota and the number of records currently charged against it. The list command gives the lengths (see below) of the files in the directory as well as their names. The user then deletes the object segment "xxxx" to make room to write the file in the editor. (Note that information is not being lost here, as the source file "xxxx.pl1" can be recompiled to create the object segment once sufficient quota has been obtained.) The get_quota command shows that a single record of storage has been freed up, and the start request causes the editor write operation to be restarted. Since he is finished editing, the user quits out of the editor.

If the user then tried to compile the program, another record quota overflow would occur because with the file "prog.pl1" successfully written, the number of records used is once again two.

```
        pl1 prog -table
        PL/I

        Error:  record_quota_overflow condition by compile_entry¦621
        (>system_library_standard>bound_cg_)
        referencing >udd>Serpent>PJApple>prog¦0

        r .... level 2,.
```

In this particular case, the user can do nothing to gain additional storage, except delete the source file xxxx.pl1. If the file were of no use, the choice of deleting it would be acceptable; and the user could restart the compilation with the "start" command. If it were unacceptable, the user could only wait for additional storage. The problem would not be as critical as in the case above, as no information would be lost by logging out.

The third context in which a record quota overflow might occur is in a user program that writes output to a file.

```
        filewriter

        Error:  record_quota_overflow condition by open_uns_file$put_chars_un
        (>system_library_standard>bound_vfile_)
        referencing >udd>Serpent>PJApple>output_file¦0
        r .... level 2,.
```

Note: The list command normally will give the "length" of the segments listed. This is the apparent size of the segment as computed from the bit count. However, bit count can be inconsistent with the number of records used by the segment if the segment contains zero page(s) (see above), or if the bit count has never been set (which can occur for a file which was never closed or a data structure or common block overlaid on a segment). Continuing with the example above:

```
        get_quota -wd
        quota = 3; records = 3.
        r ....

        list

        Segments = 2, Lengths = 4.

        re      3  test
        r w     1  test.fortran

        r ....
```

The get_quota command shows 3 records now in use (an additional record filled in by completing the compilation); but the list command shows lengths totaling four records. If the "-records" option is supplied to the list command, the actual records used by the segment will be given.

```
        list -records

        Segments = 2, Records = 3.

          2  test
          1  test.fortran

        r ....
```

This is consistent with the results given by get_quota. The point is that when looking for segments to delete or move when reclaiming disk quota, the "-records" option should be used to obtain the correct disk usage for the segment. In addition, It cannot be assumed that a file is empty because its length is zero. Rather, it is empty only when its records used is zero.

LINKAGE ERROR

This error will occur when a program tries to reference an external symbol, for example, an external program or PL/I external data, and the specified symbol is not found. If the source of the error can be determined, and the problem fixed, the program can be restarted. Note that it is possible to "fix" a linkage error in such a way as to cause another flavor of the same error to occur when the program is restarted. A little thought will prevent this from happening however.

There are four subclasses of linkage errors:

Segment Not Found

It means that a segment with the specified reference name was not found anywhere in the user's search rules. For example, assume that procedure "prog" calls another program, "zzzz$aaaa", which for some reason cannot be found.

```
prog

Error:  Linkage error by >udd>Serpent>PJApple>prog¦20 (line 34)
Referencing zzzz¦aaaa.
Segment not found.
r .... level 2,.
```

The basic approach for dealing with this error is to list the directories within which the program or data segment was thought to be in order to determine which of the following four cases apply.

- the segment referenced really did not exist

- the segment referenced exists, but its name was given incorrectly (e.g., misspelled).

- an entry (segment or link) of the correct name exists within the search rules, but was ignored

- the referenced segment exists in a directory not in the search rules.

The typical user who is working alone, (i.e., not using programs in some "private" library) and is only using his own programs, standard system commands and subroutines will usually only have to consider the first two cases. Below is a further description of each case.

a. The segment may not exist. For example, it may have never been created. A common problem for new users is forgetting to compile the program. Continuing with the above example:

```
list

Segments = 3, Lengths = 3.

re      1  prog
r w     1  zzzz.fortran
r w     1  prog.pl1

r .... level 2,.
```

Notice that there are source and object segments for prog, but only a source segment for zzzz (zzzz.fortran). The cause of the problem then, is that there is no object segment named "zzzz" to be found. Compiling the program (as shown below) will create such a segment; restarting execution will cause the search for the segment to be repeated, and this time found.

```
fortran zzzz
fortran
r .... level 2,.

list

Segments = 4, Lengths = 4.

re      1  zzzz
re      1  prog
r w     1  zzzz.fortran
r w     1  prog.pl1

r .... level 2,.

start
...
```

b. No segment of the designated name may exist. This can happen if the user is confused about the name of the segment. For example, if a PL/I program is called "subr" (i.e., subr is the label on the procedure statement) but the program resides in a segment of another name (e.g., subroutine), calling "subr" from another program will cause this error. This problem can be fixed by renaming (with the rename command) the segment (and the source segment) containing "subr".

```
rename subroutine.** subr.==
r .... level 2,.
```

c. If the search rules have been stated as the user desires,, and there is in fact a segment of the correct name in one of the directories in the search rules, then the most likely cause of the error is that the user has no access to the segment. For example, a "list" might show

```
list -pn >udd>Serpent>PJApple

Segments = 3, Lengths = 4

re       1  prog
         2  zzzz
r w      1  prog.pl1

r .... level 2,.
```

Notice that there is no access ("re", "rw", etc.) listed for the segment zzzz. This problem should be corrected with as described in Section IIIB (Incorrect Access On Entry, etc.).

The second cause of this problem is a link of the correct name which points to a non existent segment or a segment to which the user has no access. A non-existent segment can be caused by the segment having been moved or deleted or the target pathname being incorrect. This might appear in a listing of the directory as follows:

```
list -pn >udd>Serpent>PJApple
Segs=0;Msfs=0;Dirs=0;Links=1.
r .... level 2,.

list -pn >udd>Serpent>PJApple -link

Links = 1.

zzzz                      >udd>Serpent>BDLucifer>zzzz

r .... level 2,.

initiate >udd>Serpent>PJApple>zzzz
initiate: Entry not found. zzzz
r .... level 2,.
```

The first list command (listing segments) shows that there are no segments in the directory, but that there is one link. The second list command shows the link to a segment in another directory. The initiate command is used to determine the reason why the segment pointed to by the link was ignored in the search. Here it does not exist. If the target pathname is incorrect in that a directory is named incorrectly, the command error "Some directory in path specified does not exist." would be reported. If the problem is no access, the error would be "Incorrect access on entry."

d. While a segment of the correct name may be known to exist, the directory containing it is not in the search rules. The current search rules may be listed with the "print_search_rules" command.

```
print_search_rules
initiated_segments
referencing_dir
```

```
       working_dir
       >system_library_standard
       >system_library_unbundled
       >system_library_1
       >system_library_tools
       >system_library_auth_maint
       r .... level 2,.

       list -pn >udd>Serpent>PJApple
       ...
```

Note that this is a common problem to users of packages such as BMD, IMSL, SSP, or the M.I.T. Calcomp subroutines. These programs are kept in directories outside of the normal search rules. (1)

In general, when it has been determined that a segment to be referenced is outside of the search rules, one of three things can be done. The search rules can be adjusted to include the directory containing the segment; the segment may be initiated; or a link to the segment can be created. For example, assume that the segment in question is the IMSL subroutine eigrf. The search rules can be corrected with the "add_search_rules" command. (2) The problem could be resolved by:

```
       add_search_rules >libraries>imsl -after >system_library_unbundled
       r .... level 2,.

       print_search_rules
       initiated_segments
       referencing_dir
       working_dir
       >system_library_standard
       >system_library_unbundled
       >libraries>imsl
          ....
       r .... level 2,.
```

Here, the print_search_rule command has been used to show the corrected search rules. This approach is useful when the missing segment is one of a collection of programs in the same directory (like a program library) whose other members are also likely to be used.

(1) The documentation for such program libraries will usually specify how to make use of those routines. This advice should be followed.

(2) In the example below, the new search rule is added after >system_library_unbundled rather than after working_dir to avoid searching >libraries>IMSL every time a command or subroutine is referenced for the first time in the process.

The segment may also be initiated. This is useful when there is only one program needed, and it is likely to be used only within the current process.

```
initiate >libraries>imsl>eigrf
r .... level 2,.
```

A link to the program may also be created. This need only be done once, and will enable the program to be referenced without issuing additional commands at any time in the future provided that the directory containing the link remains within the search rules. The simplest way to ensure this is to place the link in the directory containing the calling program itself.

```
link >libraries>imsl>eigrf
r .... level 2,.

where eigrf
>libraries>IMSL>eigrf
r .... level 2,.
```

The "where" command gives the pathname of the segment whose reference name is given. That is the segment that will be invoked if a program of the name given is called. It has been used here to verify that the link was successful.

External Symbol Not Found

This means that a segment matching the reference name specified was found, but that the (perhaps implicitly) specified entry point was not.

```
prog
Error:  Linkage error by >udd>Serpent>PJApple>prog¦34 (line 38)
referencing xxxx¦aaaa
External symbol not found.
r .... level 2,.
```

This means that the segment xxxx was found, but the external entry point (symbol) "aaaa" was not found in the segment. In addition to trivial naming and typing mistakes, one of the more frequent causes for the error is that the program resides in a segment with a name different from the one used on the procedure statement of the program. The program is then called using the segment name.

```
edm test.pl1
Segment not found.
Input.
tester: procedure (a);
   dcl a float binary(27);
   a = a ** 2;
end;
.
Edit.
w
q
r ....

pl1 test -table
PL/I
r ....

test
Error:  Linkage error by >udd>Serpent>PJApple>call_test¦54 (line 24)
referencing test¦test
External symbol not found.
r .... level 2,.
```

This problem can be eliminated only by changing the name on the procedure statement from tester to test and recompiling the program.

Linkage Section Not Found

This means that a segment of the specified name was found, but that the segment did not have a linkage section (i.e., it is not an object segment).

```
prog
Error:  Linkage error by >udd>Serpent>PJApple>prog¦43 (line 42)
referencing xxxx¦aaaa
Linkage section not found.
r .... level 2,.
```

This may occur if the name of a data or test segment was specified instead of the name of an actual compiled program. For example, a common problem is a source segment which is given the name of its object segment.

```
list

Segments = 4, Lengths = 4.

r w     1  xxxx
           xxxx.pl1
re      1  prog
r w     1  prog.pl1

r .... level 2,.
```

The list command shows the two names on the source file "xxxx.pl1". When "xxxx" was referenced from the program, it was this segment that was found, but it was not a valid object segment.

To recover from this particular error, the name must be deleted from the segment, and the text compiled into the object program to be called. The program can then be restarted.

```
delete_name xxxx
r .... level 2,.

pl1 xxxx
PL/I
r .... level 2,.

start
 ...
```

There is No Room to Make Requested Allocation

This means that the size of a named external data area exceeded the system limit of 255K words. Examples of such areas are named common blocks in FORTRAN and external symbols given reference names (names containing a "$") in PL/I. For example

```
nospace: procedure;
   declare bigarea$ (300000) external fixed binary;
   ...
   bigarea$ (1) = ... ;
   ...
end;
```

Executing the above program would produced the following error:

```
nospace

Error:  Linkage error by >udd>Serpent>PJApple>nospace¦11 (line 4
referencing bigarea¦
(with a create-if-not-found link)
There is no room to make requested allocation.
r .... level 2,.
```

When such an error occurs in a PL/I program, the user should examine the declaration of the external symbol and calculate the size. If it is a structure containing elements each smaller than the limit, the structure can be broken up. For example

```
     declare

          1 extstruc$ external,
            2 a (100000) fixed bin,
            2 b (100000) float bin,
            2 c (100000) float bin(63);
```

would occupy a total of 400,000 words of storage. Member a uses one word per element; b, a single precision real value, uses one word per element; and c, a double precision real value, uses two words per element. It can be broken up into two or three small structures:

```
     declare

          1 extstruc1$ external,
            2 a (100000) fixed binary,
          1 extstruc2$ external,
            2 b (100000) float binary,
          1 extstruc3$ external,
            2 c (100000) fixed binary;
```

If the symbol being created is one large array, then the programmer should attempt to reduce the size of the array needed. If such a reduction is not possible, it may be possible to simulate the array as an array of pointers to cross sections of the original array.

```
     declare
          array$ (3,100000) external fixed binary;
```

would cause the error described here. This could be replaced by

```
     declare
          array (100000) fixed binary based,
          arrp (3) pointer initial
            (addr (array1$), addr (array2$), addr (array3$)),
          (array1$, array2$, array3$) (100000) external fixed binary;
```

with the program edited to replace all references to

```
          array (x, y)       by     arrp (x) -> array (y)
```

Similar problems occur in FORTRAN when very large common blocks are used. As in PL/I, there are two cases: when there are many small members of the common block, and when there is one very big member. In the first case, the problem can again be dealt with by splitting up the common block.

```
          common /data/ a(100000), b(100000), c(100000)
```

becomes

```
          common /data1/ a(100000), b(100000)
          common /data2/ c (100000)
```

In the second case, that of one very large member, there is no method to get around the problem that is particularly efficient. The best that can be done is to write a function that references cross sections of the array defined in different common blocks.

```
          common array (3,100000)
```

becomes

```
     function array (x, y)
          common /data1/ array1 (100000)
          common /data2/ array2 (100000)
          common /data3/ array3 (100000)

          go to (1, 2, 3) x
     1    return (array1 (y))
     2    return (array2 (y))
     3    return (array3 (y))
     end
```

SEGMENT FAULT

This error means that the program has addressed a non-existent segment. What has happened is that an address value (pointer, entry, or label) contains an invalid segment number. There are two general causes: using an uninitialized address datum, and using an address value designating a segment after that segment has been deleted.

