INTRODUCTION
COURSE OVERVIEW

1. BASIC I/O SYSTEM REVIEW
2. BASIC I/O SYSTEM CONFIGURATION
3. BOOTSTRAP LOADER
4. FILES UTILITY
5. DEVICE DRIVERS
6. O.S. EXTENSIONS
7. EXTENDED I/O SYSTEM
8. HUMAN INTERFACE
9. START UP SYSTEM
CHAPTER 1
BASIC I/O SYSTEM REVIEW
THE FILE

- A COLLECTION OF DATA
- ORGANIZED AT BYTE LEVEL
- MEDIA INDEPENDANT (AT FILE LEVEL)
THE FILE

- A COLLECTION OF DATA
  - DATA FROM PROCESS CONTROL
  - TEXT (LETTER, REPORT, ETC.)
  - INFORMATION PASSED FROM TASK TO TASK
  - REFERENCE INFORMATION (INVENTORY, PAYROLL, ETC.)
THE FILE

- ORGANIZED AT BYTE LEVEL
- LENGTH
- POINTER

Diagram:

θ ──► READ

SEEK ───► SEEK

WRITE ───► LENGTH

POINTER
THE FILE

• MEDIA INDEPENDENT (AT FILE LEVEL)

• DESIGN FLEXIBILITY

• TEST FLEXIBILITY

• RUN TIME FLEXIBILITY
ACCESS METHODS

• SEQUENTIAL ACCESS

• RANDOM ACCESS
RMX-86 FILE TYPES

- PHYSICAL
- NAMED
- STREAM
# File Types vs. Access Method

<table>
<thead>
<tr>
<th></th>
<th>Access Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File Type</strong></td>
<td></td>
</tr>
<tr>
<td>Physical</td>
<td><img src="#" alt="Checkmark" /> <strong>Note</strong> <img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>Named</td>
<td><img src="#" alt="Checkmark" /></td>
</tr>
<tr>
<td>Stream</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** Device must support random access.
SOME EXAMPLES

SEQUENTIAL/PHYSICAL — THE TELETYPewriter
THE LINEPRINTER, ETC.

RANDOM/NAMED — RMX DISK OR DISKETTE
OR BUBBLE

SEQUENTIAL/STREAM — INTERTASK DATA TRANSFER

RANDOM/PHYSICAL

OR

SEQUENTIAL/PHYSICAL

— READ OR WRITE ANY FORMAT
DISKETTE OR TAPE
THE FILE DRIVER

USER I/O REQUEST
READ, WRITE, SEEK, ETC.

FILE DRIVER

I/O REQUEST
SEGMENT(S)

I/O RESULT
SEGMENT

USER PROGRAM

DEVICE DRIVER
THE DEVICE DRIVER

% REQUEST SEGMENT

FILE DRIVER

% RESULT SEGMENT

DATA FROM/TO DEVICE

DEVICE COMMANDS

DEVICE
BASIC I/O SYSTEM

"MOST INTIMATE FORM OF I/O SYSTEM INTERACTION."

ADVANTAGES

- ASYNCHRONOUS (ALLOWS OVERLAPPED I/O AND USER PROCESSING)

- MOST COMPACT VERSION OF I/O SYSTEM.

DISADVANTAGE

- FAIRLY COMPLICATED USER INTERFACE
BASIC I/O SYSTEM
INTERACTION EXAMPLE

/* NOW START I/O PROCESSING */
call rqaread( , , @ respmbx, @ status);
/* TEST RESULT OF CALL ITSELF */
if (status <> @) then
/* BAD CALL */
  bad_call: do;
  /* HANDLE PROBLEM WITH CALL */
  end bad_call;
else
  /* O.K. SO FAR */
  good_call: do;
BASIC I/O SYSTEM
INTERACTION EXAMPLE (CONTINUED)

/* DO CONCURRENT PROCESSING */
/* NOW GET RESPONSE FROM I/O SYSTEM */
MSGTKN = RQ_RECEIVE_MESSAGE (RESPMBX, , , @STATUS);
/* CHECK CALL */
IF (STATUS > Ø) THEN
/* BAD SYSTEM CALL HANDLED HERE */
ELSE
/* WE CAN PROCEED */
   GO_ON: DO;
      MSGPTR = POINTERIZE (MSGTKN);
      }
BASIC I/O SYSTEM