A deleted segment may be referenced under the following circumstances. If the program was, at the time of being deleted, still active (its execution suspended by a quit signal or error condition).

```
        prog

        Error:  Attempt to divide by zero at >udd>Serpent>PJApple>prog¦24
                (line 12).
        System handler for error returns to command level.
        r .... level 2,.

        ...

        delete prog
        r .... level 2,.
        ...

        release

        Error:  Segment-fault error by unwind_stack_¦120
        (>system_library_1>bound_sss_active_)
        (while in pl1 operator cp_csa)
        referencing 367¦2
        There was an attempt to use an invalid segment number.
        r .... level 2,.
```

Or if the segment is an input or output file that was not closed prior to deleting the segment.

```
        prog
(quit)
        QUIT
        r .... level 2,.

        delete output_file
        r .... level 2,.

        release
        r ....

        prog

        Error:  Segment-fault error by open_uns_file$put_chars_uns_file¦1036
        (>system_library_standard>bound_vfile_)
        referencing 345¦0
        There was an attempt to use an invalid segment number.
        r .... level 2,.
```

An uninitialized address value is usually caused by forgetting to initialize the corresponding variable. [1]

```
        prog

        Error:  Segment-fault error by >udd>Serpent>PJApple>prog¦327
        (line 43) referencing 2349¦27
        There was an attempt to use an invalid segment number.
        r .... level 2,.

        probe
        Condition segfault raised at line 43 of prog.
```

[1] This can also cause any of the other bad address problems described under other errors. An uninitialized pointer may cause a worthless value to be displayed for a variable qualified by the pointer or for the pointer itself. (Most uninitialized automatic pointers point into the stack).

```
       source
                 p -> data = 3;
       value p
          2349¦27
         ...
```

In FORTRAN, address data problems may occur as well. One cause is passing an array argument to a FORTRAN subroutine whose corresponding parameter is not dimensioned. When the program references this parameter with subscripts, FORTRAN treats the parameter as an entry value. For example, executing a program of the following form

```
          subroutine mattran (arrin, arrout)
               dimension arrout(4,4)
               ...
               arrout (i,j) = arrin (j,i)
               ...
          end
```

could cause an error of the form

```
       mattran_test

       Error: Segment-fault error by >udd>Serpent>PJApple>mattran¦143
       (line 12) referencing 327¦756
       There has been an attempt to use an invalid segment number.
       r .... level 2,.

       probe
       Condition segfault raised at line 12 of mattran.
       source
                 arrout (i,j) = arrin (j,i)
       value i; value j
          1
          1
       value arrin (i,j)

       Error: Segment-fault error by print_reference¦2373
       (>system_library_standard>bound_probe_¦14517)
       referencing 327¦756
       There has been an attempt to use an invalid segment number.
       r .... level 3,.

       pi
       symbol arrin
       entry variable parameter
       ...
```

Here, the subroutine mattran has been called from mattran_test. A segment fault error occurs on line 12, and probe is invoked to look for the cause of the problem. The "source" request gives the text o the statement in which the error occurred; "value" requests enable the user to determine with what variable the program is having difficulty. The error reoccurs when the value of "arrin (i, j)" is displayed, indicating that "arrin" is the problem. The program_interrupt (pi) command is used to probe, and the "symbol" requests used to display the attributes of the variable. The output shows that it is an entry variable and not an array at all.

Another cause would be passing too few arguments to a subroutine. In this case, referencing a parameter for which there is no corresponding argument may cause a segfault or other addressing error.

NO EXECUTE PERMISSION

This means that the user's process is attempting to execute a segment to which it does not have execute access. Upon getting this error, the user should attempt to set access (or have the access set for him by the owner of the segment) to read and execute, [1] and if successful, restart the program.

```
prog

Error: no_execute_permission condition by command_processor_|522
(>system_library_1>bound_command_loop_)
referencing >udd>Serpent>PJApple>prog|3
r .... level 2,.

set_acl prog re
r .... level 2,.

start
...
```

This can occur if the access has been set incorrectly on the segment. For instance, if a "set_acl ** rw" command has been issued in the directory, or if the user had created the object segment before compiling by using the create command.

The error can also occur when an uninitialized label or entry variable is referenced. This particular case can be distinguished from the others by the identity of the segment being referenced. If it is one which could be expected to be called (e.g., in the example above, "prog" is being called), then the problem is probably a simple access error; on the other hand, if the segment , is a data or text segment, etc., then the problem is probably an uninitialized address datum.

[1] At least "re" access is necessary for an object segment; "e" alone will not suffice. Write access is not advisable.

NO READ/WRITE PERMISSION

These mean that the process lacks the access required to read or write the segment mentioned in the error message.

```
        prog

        Error:  no_write_permission condition by prog¦412 (line 101)
        referencing >udd>Serpent>PJApple>data_seg¦2
        r .... level 2,.
```

The simplest cause is having failed to set or obtain the necessary access. As with a no_execute_permission error above, the user may attempt to set the required access, and then restart the program.

This problem may also be caused by bad address data. This case may be distinguished from a simple access error as also given above.

NOT IN READ/EXECUTE/WRITE/CALL BRACKET

This means that an attempt has been made to reference an inner ring segment. The cause is almost always bad address data.

```
        prog3

        Error:  not_in_write_bracket condition by prog¦26 (line 5)
        referencing dseg¦0

        r .... level 2,.
```

The identity of the segment being referenced can often give a clue to the variable whose value is bad. A reference to dseg, as occurred here, usually indicates that a packed pointer (a pointer value declared unaligned) is uninitialized. A reference to the stack or linkage section, is strong evidence that an automatic or static, respectively, aligned pointer, label or entry value has not been assigned a value.

FAULT TAG 1/FAULT TAG 3

This means that an addressing modification fault has occurred while attempting to indirect through a pointer. Since these modifiers never appear in PL/I pointer datums, the problem is usually uninitialized address data.

```
        prog

        Error:  fault_tag_1 by >udd>Serpent>PJApple>prog¦14 (line 8)
        referencing stack_4¦3320 (in process dir)
        Ascii data where pointer expected.
        r ....  level 2,.
```

The fact that the program was referencing some data in stack_4 at the time of the error indicates that the bad pointer was an automatic value. If the program had been referencing "combined linkage_section 4.00, the bad pointer would be a static value. The address modifier may also be encountered when trying to execute data. In such a case, the error message will indicate that the segment causing the error is a data segment such as the stack or the combined linkage section.

The error fault_tag_1 is often caused by an uninitialized pointer occupying space previously filled with ASCII data (hence the second part of the error message).

ILLEGAL MODIFIER

This means that an illegal address modifier has been used. It may appear in a pointer value or in data being executed as regular instructions.

```
      prog

      Error: illegal_modifier condition by >udd>Serpent>PJApple>prog¦44
      (Line 18) referencing stack_4¦0 (in process dir)
      Possible illegal modifier in indirect chain or uninitialized pointer.
      r .... level 2,.
```

The causes of this error are identical to those of a fault_tag_1/3 error. It is also not restartable. The problem must be corrected before the program can be run again.

ATTEMPT TO REFERENCE THROUGH A NULL POINTER

This means that a null pointer has been used as a locator value qualifying a reference to a based variable. It usually indicates a logical bug in the program.

```
      prog

      Error:  Attempt by >udd>Serpent>PJApple>prog¦57 (line 23)
      to reference through null pointer
      r .... level 2,.
```

The programmer should carefully examine his or her program to determine how the locator (pointer or offset) value could have a null value at the location in which the error occurred. The variable may not be referenced with an explicit qualifier

```
     data        instead of        p1 -> data
```

In this case, the default qualifier e.g., based, (p) is used, and its value should be checked.

```
probe
Condition simfault_000001 raised at line 23 of prog.
source
   result = based_num + 4;
symbol based_num
fixed binary(17) aligned based (p)
Declared in prog.
value p
   null
...
```

This error may also occur for controlled as well as based data, if a controlled variable is referenced before it is allocated.

SIMFAULT NNNNNN

This means that the programmer has attempted to use a pointer with a segment number of -1 and an octal offset of NNNNNN. The cause is use of uninitialized address data.

Note: a pointer with segment number -1 and offset 000001 is a null pointer. In such a case, the error message reads "Attempt to reference through a null pointer" as described above. The condition simfault_000001 is signalled explicitly only when the pointer value is not entirely a valid null pointer (for example, it has a non-zero bit offset).

ILLEGAL MACHINE OPERATION

This means that there has been an attempt to execute an undefined machine instruction.

```
prog

Error:  Illegal machine operation by prog¦4
Current instruction is:
 000004   000000000000      ....    0
r .... level 2,.
```

The two most common causes of this error are: branching to a nonexistent element of a constant label array, or using an uninitialized label or entry value. The segment in which the error occurs may be used to distinguish the two cases. In the former, the segment will be one of those in use (in the example above prog); in the later, it will be a data segment (stack or linkage section) or some other unexpected segment.

STORAGE CONDITION

There are two causes of this error. First, the user has attempted to allocate more based or controlled storage than is available in the system area. This will be acompanied by the message that system storage is full.

```
        prog

        Error:  storage condition by >udd>Serpent>PJApple>prog¦154 (line 52)
        System storage for based and controlled variables is full.
        system handler for error returns to command level
        r .... level 2,.
```

The programmer should inspect the declaration of the variable being allocated. The system cannot allocate more than 262,144 words of storage for any one variable. If the variable being allocated has an expression for a string length or array bound, the value of those expressions should be checked. Often they may involve undefined values. If all allocations are relatively small (e.g., hundreds or low thousands of words), the problem may be that the allocation is being repeated too many times. A check should be made for an infinite loop involving the allocation.

Second, and most common, is that the stack has overflowed. This error will be accompanied by the message that the stack has been extended.

```
        prog

        Error:  storage condition by >udd>Serpent>PJApple>prog¦166 (line 58)
        Attempt to reference beyond end of stack. Stack has been extended.
        system handler for error returns to command level
        r .... level 2,.
```

This error will first occur when more than 64K words of stack space are required, or when a reference is made past the first 64K of stack. The stack is extended to the next 48K boundary. Depending on the cause for extending the stack, it may be permissible to restart the program with the start command. Subsequent storage conditions may occur if additional storage is required/referenced, and the stack will be extended in 48K increments up to a maximum length of 208K. Any attempt to use more than that will cause a fatal process error (see below).

One cause of this error is that the program is recursing too deeply (or infinitely). This case can be verified by examining the frame number, the second of the two numbers in the in ready message level information:

```
            r .... level .,137
```

A value in the hundreds is a certain sign of trouble. (1) The cummulative automatic storage requirements for a moderately recursive program (or set of programs) may also be too great. The required storage can be determined from a compilation listing produced with the "-map" option (under the heading "storage requirements for the program"). If the storage requirements will not exceed the maximum of 208K, it is safe to restart the program.

An excessively large stack frame size may also arise if there are automatic variables declared with expression length or array bounds, and the expressions reference uninitialized values. A common mistake is to make use of another automatic variable in such an expression whether or not that variable has an initial value specified. For example, a program containing the declaration

```
declare
     array (array_dim) fixed binary,
     array_dim fixed binary automatic;
```

could cause the error message appearing above. A debugging session might continue as follows:

```
probe
Condition storage raised at line 58 of prog.
source
     call subr (...);
```

Here probe has been used to determine where the error occurred. The source request shows that the error occurred while trying to call another subroutine. The reason that the error occurs at this point is that until the subroutine is called (creating a new frame for the subroutine) the stack is not actually extended. So the programmer examines the program for abnormally sized variables:

```
symbol array
fixed binary(17,0) aligned automatic dimension(71902)
Declared in prog.
```

The symbol request gives the evaluated dimensions for the array, showing it to be extremely large. (The error could appear in the same fashion if the large bounds were intended.)

Another cause is subscripting an automatic array with a value far out of bounds. This can be detected in PL/I programs by putting a subscriptrange prefix on the procedure statement.

```
(subscriptrange):
prog: procedure;
```

(1) In fact, a value in excess of 30 to 40 is uncommon, and can generally be regarded as a sign of problems.

```
          .....

        end;
```

In Fortran this can be accomplished by compiling the program with the "-subscriptrange" control argument.

```
        fortran zzzz -table -subscriptrange
```

A similar cause is a string range error; that is, the use of the substr builtin function with out-of-range arguments. In general, this is an initial position (the second argument) which is negative or far past the end of the string, or a length (the third or assumed argument) that is negative or far greater that the actual length of the rest of the string. This error can be trapped by recompiling the program with a "stringrange" prefix on the procedure statement.

A final cause is the invocation of a function that returns a value with star (expression) extents. If the bounds of an array developed as the return argument are bad, or if a bad substr expression or uninitialized character varying string is returned, a storage condition can be raised after the called function has returned, but before the calling program has resumed execution. This is indicated by a storage condition occurring in a system segment. If this is the case, there will be no other information as to what user program was executing at the time of the error.

OUT OF BOUNDS FAULT

This means that a non-existent portion of a segment has been referenced by the program. A storage condition due to a stack overflow is really an out of bounds fault on the stack; as a result, the causes and recovery methods are similar. The most common causes include an out-of-range array subscript or substring reference. The error is particularly common when the data in question is a normal Fortran variable, commoned (occurring in a segment in the process directory), or uncommoned (occurring in the linkage section), or a PL/I internal static variable (occurring in the linkage section), or an external static variable (occurring in a segment in the process directory). If the segment is the program itself, it is likely that the program is referencing outside of the bounds of a label array or an internal static array that has an initial value specified but has never been modified.

ILLEGAL PROCEDURE

This occurs when the hardware is requested to perform an illegal operation. The most usual cause is uninitialized decimal data.

```
baddec

Error: illegal_procedure condition by >udd>Serpent>PJApple>baddec
(line 5) referencing stack_4¦3320 (in process dir)

f ... level 2,.

probe
Condition illegal_procedure raised at line 5 of baddec.
source
          dv = dv + 1;
symbol dv
fixed decimal(7,0) aligned automatic
Declared in baddec.
value dv

Error:  illegal_procedure condition by arithmetic_to_ascii_¦112
(>system_library_standard>bound_trace_)
referencing pl1_operators_¦10

r .... level 3,.

release -all
r ....
```

Here probe has been used to show the source of the line at which the error occurred. It contains a reference to a decimal variable. This is sufficient evidence to believe that the problem is uninitialized decimal data. Displaying the value of the variable will cause the same error again, confirming the diagnosis of the problem. (The release command, with the -all option, is used to flush execution of probe, suspended by the second occurrence of the error, and baddec, suspended by the first.)

Other, less likely, causes of the same error are: transfering to an element of a label array outside of the bounds of the label array, and referencing uninitialized label or entry variables. In the former case, the location of the error will often be listed as the first line of the program; the line from which the condition is signalled will not be available. In the latter case, the location of the error will usually be in some unexpected segment.

CONVERSION

This means that an error has occurred in the conversion of a character string to some other data type. This condition will occur in conversion to an arithmetic value if the string is not a correctly formed number. It will occur in conversion to a bit string if the source character string contains characters other that "1" or "0".

```
badconv

Error: conversion condition by >udd>Serpent>PJApple>badconv¦22
(line 6 onsource = "one", onchar = "o"
Illegal character follows a numeric field.
system handler for error returns to command level
r .... level 2,.
```

The error message gives, in addition to the location at which the error occurred, the values of the PL/I builtin functions, onsource and onchar. Onsource represents the character string being converted; onchar is the (first) character in the string which is invalid for the conversion.

This error may arise during implicit or explicit conversions among variables (or the results of expressions) in the program, or during execution of a get statement when the input is converted to an arithmetic or bit value.

SIZE

This condition has three causes. It will occur when the value assigned to a fixed point datum exceeds the precision of the target -- for example, assigning the value 9999 to a fixed binary(3) datum. The error will occur in this way only only if size checking was enabled for the statement in which the assignment was performed by a size prefix on the statement or the procedure statement. Second, it will occur during picture controlled conversion, if the target field is too small to hold the value being converted. Again size checking must be enabled. Third, it will occur during a put list or put data statement, when the value stored exceeds the precision declared for the variable, or during a put edit statement, if the output field cannot hold the value being output. Size checking is always enabled for put statements.

```
size_err

Error: size condition by >udd>Serpent>PJApple>size_err¦136 (line 14)
Precision of target is insufficient for number of integral
digits assigned to it.
System handler for error returns to command level
r .... level 2,.
```

The user should be aware of a side effect of a size condition raised while executing a put statement. A common debugging technique is to include an error on unit within the program that dumps all the variables:

```
          on error begin;
               put data;
          end;
```

If a size condition occurred, invoking the on unit, the put data statement within the on unit will cause another size condition to be signalled when formating the variable for which the condition was originally signalled. The on unit will be invoked a second time, and the size condition signalled yet another time, and so on, ad infinitum, eventually leading to a storage condition or fatal process error.

ERROR CONDITION

An error condition will be reported when an erroneous state arises in the program, and there is no specific condition for that state. For example, this includes use of mathematical builtin functions with arguments that are out of range.

The following program is used to illustrate a typical situation in which the error condition will be raised.

```
          bigexp: procedure;
               dcl sysprint file;
               put list (exp (2345));  put skip;
          end;
```

Executing the program will cause the condition to be signaled. The system on unit gives the reason for the specific cause of the problem, and states a fixup to be taken if the program is restarted.

```
       bigexp

       Error:  error condition by >udd>Serpent>PJApple>bigexp¦53 (line 3
        exp(x), x > 88.028, not allowed
       Type ""start"" to set result =  .17014118e+39
       r .... level 2,.

       start
        1.701e+038
       r ....
```

After receiving the error, the programmer may decide that the standard fixup is acceptable, and restart the program as has been shown above. Notice that the program proceeds normally to output the result as set by the action of the system on unit.

SUBSCRIPTRANGE

This means that a subscript specified in an array reference is outside of the bounds of the array. The condition is normally raised only when the programmer has specified that subscript range checking be performed (by placing a subscriptrange condition prefix on a PL/I procedure statement, or compiling a Fortran program with the -subscriptrange control argument). Such checking is useful when there are unexplainable storage, out of bounds, or fatal process errors.

```
subrange

Error:  subscriptrange condition by >udd>Serpent>PJApple>subrange|17
(line 7).
A subscript value has exceeded array bounds.
system handler for  error returns to command level
r .... level 2,.

probe
Condition subscriptrange raised at line 7 of subrange.
source
          array (i) = i;
  value i
      5
symbol array
fixed binary(17,0) aligned automatic dimension(4)
Declared in subrange.
...
```

Above is an example of a subscriptrange condition. Upon receiving the error, the programmer enters probe to determine the cause of the problem. The source request gives the text of the line on which the error occurred (line 7). He then displays the value of i and compares it with the dimensions for the array as given by the symbol request. Here the subscript, i, is only a little bit out of range. This indicates a logical bug, specifically, that the program is not constraining the value of the subscript properly. Alternatively, if the value of the subscript were grossly out of range (for example, -72301292), this would be an indication that the problem was that the subscript was uninitialized or assigned the value of some (function of an) uninitialized variable.

This condition may also arise when a function which returns a dimension (*) array is used, and the bounds of the array returned do not match the bounds of the array to which it is assigned. For example, assume that data has dimension (4) and that array_fun returns an array with dimension (5). Then

```
data = array_fun (...);
```

will cause a subscriptrange condition to be signalled.

STRINGRANGE

This means that a substring of a character or bit string value as specified by the substr builtin function is not completely contained within the string value. Given the reference

```
substr (s, i, j)
```

the error implies that one of two conditions is true: that i, specifying the starting position of the substring, is less than one or greater than the current length of the string, or that j, specifying the length of the substring, is less than zero or greater than the number of positions included in that portion of the string from position i to the end.

The stringrange condition will only be raised if the programmer has compiled the program with a stringrange condition prefix on the procedure statement or on the statement which uses the substr built-in function.

```
stringrange

Error:  stringrange condition by >udd>Serpent>PJApple>stringrange
(line 7).  A substring specified by substr is not completely
contained in the first argument.  System handler for condition
returns to command level.
r .... level 2,.

probe
Condition stringrange raised at line 7 of stringrange.
source
          substr (str, 1, i) = "a";
value i
    -1
...
```

FIXEDOVERFLOW, OVERFLOW, UNDERFLOW

These errors indicate that the result of a computation has exceeded the precision of the machine. Fixedoverflow applies to fixed point computations and indicates that the result is too large. It should not be restarted.

```
        folf

        Fixed point overflow by >udd>Serpent>PJApple>folf¦143 (line 27)
        System handler for error returns to command level
        r .... level 2,.
```

Overflow applies to floating point computations, and indicates that the result is too large. It should not be restarted.

```
        olf

        Error:  Exponent overflow by >udd>Serpent>PJApple>olf¦160 (line 33)
        System handler for condition returns to command level
        r .... level 2,.
```

Underflow applys to floating point computations, and indicates that the result is too small. The program is automatically restarted with the result of the computation set to 0.

```
        unfl

        Error:  Exponent underflow by >udd>Serpent>PJApple>unfl¦167 (line 39)
        r ....
```

Notice that after an underflow condition the system does not enter a new command level, but instead continues with the program. Here it has terminated normally, returning to command level 1.

PROGRAM INTERRUPT

The program_interrupt command is used to reenter a command subsystem such as edm or probe after an error condition or quit signal. The command signals the program_interrupt condition which is trapped by the subsystem. If the user mistakenly issues a program interrupt command to reenter a subsystem that does not handle the condition, or when there is no subsystem active, the condition will be reported as an error at command level.

```
        program_interrupt

        Error:  program_interrupt condition by program_interrupt|71
        (>system_library_standard>bound_command_env_)

        r .... level 2,.
```

If there is no subsystem active, the user should issue a release command to flush the program_interrupt condition. If the user is trying to reenter a substem that does not handle program_interrupt, he should issue a release command to flush the program_interrupt and then a start command to reenter the subsystem. (Normally, however, a subsystem may be reentered by a start command only if it was suspended by a quit signal.)

SECTION V

FATAL PROCESS ERRORS

In general, a fatal process error occurs when the system detects a condition such that the process is not able to continue running. (In particular, the system default on unit cannot be executed to interpret the cause of the error.) The action taken in this case is to terminate the process in which the error occurred and to create a new process for the user. Because it is a new process, there is no information available about the programs that were running when the error occurred, the value of program variables, etc. The only clue as to the cause of the error is the error message.

The single most common form of a fatal process error is an out of bounds error on the stack. The causes are the same as for a storage condition (see above) arising on the stack. The message that is generated by the system designates that a fatal error has occurred, and then gives a error message indicating a more specific problem.

```
     Fatal error. Process has terminated. Out of bounds fault on user's
     stack.  New process created.
```

In the event of this kind of fatal process error, it is advisable for the user to recompile his program with subscriptrange and stringrange checking enabled and try the program again. If a stringrange or subscriptrange condition then occurs instead of the fatal process error, then it is likely that new error is the source of the problems.