INTERACTION EXAMPLE (CONTINUED)

/* CHECK STATUS FIELD I/O RESULT SEGMENT */
IF (MSG. STATUS < > 0) THEN
    /* BAD I/O, HANDLE IT AND DELETE IORS */
ELSE
    /* FINALLY PROCESS DATA IN THE BUFFER */
EXTENDED I/O SYSTEM
"THE USER FRIENDLY I/O INTERFACE"

ADVANTAGES

• SIMPLE INTERFACE - SINGLE CALL

• AUTOMATIC BUFFERING - READ AHEAD, WRITE BEHIND

DISADVANTAGES

• MORE MEMORY REQUIRED (ABOVE BASIC I/O SYSTEM)

• NOT EFFICIENT FOR RANDOM ACCESS
EXTENDED I/O SYSTEM
INTERACTION EXAMPLE

/* READ DISK FILE AND PLACE DATA IN BUFF */
NUMBYTES = RQS READ MOVE ( , BUFF PTR, BYTES REQ, @STATUS);

/* CHECK STATUS */
IF (STATUS < > 0) THEN
   /* PROCESS ERROR */
ELSE
   /* PROCESS DATA */
WHICH WAY TO GO?

 BASIC  EXTENDED
WHY USE THE BASIC I/O SYSTEM?

• I NEED EXTREME FLEXIBILITY

• I NEED EFFICIENT RANDOM ACCESS

• I MUST CONSERVE MEMORY

• I WANT TO OVERLAP MY PROCESSING WITH I/O PROCESSING

• I'M MASOCHISTIC
WHY USE THE EXTENDED I/O SYSTEM?

• I LIKE THE EASY INTERFACE

• I CAN AFFORD THE MEMORY

• I'M PRIMARILY USING SEQUENTIAL ACCESS
  SO AUTOMATIC BUFFERING HELPS THRUPUT

• I DO NOT NEED OVERLAPPED I/O AND
  USER PROCESSING
CHAPTER QUIZ

1. WHAT ARE THE THREE ATTRIBUTES OF A FILE?
   a. ________    b. ________    c. ________

2. WHAT ARE THE THREE RMX-86 FILE TYPES?
   a. ________    b. ________    c. ________

3. CAN I TREAT A STREAM FILE DRIVER IN A RANDOM ACCESS MANNER? ________.

4. WHAT COMBINATION OF FILE TYPE AND ACCESS METHOD WOULD I USE TO READ AN ISIS FORMAT DISKETTE?
   ________ AND ________.
CHAPTER QUIZ (CONT.)

5. WHAT KIND OF INFORMATION PASSES BETWEEN THE FILE DRIVER AND THE DEVICE DRIVER?

6. LIST AN ADVANTAGE OF THE BASIC I/O SYSTEM.

7. LIST AN ADVANTAGE OF THE EXTENDED I/O SYSTEM.
CHAPTER 2

BASIC I/O SYSTEM CONFIGURATION
BASIC I/O SYSTEM CONFIGURATION

• SELECT FEATURES DESIRED
  (I TABLE.A86)

• DESCRIBE THE I/O DEVICES
  (IDEVCF.A86)
TABLE A86
SYSTEM CALL SELECTION
NON-FILE INTERFACE

- PARAMETER INTERFACE
  LOCAL PARAMETERS

- CONFIGURATION INTERFACE
  ATTACH-DETACH

- POWER-FAIL INTERFACE
  POWER-UP, POWER-DOWN

- DATE/TIME INTERFACE
  DATE AND TIME INFORMATION
ITABLE.A86
FILE DRIVER GLOBAL DATA

- NUMBER OF FILE DRIVERS
- ATTACH DEVICE PRIORITY
- TIMER TASK PRIORITY
ITABLE.A86
FILE DRIVER TABLES

• DO NOT TOUCH!

GOT THAT?
OPTIONAL FEATURE SELECTION

- DUMMY_TIMER
- NO_CREATE_FALSE
- NO_TRUNCATE
- NO_ALLOCATE
I DEV CF. A86

$INCLUDE(

DEVICE-UNIT INFO.BLOCKS

DEVICE INFO. TABLES

UNIT INFO. TABLES

GENERAL DEVICE INFO.

END!
I DEVCF, A86
DEVICE-UNIT INFORMATION BLOCKS

COMPONENTS

- DEVICE NAME (UP TO 14 CHARACTERS)
- FILE DRIVERS (SUPPORTED)
- FUNCTIONS (SUPPORTED)
- FLAGS (DISKETTES ONLY, TYPE OF DRIVE)
- DEVICE GRANULARITY (RANDOM ACCESS USUALLY)
- LOW/HIGH SIZE (DEVICE STORAGE CAPACITY)
- DEVICE NUMBER (PER DEVICE (OR CONTROLLER))
- UNIT NUMBER (PER UNIT ON A GIVEN DEVICE)
I DEVCF.A86

DEVICE-UNIT INFORMATION BLOCKS

- DEVICE-UNIT NUMBER (UNIQUE IN THE SYSTEM)
- INIT_IO (INITIALIZE I/O DEVICE DRIVER)
- FINISH_IO (FINISH I/O DEVICE DRIVER)
- QUEUE_IO (QUEUE I/O DEVICE DRIVER)
- CANCEL_IO (CANCEL I/O DEVICE DRIVER)
- DEVICE_INFO (ADDRESS OF DEVICE INFO. BLOCK)
- UNIT_INFO (ADDRESS OF UNIT INFO. BLOCK)
- UPDATE_TIMEOUT (FREQUENCY OF UPDATE)
- NUM_BUFFERS (NUMBER OF BUFFERS FOR RANDOM ACCESS DEVICES)
- PRIORITY (SERVICE TASK PRIORITY)
## DEVICE INFORMATION TABLES

**COMMON OR RANDOM DEVICE TABLE**

- **LEVEL** (Interrupt Level)
- **PRIORITY** (Device Interrupt Task)
- **STACK_SIZE** (User Written Interrupt Procedure)
- **DATA_SIZE** (User Portion of Device Data Object)
- **NUM_UNITS** (Number of Units Supported)
- **DEVICE_INIT** (User Written Device Initialization)
- **DEVICE_FINISH** (" " " " Finish)
- **DEVICE_START** (" " " " Start)
- **DEVICE_STOP** (" " " " Stop)
- **DEVICE_INTERRUPT** (" " " " Interrupt)
UNIT INFORMATION TABLES

• NORMALLY RANDOM ONLY

RANDOM_UNIT_INFO

• TRACK_SIZE (ONE TRACK, Ø IF CONTROLLER CAN CROSS TRACK BOUNDARIES)

• MAX_RETRY (NUMBER OF ATTEMPS)

• Ø
I.DEV CF. A86

GENERAL DEVICE INFORMATION

DEVICE TABLES

- TOTAL NUMBER OF D.U.I.B.'S
- NUMBER OF DEVICE UNITS DEFINED
- NUMBER OF DEVICES DEFINED
ASSEMBLING, LINKING AND LOCATING THE BASIC I/O SYSTEM

- MODIFY ITABLE.A86 AND IDEVCF.A86 TO YOUR TASTES

- SET UP SUBMIT FILE TO MATCH YOUR DEVELOPMENT RESOURCES

- SUBMIT :fx:IOS(DATE,LOC_ADR)
CHAPTER QUIZ

1. T-F I can modify the file driver tables.

2. Which file contains the dummy timer?

3. What are the 3. tables for a random driver?
   A. _________  B. _________  C. _________

4. In which file do you find the address of the device start procedure?
   ________________________________
CHAPTER QUIZ
(CONTINUED)

5. IF I HAD 3 ISBC 204 CARDS AND 1 ISBC CARD IN ADDITION TO THE TERMINAL IN A SYSTEM, HOW MANY DEVICES WOULD I HAVE? __________

6. EACH DISK INTERFACE CARD HAS 2 DRIVES ASSOCIATED WITH IT. HOW MANY DEVICE-UNIT NUMBERS WOULD I HAVE? __________
CHAPTER 3

THE BOOTSTRAP LOADER
WHAT IS IT?