If the fatal process error recurs despite the checks enabled, then the cause of the problem can be just about anything. It is recommended that the user check his access to all programs and files that he is using to insure proper access. He should also check for the possible causes of a segment fault error. Finally, calls to system programs should be checked to see if they conform to all documented conventions. Should these checks fail to turn up a clue. He should use the probe command to set breakpoints at various strategic points in his program to isolate the point at which the fatal process error is occurring. Often the process will have to be repeated with additional breaks set until the location is narrowed down to a single statement.

There are several other kinds of fatal process errors that the user may see. They include:

No unclaimed signal handler specified for this process.
This means that no default on unit could be found. The possible causes include subscript and stringrange errors, and the use of uninitialized address data. (See above and the previous section.)

Fault in signaller by user's process.
This indicates the presence of a very complex error condition and probably involves more than one cause. The user should apply the methods of described for the other errors and hope for the best.

Unable to perform critical I/O.
This means that the user's process was unable to perform an input or output operation at a crucial point. For example, writing out an error message. This indicates that the I/O attachments for the user_input, user_output, error_output and user_i/o I/O switches are in an untenable state. The user should consider the kinds of operations that he performed prior to the fatal error, and determine if they conformed to documentation.

Process terminated because of system defined error condition.
This is a catch all message. Again, the user should try the methods described above.

The reader should recall the comments about errors that vanish after a new process is created made in the introductory remarks to section IV. They apply to a fatal process error as well.

APPENDIX G

TYMNET DATA COMMUNICATIONS NETWORK

Page

This page has intentionally
been left blank.

THE TYMNET NETWORK

Assignment of TYMNET user id's and passwords contact:

Lacy Johnson
Honeywell LISD
Multics Computer Center
Box 6000 Mail Station K40
Phoenix, AZ 85005
Phone: (602) 249-7303 HVN 341-7303

LOCAL LINES

Do a "help TYMNET_lines" to obtain a list of TYMNET dial-up lines in the major cities serviced by TYMNET.

COST

TYMNET charges $7.50 per connect hour which will be passed on to the appropriate cost center.

THE TYMNET NETWORK

- SUPPORTED TERMINAL

▯ TYMNET supports a wide variety of terminal types. Each terminal type has an identifying character which must be sent to TYMNET when you first log in to establish the transmission characteristics of the terminal.

IDENTIFIER	TERMINAL			REMARKS
	Speed (cps	Baud rate	Type	
b	15	150	TermiNet-300 TTY-37 Hazeltine Execuport	Full Duplex, odd parity
d	10	110	TTY 33/35 TermiNet-300	Full Duplex
a (RECOMMENDED)	30	300	TermiNet-300 Execuport Texas Inst. CSC Hazeltine CRT	For thermal printer terminals, a delay of N+6 or 7 character times is used.
carrigae return	15	134.5	Datel IBM 2741	Multics only accepts EBCDIC terminals through TYMNET.

- Only 300 and 134.5 baud lines are available on Multics for access through the TYMNET network.

THE TYMNET NETWORK

LOGGING INTO TYMNET

1. Turn on the terminal and coupler.

2. If the terminal has adjustable data transmission speeds, set the terminal to 30 cps. This is to handle TYMNET's initial message. Lower settings give a garbled message during login.

3. If you are using a terminal that has an Auto Linefeed switch (such as the TermiNet 300), turn it off during the initial login procedure with TYMNET. Once you have made contact with Multics you may turn it back on.

4. Dial the TYMNET phone number and wait for the high-pitched tone. (If the tone sounds weak or raspy, hang up and re-dial to obtain a better line).

5. Place the telephone handset in the coupler or depress the "DATA" button if using a 103A or 113A dataset.

6. The following message will be sent to your terminal at 10 cps: PLEASE TYPE YOUR TERMINAL IDENTIFIER

7. Set your terminal to the desired speed (if necessary), and type the appropriate letter or CR (carriage return).

8. TYMNET will then send: -XXXX-YY-- PLEASE LOG IN: where: XXXX is the remote access node number, and YY is the port on the node to which your terminal is connected.

9. (OPTIONAL) You may check for the presence of the network supervisor (which controls the Tymsat you have called) by typing a single CR. If the supervisor is in control, the following prompt will appear: USER NAME:

10. You may need the following control characters before entering your user name:

    a) control-H   notifies TYMNET not to echo characters

    b) control-X   to be used if you plan to input data from paper tape, cassette tape, or any other non-keyboard device. TYMNET must be able to start and stop data entry.For this, control-Q starts the device and control-S stops the device.

    c) control-P   to be used if your terminal can only accept even parity.

11. After receiving "PLEASE LOG IN:" or "USER NAME:" enter your TYMNET user id (preceded by control characters if necessary). followed by a CR. (user id may be in upper OR lower case).

12. TYMNET will then prompt you for your password: PASSWORD:

13. Respond by typing your password(in upper or lower case), followed by a CR. NOTE: If you want to eliminate the prompt "PASSWORD:", enter your TYMNET user id, a semi-colon(;), and the password followed by a CR.(THIS MAY NOT WORK FOR NON-ASCII TERMINALS)

14. You should then receive either a semi-colon(;) or the following: P nn (which indicates the port number of the TYCOM connected to Multics) HOST IS ONLINE

15. From here on, the login procedure is exactly as if you were dialing directly into Multics.

16. When you logout from Multics, TYMNET will respond:

```
CP DISCONNECTS  DROPPED BY HOST SYSTEM
PLEASE LOGIN: (at this point you may hang up the phone)
```

THE TYMNET NETWORK

- TYMNET PROBLEM RESPONSES

1. ALL PORTS BUSY	All available Multics TYMNET ports are in use at the transmission speed you have requested. Try again later or try another transmission speed.
2. BUSY TONE	If received when dialing the TYMNET network, wait a few minutes and try again. If the busy tone persists, call the local telephone repair service to check if the local lines are truly busy or out of service. If lines are continously busy, but not out of service, notify the Multics Computer Center.
3. HOST DOWN	Indicates the Multics system is not in operation. Wait and try later or contact the Multics Computer Center.
4. ERROR ON PORTn	System is operational but a specific port or channel is not answering. Notify the Multics Computer Center.

- To report trouble call the Multics Computer Center - (602)249-77 or HVN 341-7567

TYMNET TELEPHONE NUMBERS

ALABAMA	
Birmingham	205/942-4141
ARIZONA	
Phoenix	602/249-9261
ARKANSAS	
Little Rock	501/372-5780
CALIFORNIA	
Alhambra	213/572-0999
El Segundo	213/640-1570
Los Angles	213/629-1561
Los Angles	213/683-0451
Mountain View	415/961-7970
Mountain View	415/941-8450
Newport Beach	714/540-9560
Oakland	415/465-7000
Oxnard	805/487-0482
Palo Alto	415/326-7015
Riverside	714/825-9372
Sacramento	916/441-6550
San Clemente	714/498-3130
San Diego	714/291-8700
San Francisco	415/391-9325
San Jose	408/984-5500
Santa Barbara	805/966-3184
Santa Rosa	707/526-2180
Van Nuys	213/986-9503
COLORADO	
Colorado Springs	303/471-9815
Denver	303/458-7921
CONNECTICUT	
Bridgeport	203/579-7820
Danbury	203/792-3060
Darien	203/655-8931
Hartford	203/568-2610
New Haven	203/787-5974
Waterbury	203/757-2537
DELAWARE	
Wilmington	302/658-5261
DISTRICT OF COLUMBIA	
Washington	703/841-0200
Washington	703/841-9560
FLORIDA	
Fort Lauderdale	305/467-7550
Jacksonville	904/721-8100
Miami	305/374-7120
Orlando	305/841-6850
Pensacola	904/434-5514
Tampa	813/229-0981
St. Petesburg	813/536-7823
W.Palm Beach	305/622-2871
GEORGIA	
Atlanta	404/659-6670
HAWAII	
Honolulu	808/521-7481
IDAHO	
Boise	208/343-4851
ILLINOIS	
Chicago	312/346-4961
Chicago	312/368-4607
Freeport	815/232-2181 (30 cps)
Freeport	815/233-2186 (10 cps)
Rockford	815/398-6090
INDIANA	
Fort Wayne	219/424-5162
Indianapolis	317/257-3461
Marion	317/662-0091
Southbend	219/259-9941
IOWA	
Cedar Rapids	319/364-3371
Des Moines	515/280-9600
Iowa city	319/351-4046
KANSAS	
Topeka	913/233-1612
Wichita	316/265-1241
KENTUCKY	
Louisville	502/361-3881
LOUISIANA	
Baton Rouge	504/927-6400
Lafayette	318/235-5202

TYMNET TELEPHONE NUMBERS

New Orleans	504/586-1071
MARYLAND	
Annapolis	301/268-9290
Baltimore	301/547-8100
MASSACHUSETTS	
Boston	617/964-3925
Boston	617/964-3900
Cambridge	617/491-5476
MICHIGAN	
Ann Arbor	313/665-2627
Detroit	313/963-3388
Jackson	517/784-8522
Kalamazoo	616/385-3150
Southfield	313/355-2950
St. Joseph	616/429-2568
INNESOTA	
Minneapolis	612/854-6659
ISSOURI	
Kansas City	816/753-6620
St. Louis	314/421-5110
EBRASKA	
Omaha	402/329-2970
EVADA	
Carson City	702/882-7810
Las Vegas	702/386-1899
EW HAMPSHIRE	
Nashua	603/888-3354
EW JERSEY	
Englewood Cliffs	201/894-8250
Moorestown	609/235-3761
Piscataway	201/981-0370
Princeton	609/452-8228
Wayne	201/785-4480
Union	201/964-6300
EW YORK	
Albany	518/463-3111
Buffalo	716/856-1400
Corning	607/962-5071

Huntington, L.I.	516/673-5780
New York City	212/350-9100
New York City	212/551-9322
New York City	212/344-7445
Niagra Falls	716/285-9354
Rochester	716/546-1410
Syracuse	315/437-7111
White Plains	914/761-8449
NORTH CAROLINA	
Charlotte	704/376-0320
Durham	919/549-0441
OHIO	
Akron	216/535-1861 (30 cps)
Cincinnati	513/242-7040
Cleveland	216/781-7050
Columbus	614/421-7270
Dayton	513/223-3847
Toledo	419/243-3144
OKLAHOMA	
Oklahoma City	405/947-0561
Tulsa	918/492-5306
Tulsa	918/492-1687
Tulsa	918/663-2220
OREGON	
Portland	503/224-0750
PENNSYLVANIA	
Allentown	215/433-6131
Erie	814/725-8671
Erie	814/454-6467
Harrisburg	717/236-1190
Philadelphia	215/561-6120
Pittsburgh	412/765-1320
Valley Forge	215/666-9190
York	717/846-4802
RHODE ISLAND	
Providence	401/351-2920
SOUTH CAROLINA	
Greenville	803/271-2418
TENNESSEE	
Memphis	901/345-1111

TYMNET TELEPHONE NUMBERS

TEXAS

Austin	512/444-3280
Baytown	713/427-1123
Beaumont	713/832-2589
Dallas	214/638-5800
El Paso	915/544-9590
Ft. Worth	214/263-2341
Houston	713/785-4420
Houston	713/785-4411
Houston	713/780-7390
Longview	214/758-0801
Lubbock	806/762-2402
Midland	915/683-5645
Odessa	915/563-0273
San Antonio	512/734-7381

UTAH

Salt Lake City	801/582-8972

VIRGINIA

Norfolk	804/622-0435
Richmond	804/649-3050

WASHINGTON

Seattle	206/622-7930

WISCONSIN

Madison	608/221-4211
Milwaukee	414/257-3482
Oshkosh	414/235-4594

BELGIUM

Brussels

Contact E. Godhaird
233-3700

CANADA

Calgary	403/263-7301 (30 cps)
Calgary	403/263-2006 (14.8 cps)
Calgary	403/263-2072 (10 cps)
Edmonton	403/423-4888
Montreal	514/878-0584 (30 cps)
Montreal	515/878-0589 (14.8 cps)
Montreal	514/878-0588 (10 cps)
Ottawa	613/563-9841 (30 cps)
Toronto	416/863-6202
Vancouver	604/688-9811 (30 cps)
Vancouver	604/688-4838 (14.8 cps)
Vancouver	604/688-4338 (10 cps)

ENGLAND

Contact Allen Jenkins at 01-606-4671 or 01-432-5573 for further details to use the Data Base Service of the British Post Office.

FRANCE

PARIS

Contact Mr. Feuvrier
DTRI
246 RUE DE BERCY
PARIS CEDEX 12 75584
(All France is "Toll Free" to TYMNET Nodes)
346-12-55 EXT. 4331
TELEX: 670372

GERMANY

FRANKFURT

Contact Mr. Helmuth Wolf
(611) .211.33.20

HOLLAND

AMSTERDAM

Contact Mr. V. Moorsel
070-753652

MEXICO

TYMNET TELEPHONE NUMBERS

MEXICO CITY
Contact Mr.Rafael Rivera
of CONACYT
905/524-7365

NETHERLANDS
The Hague 46-97-61

PUERTO RICO
SAN JUAN
Contact Mr. Ed Lukas
724-3989 .

SPAIN
MADRID
Contact Mr. Jaime Carvana
248-8531

SWITZERLAND
BERNE
Contact Mr. Urs Loosli
022-28-7117
(All Switzerland is
"Toll Free" to TYMNET Nodes

This page has intentionally

been left blank

APPENDIX H

ABBREV EXAMPLES

This page has intentionally
been left blank.

ABBREV EXAMPLES

CRF	"-ds CRF_cabinet"
b :pwl	"cwl &1; twl &1; pwl &1; dl &1.wl"
:wta_ad	"tape_ansi_ 1 -create -name sample -number 1 -format db -expires 12/31/99 -density 800 -ring"
Dirs	([dirs **])
Segs	([segs **])
GRT	-ctl "p^a ^a^a v^.3f r^.3f m^.3f b^d d^d $^.2f^/" -date -hour -min -vt -rt -mut -pgt -$t
cons	-dev user_i/o -dim syn
FOREDW	-ds "FOR Wallman" -he "AT PCO"
FORDAK	-ds "KAYDEN: CISL" -he "Cambridge MA"
FORRRR	-ds "STA-D DVP" -he "RON RIEDESEL"
-dtm	-dtcm -sort dtcm
-dtu	-dtu -sort dtu
-master	-in 15 -dv dtc300s -hyph
bob	Alvarado
E	[home_dir]>exl
JUNK	[pd]>junk_
ME	[user name]
-wd	[wd]
-acx	ac x ([segs **.archive])
am	accept_messages; memo -on
DEFER	am -hold -call "do ""ec >udd>m>lls>ec>mess_handler defer &f1""";gr -set -call "print_messages"
no	answer no -brief
rest	answer rest
yes	answer yes -brief

ABBREV EXAMPLES

```
b xpl1          asr >exl>o -before >sss

  :ale          asr    >udd>Demo>dbm_test>linus>executable    -after
                working_dir

b sysup         calc "(page_fault_histogram|pfh.last_pf_time)-
                                     (disk_traffic_data|2,d)/6000000"

b ivt           calc "(tc_data|400+apte.virtual_cpu_time)/60000000"

b unmask        call hcs_$set_ips_mask -o -2 -r -o 0

  athvn         can be reached at 703-790-3213, hvn 231

b cl            cf -a; rl -a

b cispn         check_info_segs   -pn   >doc>iml_info>**.info   -pn
                >doc>info>**.info   -pn   >udd>m>lib>info>**.info   -pn
                >exl>info>**.info

  ro            ro; am -pr -bf; rdy

  mnth          comp  monthly  -in  10  -of  [month_name].report  -pm
                [month_name] -pass 2

b rtape         copy_file   ids  [string  tape_ibm_   [response  "tape
                name:"]  -nb  [response  "file  number on  tape:"]  -bk
                [response   "block   length:"]  -rec   [response   "rec
                length:"] -fmt  fb -mode [response  "ascii or ebcdic:"]
                -den [response "density:"] -no_labels -retain all] -ods
                [string   record_stream_   -target   vfile_   [response
                "segment name:"]]

b wtape         copy_file   -ods  [string  tape_ibm_   [response  "tape
                name:"]  -nb  [response  "file  number on  tape:"]  -bk
                [response   "block   length."]  -rec   [response   "rec
                length:"] -fmt  fb -ring -cr -mode  [response "ascii or
                ebcdic:"]  -den  1600   -no_labels  -retain  all]  -ids
                [string   record_stream_   -target   vfile_   [response
                "segment name:"]]

b cpa           cpa -minlines 1  minchars 2

  U             cwd <

b LSOLD         cwd >udd>m>tac>s;in ls;tmr ls;dn ls ln lt;an working_ls
                ls ln lt

b return        cwd [value old_dir]; pwd
```

ABBREV EXAMPLES

```
b cwdd           cwd; cwd

b tapelist       cwd;fo tape_list;tapac tl cdt;ro;dp -dl tape_list

b h              cwd;pwd

b /status        date_time;ioa_  [string "USER  ID          " [user name]
                 [user  project]];ioa_$nnl  "TERMINAL  NUMBER    ";user
                 device_channel;ioa_ [string  [system n_users] USERS  ON
                 SYSTEM]

b :dd            dd -force

b dm             defer_messages; memo -off

b NUGAMES        df p>new_games -bf;fo p>new_games;do "ioa_ &1;lsd &1 -a
                 -sort"  ([contents  p>gamelist]);ro;cpa  p>master_games
                 p>new_games;dl       p>master_games;rn       p>new_games
                 master_games

b DIE            dl [pd]>stack_4

  later          dm;a  -hold -call do ""sms [lms] Messages Deferred."""

b mftx           do       "answer yes -bf move test>&1.runout save>=="

  ,inf           do          "comp   &1.info   -of   &1.info  -gl;an
                 &1.info.compout &1.info; dn &1.info.compout"

  crf            do       "ioa_   &1;cpa  save>&1  test>&1  5 1" ([segs
                 test>**])

  ,pf            do " answer no  -bf cr &1.p; tc  &1.p; fo &1.p; profile
                 &1; ro"

  :rm            do " rdn; gr -control  ""^/Yes Master??^/""  -set -call
                 pm"

  vip            do     "      set_tty       -ttp      vip7200       -modes
                 echoplex,1180,^ctl_char,polite,replay,&f1"

  do*            do "&1  ([segs &2])"

  *              do "&1 [directory &2]>([files &2]) &f3"

  please         do "&qf1"

  cpmsf          do "(cd  cwd)  [entry  &1];cp -acl  &1>**  ==;cwd <;sbc
                 [entry &1] &2"

  finde          do ". call hcs_$make_ptr -1¦1 &1 &2 -r -1¦1 -r -o 0"
```

ABBREV EXAMPLES

```
b lst_          do ". call  lib_sort_tree_$by_primary_name -p &1 -p  &2
                -p &3 -p &4"

b cr            do ". create &1; if arg &2 -then ""addname &f1"""

b rf            do ". dm;rf &f1;ioa_ ^3/Done.;am -short -print"

b pcd           do ". if arg &1  -then "". pcd  &f1"" -else "". pcd mem
                cpu page"""

b macabs1       do ". if  argeq true  [exists segment  &1.absout] -then
                ""truncate  &1.absout"";  ear &1 -rt -q  1 -ag [spe &1]
                [suffix &1]"

b find          do ". in &1; lrn &1"

b sval          do ".  ioa_  ""^a: '^a'  changed  to '^a'""  &1 [string
                [value &1]] &r2; value$set &1 &r2"

b valu          do ". ioa_ [string [value &1]]"

b gpn           do ". ioa_ ^a [gpn ""&1"" &2]"

b listprog      do ". list -nm -all -bf -sort nm &1.**"

b IPSstaff      do ". memo -al -time  ""1245. &1"" &2 [substr [day_name
                &1] 1 3]"" 1300: IPS Staff Meeting"""

b run           do ".  on  cleanup im  -brief  ""dm;  runoff  &rf1; im;
                ioa_$nnl """

b :def          do ".abf :rd do "":r &1"""

b :4            do ".af ::comp &1"

b :2            do ".af ::pl1 &1"

b WEEKLY        do  ".1";ioa_  ^¦;ws  >udd>m>lls "ls  -all   -sort  dtu
                -mode   -length   -name   -dtem  -dtu";do   "ioa_  ^¦;pr
                >udd>m>lls>ec>&1" ([segs >udd>m>lls>ec>**])

b PLI           do ":2  [spe &1  pl1];LK &1  list pl1;UL  &1 pl1;pl1 &1
                -map &f2;:3 [spe &1 pl1].list;SEVERITY pl1 &1"

b COMP          do ":4 [spe  &1 compin];LK  &1 compout  compin;comp &f1
                -of;:3 [spe &1 compin].compout"

b :cispn        do ":lf 4; ws &1 ""cis -pn  [wd]>** -dt [default [value
                doc_changes_date] &2] &3 &4 &5 &6 &7 &8 &9"""
```

ABBREV EXAMPLES

```
b :doud          do ":par &1;ear [home_dir]>&1 -tm [response ""What
                 time?""] -ag do [wd] &fr2"

b zap            do ">udd>m>ab>value$set dir [wd] -cm ""Set by the zap
                 abbrev"";new_proc"

b ABS            do "ABSUT1 &1 &r3;ear ME>&1 -rt -q [default "2" &4] -of
                 pooldir>&1.absout -bf -ag &1 &2 [fl ""^a"" &5 &&]
                 &r3;lar -position -q [default "2" &4]"
                 [>udd>m>rab>new>abs_date] [wd]

  URAT           do "C [sysn] / [sysm]"

b MORE           do "HEY ::sm &rf1"

b REPLY          do "HEY [lms] &rf1"

b CPW            do "ac xf &1 [pd]>[entry &2];if is [pd]>[entry &2]
                 -then ""compare [pd]>[entry &2] &2 &3 &4;dl [pd]>[entry
                 &2]"""

b pushname       do "add_name &1 &!; delete_name [directory &1]>([name
                 &1 2]); rename [directory &1]>[name &1 1 1] &2;
                 add_name [directory &1]>&2 [name &1]; delete_name
                 [directory &1]>&!"

b sm             do "am -print;send_message &f1"

  ckm            do "answer no mail >udd>&2>&1>&1"

b no             do "answer no -bf &f1"

  =pr_cal        do "calendar &1/01 -fw holidays birthdays"

b CD             do "cd &1;SIS &1;SID &1"

b :cmpsr         do "comp psr.[substr &1 1 3] &f2 -pm ""&1, 1978"" "
                 [month_name [response "Month No? (1-12)"]/1]

b :cmpmsf        do "compare &2>&1 &3>&1" ([index_set 0 [response "Last
                 comp_no?"]])

b cpit           do "cpa &1 <(old new)>=="

  =cd            do "create_dir &1;(sis sid) &1 (rew sma) [user
                 name];(sis sid) &1 (re s) *;da &1 [user name].[user
                 project];sa &1 (sma s) ([user name].[user project] *)"

  typos          do "create_wordlist &1;trim_wordlist &1.wl;
                 print_wordlist &1.wl"
```

ABBREV EXAMPLES

```
b yet,           do  "cwd;if  [less [value  Time]  [time]]  -then ""ioa_
                 YES!!"" -else ""ioa_ No.^x[value Time]"";cwd &1" [wd]

b cig,           do "cwd;value$set Time [time 30 min];cwd &1" [wd]

b setre          do "da &1 ..;sa &1 re *.*.*; da &1 *.SysDaemon.*"

  =dla           do "dl &1.(absin absout)"

b REFRESH        do "dl  [pd]>[entry  &1] -bf;mv  &1  [pd]>[entry &1];mv
                 [pd]>[entry &1] &1"

b qxab           do "dl  [pd]>abr -bf;fo  [pd]>abr;if arg  &2 -then ""do
                 """".u  &2"""""""";do  "".l  &q1"";do  "".u"";ro;if isnzf
                 [pd]>abr -then  ""qxr [pd]>abr  1s/^/e.a/ s/^e.ab*/&&f/
                 1p"""'

b comp           do "dm; compose &f1; am"

b :rfdtm         do "dm; memo -off;rf &1 -pm [string [year] [month_name]
                 [day] at [time]] &f2;im;memo -on"

b filexist       do "do ""if  [exists segment &&1]  -then -else """"ioa_
                 &&1"'""""" ([contents &1])"

b :dpnt          do  'dp -ds  ""MAY,  CABINET""  -he  [string  [default
                 [upper_case  [entry &1]]  &r2]] -q  [default 3 &3] -ned
                 &f4 &1"

b dofa           do "fa &1 &2 ""push_wdir  [if [ngreater [index  [status
                 -tp &1] directory] 0 ]  -then &1 -else [directory &1]];
                 &3 &rf4; pop_wdir"""

b ADDTIME        do "fo &1.list;ioa_ ^|^3/;pl1$times;ro"

b fout           do "fo &1;io detach error_output;io attach error_output
                 syn_ user_output"

  =dsrc          do "fo =cp>+src.&1.list;ds  [=gtt>aft *src] -ch;ro;ioa_
                 ""+src.&1.list Created."""

b ,,log          do "fo    >udd>m>ejw>[month_name].log_time;    ioa_$nnl
                 [date_time]; ro"

b CALENDAR       do     "fo     CPS>calendar;calendar      [date]      -fw
                 HDS>p>holidays;do    ""calendar  """"&&1  month""""  -fw
                 HDS>p>holidays""  ([index_set 12]);ro;DPDL CPS>calendar
                 &f1"
```

ABBREV EXAMPLES

```
b ec            do "if [and [not [exists segment &1.ec]] [exists
                segment &1.absin]] -then ""an &1.absin &1.ec;exec_com
                &f1"" -else ""exec_com &f1"""

b usage         do "if [equal &n 0] -then ""ps usage: Usage is:
                usage"" -else ""ps usage: Wrong number of arguments
                supplied. For usage instructions, type
                """"usage"""""""

b rl            do "if [exists argument &1] -then ""release -all""
                -else ""release"""

b check         do "if [greater [status &1.pl1 -dtcm] [status &1
                -dtcm]] -then ""ps &1 has been modified"""

b fini          do "if [have_mail] -then ""mail &rf1"" -else ""do
                >udd>m>vv>basement>say;logout &rf1"""

b ansrvxxx      do "if [or [equal [format_line ^a
                [last_message_sender]] [user name]] [equal [string
                [substr [last_message] 1 10]] Acknowledg]] -then -else
                ""sm s [last_message_sender] """"(Answering
                service):"""" &f1"""

b TALK          do "am;sm &1 &2;dm"

b ARDIND        do "ind &1 -lm 3 -in 2 &f2"

b :indip        do "ind &1.incl; if [query Bad?] -then ""do """":r
                [pd]>&1.incl.ind"""""""

b deo           do "io attach junk discard_ io open junk so;io detach
                error_output;io attach error_output syn_ junk;deomess"

b reo           do "io close junk;io detach (junk error_output);io
                attach error_output syn_ user_i/o;reomess"

b syn           do "io_call attach &1 syn_ &2"

b e             do "ioa_ ""NOT TED""; do ""&f1 &rf2"""

b GONE          do "ioa_ ""I WILL BE AWAY 'TIL &1 - LOG ME OUT IF YOU
                NEED THE TERMINAL"" "

  asd           do "link (&f1) [home_dir]>hh>=="

b lsw           do "list [where &1] &rf2"

b first         do "ls -nhe -first [default 1 &1]"

b prmail        do "mail >udd>&2>&1>&1 &f3"
```

ABBREV EXAMPLES

```
b bday          do "memo -alarm -repeat  12months -date ""&f3"" &1 &2's
                birthday!"

b :pl1          co "pl1 &1 -table -symbols &f2;ioa_ Done."

b :lo           co  "pmotd; if  [have_mail] -then  ""if  [query  MAIL?]
                -then """"mail"""""""; logout &1 &2"

b p             do "pr &1 1 9"

b prf           do "print &1.runout 1 "

b :dpx          do  "px  &1  -sm   :cp>[entry   &1].px; dp  -ds  ""MAY,
                CABINET"" -he   [string  [default   [upper_case  [entry
                &1].px] &r2]] -q [default 3 &3] -ned -dl &f4 :cp>[entry
                &1].px"

b :pxa          do "px &1!** -sm  :cp>[entry &1].archive.px &f2"

b =exc          do "rn &1 !!!!;rn &2 &1;rn !!!! &2"

b vip           do        "set_tty       -ttp       vip7200        -modes
                echoplex,ll80,polite,replay,&f1"

b reply         do "sm [nmf_last_sender] &f1"

b w             downlog memo -al -tm [minus [system next_down_time] .1]
                -call LOGOUT//

b ddl           dp -dl

  dpl           dp -dl [segs **.list]

b lv            general_ready -control ^a^/ -level;

b RDN           gr -string [substr [string  [system installation_id]] 1
                1] -date_time -level -set -call pm;dm;memo -on;rdn

b linus?        help -pn >udd>Demo>dbmt>x>doc>(linus linus_pre_release)

b IOCD          io (close detach)

b speak         io (close  detach)  user_output;io attach  user_output
                syn_ user_i/o

  =edl          io  attach  efo_   discard_;io  detach  error_output;io
                attach error_output syn_ efo_

  =erro3        io_call        attach          error_output          vfile_
                [home_dir]>error.msgs
```

ABBREV EXAMPLES

```
b :open        io_call open file0(3 4 5 9); io_call position file0(3 4
               5 9) bof

b mark         ioa_ "        0       .       1       .       2       .       3       .       4
               .       5       .       6       .       7       .       8       .       9       .
               0"

b =us          ioa_  "(users ^a:^a)  (units  ^a:^a)"  [system n_users]
               [system max_users] [system n_units] [system max_units]

b down?        ioa_     "Scheduled    shutdown:     ^a    ^a  "    [system
               next_down_date] [system next_down_time]

b lkcl         ioa_    "^5/From:^-^a.^a^/To:^-Whom It  May   Concern^/"
               [user name]  [user  project];  lkc -time  60 -gr 3 -spw
               [value pass]

b WAIT         lkc -gr 5 -nol

  newu         logout -hold

  Dear         mail *

b mail?        mail -brief

b ti745        stty -delay 0,0,0,0,0,0

b tn300        stty -modes tabs,ll118,replay,polite;tab_set

  ROSY         stty -ttp rosy -modes ^echoplex,ll130

  diablo       stty  -ttp   tn300  -modes    tabs,ll130,vertsp  -delay
               0,.18,0,0,0,59

  whoa         who -absentee

  vpi?         who .VPI1 .VPI2 .VPI3 .VPI3 .VPI4 .VPI5

  liz?         who LMullen

  PUBLIC       ws [hd] "sa  ([segs *]) re  *.*.*;sa  ([segs *.*.**]) r
               *.*.*;sa ([dirs **]) s  *.*.*;sid [wd] s *.*.*;sis [wd]
               re *.*.*"

  PRIVATE      ws [wd] "da ([segs **] [dirs **]) *.*.*;did ([dirs **])
               *.*.*;dis ([dirs **]) *.*.*"
```

This page has intentionally
been left blank

APPENDIX W

WORKSHOPS

This page has intentionally
been left blank.

WORKSHOP ONE

NOTE: Because these workshops build upon one-another, you should follow the instructions precisely, using the names indicated.

Throughout these workshops, the Person_id "Student_??" is intended to mean your own Person_id.

1. Refer to the section "Access Sequence" in topic 2, and log in to Multics.

2. Exercise the delete character (#) and the delete line (@) symbols. For example, type the following how_many_user commands:

```
hmu

hnu##mu

jmu@hmj    T###u TSmith
```

3. Type the following commands and observe the results. Supply the answer "yes", "no", or "rest" when asked.

```
help hmu

help sked

help help
```

4. Enter the accept_messages mode by typing "am". Note that the accept_messages command created a mailbox for you. This mailbox is now a permanent part of your file space. You can now receive messages.

   List the names of all segments (files) that belong to you (ls). Note that your mailbox is the only segment you have.

5. Type the following sequence of who commands and observe the results of each form.

```
who

who .F01

who .F01 -long

who Student_04
```

These are the names and projects of users who are currently logged in. Note your own "name".

6. Select two or three F01 users who are currently logged in and send them messages. For example, the following commands will send messages to Student_04 and Student_09:

```
sm Student_04.F01  Does this really work?

sm Student_09.F01  Great class, isn't it?
```

7. Send yourself a message.

```
sm Student_??.F01 From me to you!
```

8. Enter the defer_messages mode (dm) and again send yourself a message. Note that you did not receive the message as you did before. The message is in your mailbox and will remain there until you print_messages. Print the message(s) in your mailbox (pm).

9. Again, send yourself a message. As before the message was placed in your mailbox because you are still in the defer_messages mode. Return to the accept_messages mode (am). You will now receive messages immediately, as they are sent. Note that the accept_messages command did not print messages currently in your mailbox. Print the message(s) in your mailbox.

10. Determine the current system configuration by typing the following print_configuration_deck commands:

```
pcd cpu mem

pcd
```

11. Type the following sequence of commands:

```
date

date_time

long_date

time

minute

hour

day

day_name

month

month_name

year
```

While available in this form, these commands will take on more significance when the topic of "Active Functions" is discussed.

12. Generate your resource usage report for this billing period by typing the resource_usage command (ru).

Note: Your current login session may not be reflected, however, if other F01 classes have been conducted this month, you may see considerable usage.

3. Log out (logout) and return to the classroom.

This page has intentionally

been left blank.

WORKSHOP TWO

1. Log into Multics. Accept messages (am) and print messages (pm) if desired. List the names of all segments (files) that belong to you (ls). Note that you still have only one segment, your mailbox.

2. Invoke the qedx text editor (qx). Do not expect a ready_message as you are now in the qedx subsystem.

3. You are in the edit mode of qedx and have an empty buffer (a scratch-pad). Enter the append mode (a) and type the following three lines exactly as you see them:

```
In anything att all, perfection es attained
when there is no longer anything to take away.
1,2p
```

4. Return to the edit mode (\f) and print the contents of your buffer (1,$p).

5. Correct the intentional typing errors, as well as any you may have made. For example, to change "es" in the first line to "is", type the sequence:

```
1p         (positions you to the first line and prints it)
s/es/is/   (substitutes all "es" strings for "is")
p          (prints the current line)
```

   To remove the last line, type the following:

```
$p         (positions you to the last line and prints it)
d          (deletes the current line)
```

6. After correcting all of the errors, write the contents of your buffer to a segment by the name of seg_1 (w seg_1).

7. Quit the editor (q) and note the ready message. You are now back at command level. Your qedx buffer has been destroyed. (Don't confuse your qedx buffer with the permanent storage system segment seg_1)

8. List the names of all segments that belong to you (ls). Note that seg_1 was created for you by qedx's "w" request.

TRENDATA
for \ = ESCAPE 'r'

9. Invoke qedx. You again have an empty buffer. (Confirm this by printing the contents of your buffer.) Read the contents of seg_1 into your buffer (r seg_1) and again print the buffer's contents.

10. Append the following text between the first and second line of your buffer by typing the following requests:

```
1p
a
not when there is no longer anything to add, but
```

escape 'r'

11. Return to the edit mode ~~(\f)~~ and print the contents of your buffer. Correct any typing errors.

12. Write the contents of your buffer to seg_1 again. This write will replace the old contents of seg_1 with the contents of your buffer.

13. Print the contents of your buffer. Note that the above write did not affect the buffer's contents. Write the contents of your buffer to seg_2 (just for fun). The contents of seg_1 and seg_2 are now identical.

WORKSHOP 2 CONTINUED ON NEXT PAGE

14. Type the following sequence of qedx requests in exactly the order as shown:

```
1,$s/e/zzz&/
1,$p
e sm Student_??.F01 Can I do this within qedx?
1,$s/zzz//
1,$p
w                       (To which segment did you write?)
/is/
/is/
/is/
/is/                    (Notice the wrap-around)
1,$d
1,$p                    (Why is your buffer empty?)
r seg_1
r seg_1                 (Remember the default address here?)
w seg_1
r seg_2
1,$p
w                       (Why has qedx forgotten the pathname?)
5,$d
1r seg_1                (We're forcing the read address)
1,$p
```

15. Quit the editor. Note that qedx allowed you to quit even though you had not written out your modified buffer (Hsssssss!).

16. Print seg_1 (pr seg_1). It's contents should be:

seg_1

```
In anything at all, perfection is attained
not when there is no longer anything to add, but
when there is no longer anything to take away.
In anything at all, perfection is attained
not when there is no longer anything to add, but
when there is no longer anything to take away.
```

17. List the names of your segments and log out.

WORKSHOP THREE

1. Log in. Accept and print messages if desired. Execute the print_wdir command (pwd). This is your home directory and is currently your working directory.

2. List the names of the segments in your working directory.

3. Print the contents of your seg_1 segment (pr seg_1).

4. Create a segment by the name of Prince (cr Prince). List again the names of the segments in your working directory. Note that Prince has a length of zero records (i.e.: it is empty)

5. List names of all entries in your working directory (ls -all). Note that you have no subordinate directories.

6. Create a subordinate directory by the name of Programs (cd Programs). List again the names of all entries in your working directory.

7. Change your working directory to Programs (cwd Programs). Verify the change by printing your working directory.

8. List the names of the segments in your working directory. Note, of course, that your Programs directory is empty.

9. Without changing your working directory, print the contents of your seg_1 segment located in your home directory (pr >udd>F01>Student_??>seg_1). What would (pr seg_1) have done? Try it.

10. Return to your home directory (cwd). Why is no pathname required for the cwd command in this case? Verify that you are back in your home directory.

11. Change your working directory to Student_01's home directory (cwd >udd>F01>Student_01) and verify this change by printing your working directory. What would the command "cwd Student_01" have done? If you're not sure, return to your home directory and try it, but remember to change back to Student_01's home directory before continuing the workshop.

12. List the names of the segments in your working directory. Understand that this is Student_01's home directory. You are able to list the names of his segments only because he has granted you access to do so. Note that you have read permissions on many of Student_01's segments.

13. Print the contents of Student_01's treasure_hunt segment and obey all instructions specified. You are able to print the contents of this segment only because Student_01 has granted you access to do so.

14. Change your working directory to the system's root directory (cwd >). Verify by printing your working directory. List all directories under the root (ls -d). These are the system level directories. Note that they are primarily system libraries. Feel free to explore -- time permitting.

15. Return to your home directory and execute the following commands:

```
     cr X                                              (create)
     ls X                                              (list)
     dl X                                              (delete)
     ls X                                              (list)
     dl >udd>F01>Student_01>treasure_hunt              (delete)
```

Why were you unable to delete Student_01's treasure_hunt segment?

```
     cd X                                              (create_dir)
     ls X                                              (list)
```

Recall that by default, the list command only deals with segments.

```
     ls X -d                                           (list)
     dd X                                              (delete_dir)
```

16. Add the names s_1, s1 and s1.compin to your seg_1 segment (an seg_1 s_1 s1 s1.compin). List the contents of your working directory. Notice how the alternate names are listed.

17. Print the contents of seg_1 using one of the alternate names (pr s1). Get the status of seg_1 using another alternate name (st s_1). Compose the contents of seg_1 one using the name s1.compin (comp s1.compin). Note the full-page formatting done by the compose command.

18. Delete the segment seg_2 (dl seg_2). List the contents of your working directory. Notice the change in the segment count. Attempt to print seg_2.

19. Delete the names seg_1, s_1, s1.compin (dn seg_1 s_1 s1.compin). List the contents of your working directory. Notice that the segment count is unchanged.

20. Rename the segment s1 to seg_1 (rn s1 seg_1). List the segment seg_1 (ls seg_1) and observe the add names.

21. Rename seg_1 to Prince (rn seg_1 Prince). Answer "yes" to delete the original Prince segment. (Recall that it is empty and is of no practical use to you.)

22. Attempt to delete the name Prince (dn Prince). Since this is the only name on the segment, the command will inform you that deleting the only name on a segment is not allowed.

23. Copy the segment >udd>F01>Student_01>alphabet into your working directory (copy >udd>F01>Student_01>alphabet). List the contents of your working directory and print the contents of this copied segment.

    Why does your copy also have the name alphabet? What would the copy command look like if alphabet was to be copied and called my_alphabet?

WORKSHOP FOUR

1. Log in. Accept and print messages if desired. Print your working directory.

2. Change your working directory to >udd>F01>Student_01. Execute the following commands:

   ~~ls~~
   ~~ls~~ seg_1 alphabet
   ~~ls~~ *
   ~~ls~~ s*.*
   ~~ls~~ s*.**
   ~~ls~~ *.*
   ~~ls~~ *.* -exclude s*.*

3. Return to your home directory and execute the following sequence of commands: (Note: If the printer queues are near empty, you may not be able to cancel some of your dprint requests)

```
ldr                               (list_daemon_requests)
dp Prince                         (dprint)
dp -q 4 Prince                    (dprint)
ldr                               (list_daemon_requests)
ldr -all                          (list_daemon_requests)
cdr -q 4 Prince                   (cancel_daemon_request)
ldr -all                          (list_deamon_requests)
cdr Prince                        (cancel_daemon_request)
ldr                               (list_daemon_requests)
```

   Note how some of these commands reported the total number of requests (system wide) as well as your total number, but only for the specified queue.

~~4. If there~~ is an on-line printer accessible to you, dprint (for keeps) the contents of your Prince segment (dp Prince). Do you recall what the default heading and destination banners will be? Remember to pick up this output sometime before class tomorrow morning.

5. Enter the abbrev mode (ab) and type some command, such as pwd. Note that a profile segment was created for you. List the contents of your home directory and note this new segment.

6. Type a period (.). The reply is from the abbrev processor confirming the fact that you are in abbrev mode. Quit the abbrev mode (.q) and again type a period the see what the command processor will do with it. Re-enter the abbrev mode.

7. Create an abbreviation for your User_id by typing a command line similar to the following:

```
.a ME Student_??.F01
```

8. Check your abbreviation via the show request (.s ME). Recall the abbrev break characters and observe how they function by typing the following lines:

```
.s Who is ME?
.s Who is ME.
.s Who is MEAN?
.s Who is ME.the_great?
.s Who is ME_the_great?
```

9. Use your abbreviation to send yourself a message. For example:

```
sm ME Does this work?
```

10. Define abbreviations for those users with whom you frequently communicate. For example:

```
.a S9 Student_09.F01
.a S4 Student_04.F01
```

11. List all of your abbreviations (.l). List one of your abbreviations (.l ME). List all of your abbreviations that start with "S" (.la S).

12. Enter the qedx text editor and write the following text to a segment by the name "start_up.ec".

start_up.ec

```
accept_messages
print_messages
abbrev
```

WORKSHOP FOUR

3. Quit the text editor and type the following command and observe the results:

```
ec start_up.ec
```

If you received any error messages, invoke the qedx editor, read the segment start_up.ec, correct your typing errors, rewrite your corrected copy, and try again.

Remember: From now on this start_up.ec will be invoked for you (automatically) whenever you log in.

4. Using qedx, create another segment named A.ec containing the following lines:

A.ec

```
ec start_up.ec
time
ec start_up.ec
```

Quit the text editor and type the following command and observe the results:

```
ec A.ec
```

5. Execute the print_motd command (pmotd). Note that a Person_id.motd segment was created for you (in your home directory). It is through this segment that the system knows what messages you have already seen.

6. Again, execute the print_motd command. Since you have already seen the message of the day, the command will suppress additional printings.

7. Using the qedx text editor, add the print_motd command to your start_up.ec.

WORKSHOP FIVE

1. Using the qedx editor, input the following PL/1 source and write it to the segment "add.pl1":

add.pl1

```
add: proc;
/*  Written by Student_??  */

dcl (sysin, sysprint) file;
dcl (I, Sum) fixed bin init (0);

do while (I >= 0);
put skip list ("value? ");
get list (I);
Sum = Sum +I;
put skip list ("Sum is:", Sum);
end;
put skip list ("Fini");
put skip;
end add;
```

2. Indent the PL/1 source (ind add.pl1). Correct any errors found by the indent command and try again. Print the PL/1 source and note the indentation that's been done for you.

3. Compile add.pl1 (pl1 add.pl1). Correct any errors found by the pl1 command and try again.

4. List the contents of your working directory. Note the segment add.pl1 (created by the text editor) and add (created by the PL/1 compiler). add.pl1 is the source segment, and add is the object segment.

5. Print the contents of the object segment add. This will take a few minutes as the "unprintable" binary code is printed as an octal dump. Each \nnn represents three octal characters, which is 9 bits, which is one byte.

6. Execute the PL/1 object program add which is in the object segment add (add -or- add$add). Supply positive integer values between 0 and 9999 when prompted and observe the results.

7. Hit the break key and note the level number in the ready message. You are back at the command level -- free to do as you please. Execute some command such as pwd.

8. Type the start command (sr). The program add has been re-started at exactly the same point it was at when you hit the break key. In other words, the program add is again waiting on you to supply integer values. Key in a value and see if the Sum returned is consistent with the previous Sums.

9. Key in a negative value to stop the program add. Add has terminated normally. Note the level number in your ready message. (No level number denotes level one.)

10. Again compile your add.pl1 source, but use the -table control argument (pl1 add.pl1 -table). This will allow you to symbolically "probe" the object program.

11. Again execute your program add and supply several positive integers as before. Hit the break key. After receiving the ready message type the following command:

```
probe
```

You are now inside the probe debugging subsystem. Note that probe printed a small status report about the current state of your process. Now type the following sequence of probe requests:

```
stack
```

This request traces your stack beginning with the most recent frame. Note the frame number of add's stack frame. Who called add, and who did add call? Probe's "current" stack frame is the one in which the unexpected event occurred (i.e., frame 10 if you're in abbrev mode). Any request that you issue will be relative to this stack frame.

```
use add
```

This request instructs probe to make add's frame (i.e., frame 7 if you're in abbrev mode) your "current" stack frame.

```
where                      (Where are you, probe?)
source                     (Print the "current" source line)
value I                    (What's the value of I?)
value Sum                  (What's the value of Sum?)
let Sum = 500+Sum          (Change the value of Sum)
value Sum                  (What's the new value of Sum?)
quit                       (Quit probe)
```

12. You are back at command level. Note that your level number is unchanged. Type "start" and supply another positive integer for add. Does the new Sum reflect the change you made while in probe? Key in a negative value to stop the program add.

13. Enter qedx and read the segment >udd>F01>Student_01>long. Commence printing of the entire segment and then hit the break key. After receiving the ready message, type the start command and observe the results.

    While the printing continues, again hit the break key. After receiving the ready message, type the program_interrupt command (pi). You are again in qedx, but at request level. Verify by typing 1,5p. Quit the editor.

14. Print your current search rules (psr). These are the default search rules given to users at login.

15. Note: The following exercise will work properly only if you have not executed the who command during this login session. If you have executed the who command, type the command: "tmr who".

    Add the name who to your object segment add (an add who). Attempt to list the users currently logged in (who). Suffice it to say: Do not indiscriminately use command names for your program names.

    Remove this extra name from add (dn who), and check to see if who command works as usual. If it doesn't work - why doesn't it? (For a hint, type: where who)

16. Terminate (delete) the name who from your initiated segments (tmr who) and again attempt to execute the who command.

17. Change your search rules to include the directory >udd>F01>Student_01>tools after your working directory (asr >udd>F01>Student_01>tools -after working_dir). Print your current search rules.

18. Note: The following exercise will work properly only if you have not executed the hmu command during this login session. If you have executed the hmu command, type the command: "tmr hmu".

    Attempt to list the number of users currently logged in (hmu). Suffice it to say: Do not indiscriminately add unfamiliar directories to your search rules.

19. Log out. When you next log in, you will again be given the normal default search rules.

This page has intentionally

been left blank.

WORKSHOP SIX

1. Log in and type the mail command. Recall that the mail command will extract messages and mail. Use this command if and when you receive notification that you have mail.

2. If you have not received a message requesting partnership, choose an F01 user who is currently logged in and send him/her a message similar to the following:

```
sm Student_07.F01 May I be your partner?
```

3. Send your partner messages in the dialogue mode, as in the following example. Wait for a reply between lines.

```
sm Student_07.F01
How are you doing today?
That's good.  I'm not doing too bad myself.
See you later.
.
```

4. Mail your partner the contents of your add.pl1 segment. For example:

```
mail add.pl1 Student_07.F01
```

5. List the ACL of your add.pl1 segment (la add.pl1). Note: This ACL was given automatically by the system when you created add.pl1.

6. When your partner requests access to your add.pl1 segment (and he will), do the following: give your partner read access, verify by listing the ACL of your add.pl1 segment, and send a reply to your partner. For example:

```
sa add.pl1 r Student_07.*.*
la add.pl1
sm Student_07.F01 You now have access to add.pl1.
```

7. Attempt to print your partner's add.pl1 segment. For example:

```
pr >udd>F01>Student_07>add.pl1
```

8. Request read access to your partner's add.pl1 segment and attempt to print the segment again. (Keep trying until it works)

9. Note: Do not proceed until your partner has requested (and has been given) access to your add.pl1 segment.

   Remove your partner's name from your add.pl1 ACL. For example:

```
da add.pl1 Student_07.*.*
```

   Verify the change by again listing the ACL for add.pl1

10. List the ACL on your home directory (la). Who are these users? Attempt to set access for yourself on your home directory (sa -wd sma Student_??.*.*). Why can't you set access on your own home directory?

11. Execute the following commands:

```
ls -d
la Programs
copy alphabet Programs>A
ls Programs>**
```

   Note your access on segment A.

```
pr Programs>A
da Programs Student_??.F01.*
la Programs                   (What permissions are needed?)
cr Programs>B                 (What permissions are needed?)
ls Programs>**                (What permissions are needed?)
dl Programs>A                 (What permissions are needed?)
pr Programs>A                 (What permissions are needed?)
```

   Note that you can do nothing except print the contents of the segment A.

12. Type the following command sequence. (If necessary, include your time zone in the -time request. For example: -time "4pm est")

```
memo
memo Happiness is Multing the day away.
memo -time 4pm -alarm It's almost time to go home.
memo -list
memo
```

13. Modify your start_up.ec to include the mail and memo commands.

WORKSHOP SIX

14. Type the following command sequence.

```
    pat                      (print_attach_table)
    fo                       (file_output output_file)
    pat
    ro                       (revert_output)
    pr output_file
```

Notice the difference in the I/O switch attachments.

15. Before logging out, check your mailbox.

WORKSHOP SEVEN

1. Send yourself the following messages:

```
sm ME My working dir is [wd].

sm ME Last message sender was [lms]

sm ME The time is [time].

sm ME The date is [date] which is the same as [long_date].

sm ME [user name].[user project] login at [user login_time].

sm ME Current user load is [system n_users].

sm ME Multics: [system installation_id].

sm ME Multics: [system company].

sm ME The absolute pathname is [path add].

sm ME My one-component segments are [segs *].
```

2. Create the following abbreviation:

```
.ab reply sm [last_message_sender]
```

List all of your abbreviations.

3. Execute the following command lines:

```
.s reply Who are you?
reply Who are you?
```

Who in fact was the last_message_sender?

4. Create two segments "Names_A" and "Names_B" containing the following lines:

Names_A

```
A10
A20
A30
```

Names_B

```
B10
B20
B30
```

Now execute the following command lines and observe the results:

```
sm ME  [contents Names_A]

create [contents Names_A] [contents Names_B]

list A* B*

list  [contents Names_A]

dl  [contents Names_A]

list A* B*
```

5. Create an exec_com segment containing the following text:

query.ec

```
&command_line off
cwd [response "Working directory desired?"]
&print Your working directory is:
pwd
sm ME [response "What would you like to say to yourself?"]
```

The first line of the above exec_com wil] inhibit the printing (echoing) of the command lines when the exec_com is executed.

Now invoke this exec_com and provide legitimate answers to the questions asked. Invoke the exec_com again, but this time respond to the first question by simply hitting the line feed key. Do you understand why you are now back at your home directory?

6. Use qedx to create the following exec_com:

recur.ec

```
&command_line off
&print ENTRY:  &n arguments, first argument is &1
&if [ngreater &n 0] &then &goto START
&quit

&label START
&print Executing exec_com command
ec recur.ec &2 &3 &4 &5 &6 &7 &8
&print EXIT:  &n arguments, first argument is &1
&quit
```

Invoke the exec_com and observe the results by typing:

```
ec recur.ec A B C D E F G H
```

Change your working directory to your Programs directory and again invoke this exec_com by typing:

```
ec >udd>F01>Student_??>recur.ec A B C
```

Why does the exec_com fail now? Can you correct the problem? Try to correct the problem.

7. Return to your home directory and execute the following sequence of commands

```
.ab pl1_ex do "pr &1.pl1;pl1 &1.pl1; [entry &1]; logout"
.a F1 >udd>F01
.l
pl1_ex F1>Student_01>nothing
```

The entry active function returns the entryname of the supplied pathname. Why was the entry active function used? Was it necessary? Why were you logged out?

WORKSHOP EIGHT

1. Use command line iteration to send yourself several messages. For example, if you are Student_06 type the following line:

```
sm Student_0(6 6 6 6).F01  This is number (1 2 3 4)!
```

2. Recall the contents of your segments Names_A and Names_B and execute the following command lines:

```
sm ME  [contents Names_B]

sm ME ([contents Names_B])

list [contents Names_B]

rename ([contents Names_B]) ([contents Names_A])

list A* B*

create ([contents Names_A]).[date].[user name]

list A*.**

rename  A*.**  =.=

list A*.**
```

WORKSHOP EIGHT

3. The print command only allows one pathname argument such as "print add.pl1". Use command line iteration to print the segments Prince and alphabet as follows:

```
print (Prince alphabet)
```

Now execute the following command lines to create and test an abbreviation for the print command that allows it to accept up to five pathnames as arguments.

```
.ab :pr do "print (&1 &2 &3 &4 &5)"
:pr Prince alphabet >udd>F01>Student_01>nothing.pl1
```

What would happen if the abbrev had been defined as follows:

```
.ab print do "print (&1 &2 &3 &4 &5)"
```

Try it! When you get the error message, notice the level clause of your ready message. Go into probe and execute the "stack" request. Do you understand why the stack overflowed? (You will probably want to "break" inorder to stop the output from the stack request.) Recursion is one reason why command names are normally not used for abbreviations.

Now generalize the print command abbreviation to accept star names and test it by typing the following command lines:

```
.ab :pr do "print ([segs &1])"
:pr s*.*
:pr alphabet
```

4. Modify your start_up.ec to include the three new lines as indicated below:

start_up.ec

```
&command_line off
      .     .
      .     .
{previous contents}
      .     .
      .     .
&print Start_up complete.
&quit
```

WORKSHOP EIGHT

Execute your start_up.ec to verify its proper functioning. Execute the new_proc command and observe the results (this may take a minute or so).

5. Create the following exec_com and name it weird.ec.

weird.ec

```
&print Beginning &ec_name exec_com
&if [nequal &n 2] &then &goto LOGOUT

pwd
ls
ind &1.pl1
pl1 &1.pl1
pr &1.pl1
&quit

&label LOGOUT
&print Bye Bye
logout
&quit
```

6. Execute your weird exec_com using the following command lines (you will receive some error messages since sub.pl1 does not exist):

```
ec weird.ec add
ec weird.ec sub
```

7. Add the name weird.absin to your weird.ec segment (an weird.ec weird.absin). Verify by executing the following list commands:

```
ls weird.ec
ls weird.absin
ls weird.**
ls w*.**
ls **.ec
```

8. Submit an absentee request using your weird.absin segment as the absin segment. (ear weird.absin -ag add). Attempt to observe the absentee process by repeatedly typing:

```
who Student_??
```

Hopefully you will notice yourself listed twice, for example:

```
Student_08.F01
Student_08.F01*
```

9. After the absentee job has completed, you will find a weird.absout segment in the same directory as the weird.absin segment. Print and observe its contents. In particular, note the login and logout times, and the fact that the absentee process used your start_up.ec (which is potentially undesirable).

10. Modify your start_up.ec to recognize and respond to an absentee login. Add the &if control line as indicated.

start_up.ec

```
&command_line off
abbrev
&if [equal &2 absentee] &then &quit
      .       .
      .       .
      .       .
      .       .
```

11. Submit again an absentee request for weird.absin. This time, however, schedule the absentee job for 8 am local time tomorrow morning (use your time zone if necessary) . For example:

```
ear weird.absin -time "Friday 8am est" -ag add
```

12. Execute your weird exec_com using two arguments. For example:

```
ec weird.ec add mad
```

Why were you logged out?

WORKSHOP NINE

- Log in to the system and see if your absentee request from yesterday ran as expected.

- Log out using the -hold control argument (logout -hold). Log back in.

- Type the following sequence of change_wdir commands. Verify your working directory after each line via the print_wdir command.

```
cwd <
cwd <
cwd
cwd <<
cwd
cwd <Student_01
cwd
```

- Type the following sequence of commands. (You will receive some error messages.)

```
cr A.ec B.ec C.ec                  (create)
ls **.ec                           (list)
an **.ec ==.absin                  (add_name)
ls **.ec                           (list)
rn (A B C).ec (a b c).ec           (rename)
ls **.ec                           (list)
cr "X Y" "A""B"                    (create)
ls *                               (list)
rn a.ec ;hmu                       (rename)
rn a.ec ";hmu"                     (rename)
ls *                               (list)
```

Type the following sequence of commands: (Note: You will receive some "Segment not found" error messages)

```
wh pwd                             (where)
pwd                                (print_wdir)
wh funny                           (where)
funny                              (funny)
in >udd>F01>Student_01>funny       (initiate)
wh funny                           (where)
funny                              (funny)
```

WORKSHOP NINE

6. Type the following sequence of commands: (Note: You will receive some "Segment not found" error messages)

```
     ls -lk                                (list)
     ls sad                                (list)
     sad                                   (sad)
     lk >udd>F01>Student_01>sad            (link)
     ls -lk                                (list)
     ls sad                                (list)
     wh sad                                (where)
     sad                                   (sad)
```

7. Type the following sequence of commands:

```
     copy Prince P1                        (copy)
     list P*                               (list)
     cpa Prince P1                         (compare_ascii)
     ac rd A.archive Prince P1             (archive)
     ac t A.archive                        (archive)
     list P*                               (list)
     ac x A.archive                        (archive)
     list P*                               (list)
     ss P1                                 (sort_seg)
     pr P1                                 (print)
     cpa Prince P1                         (compare_ascii)
```

8. Type the following sequence of commands:

```
     wh list                               (where)
     ls [wh list]                          (list)
```

Note the permissions you have on the list command procedure (this is the object program). Note also (from the add names) that the list command procedure is "bound" (see the bind command) with other operating system command procedures.

9. Type the following sequence of commands. (You will not receive any terminal output for the first four commands.)

```
     fo t_file                             (file_output)
     pwd                                   (print_wdir)
     ls -all                               (list)
     who .F01                              (who)
     ro                                    (revert_output)
     pwd                                   (print_wdir)
     pr t_file                             (print)
```

WORKSHOP NINE

0. Type the following sequence of commands:

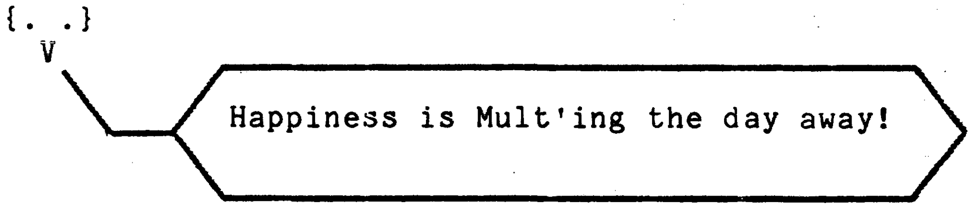
```
pr Prince                           (print)
ll 35                               (line_length)
pr Prince                           (print)
ll 80                               (line_length)
```

1. Type the following sequence of commands:

```
gr -string "Done Master!" -set      (general_ready)
pwd                                 (print_wdir)
time                                (time)
ru                                  (resource_usage)
gr -revert                          (general_ready)
date                                (date)
```

2. Turn off the ready message (ready_off). Most experienced users prefer this mode because they can interact faster. Type the following sequence of commands just to get the feel of having no ready message:

```
gq                                  (get_quota)
gq <                                (get_quota)
cwd                                 (change_wdir)
ll 79                               (line_length)
an P1 P2                            (add_name)
>udd>F01>Student_01>reassure        (reassure)
ready                               (ready)
dn P2                               (delete_name)
cr seg_3                            (create)
.                                   (period)
ready_on                            (ready_on)
ls seg_3                            (list)
logout                              (logout)
```



This page has intentionally
been left blank.