THE BOOTSTRAP LOADER IS A PROGRAM WHICH ALLOWS AN RMX-86 SYSTEM TO BE LOADED INTO MEMORY FROM SOME PERIPHERAL DEVICE.
BOOTSTRAP LOADER FEATURES

- AUTOMATIC OR CONTROLLED LOADING

- AUTOMATIC OR USER SELECTABLE DEVICE

- AUTOMATIC OR USER SELECTABLE FILE NAMES
DEVICES CURRENTLY SUPPORTED

- ISBC 204 SINGLE DENSITY FLOPPY DISK
- ISBC 206 CDC HAWK HARD DISK
- ISBC 215 WINCHESTER DISK
- ISBX 218 SINGLE DENSITY FLOPPY DISK (WHEN USED WITH ISBC 215)
- ISBC 254 BUBBLE MEMORY CONTROLLER
BOOTSTRAP LOADER STRUCTURE

ROM

FIRST STAGE

DEVICE DRIVER(s)

100 TO 500 BYTES + DRIVER

LOADS

SECOND STAGE

≈ 6K BYTES

APPLICATION SYSTEM

?
FIRST STAGE OPTIONS

- LOCATION OF FIRST STAGE IN ROM
  (ASSIGNED BY USER THROUGH LOC86)

- LOCATION OF SECOND STAGE IN RAM
  (ASSIGNED BY USER THROUGH LOC86)

- DEVICE SELECTION METHOD
  (ASSIGNED BY USER THROUGH CONFIGURATION)

- FILE SELECTION METHOD
  (ASSIGNED BY USER THROUGH CONFIGURATION)
BOOTSTRAP LOCATION NOTES

- FIRST STAGE MUST BE AVAILABLE AT RESET (USUALLY IN ROM)

- SECOND STAGE MUST NOT OCCUPY MEMORY ALREADY OCCUPIED BY THE SYSTEM TO BE LOADED (CODE AREAS OR INITIALIZED DATA AREAS)
DEVICE SELECTION

• NONE (ONE DEVICE ONLY)

• AUTOMATIC SELECTION (HUNT FOR READY DEVICE)

• MANUAL SELECTION (PROMPT USER FOR DEVICE THROUGH SYSTEM TERMINAL)
DEVICE SELECTION NOTES

• NONE

  • ONE TRY PER RESET. IF DEVICE IS NOT READY: QUIT.

• AUTOMATIC

  • TRY EACH DEVICE IN THE LIST IN ROTATION UNTIL A READY DEVICE IS FOUND. IF NO DEVICE IS FOUND READY, REPEAT LIST.
DEVICE SELECTION NOTES (CONTINUED)

- MANUAL
  - PROMPT USER FOR A DEVICE NAME THROUGH THE TERMINAL
  - IF RESPONSE IS ON THE LIST TRY THAT DEVICE
  - IF RESPONSE IS NOT ON THE LIST BEGIN AUTOMATIC DEVICE SELECTION FROM LIST ENTERED AT CONFIGURATION
FILE SELECTION NOTES

• NONE
  • FILE NAMED `/SYSTEM/REM86`
    IS LOADED FROM SELECTED DEVICE

• AUTOMATIC
  • SAME FILE IS LOADED FROM THE FIRST
    AVAILABLE DEVICE

• MANUAL
  • IF FIRST CHARACTER IS A COLON, TRY TO
    PARSE A DEVICE NAME. IF DEVICE NAME
    IS IN TABLE, TRY IT.
FILE SELECTION NOTES
(CONTINUED)

• MANUAL (CONTINUED)

• IF BOOTSTRAP CANNOT PARSE A DEVICE NAME OR IF NAME PARSED IS NOT IN THE TABLE, SWITCH TO AUTO DEVICE SELECTION AND USE STRING AS A FILE NAME.

• BLANK LINE IS INTERPRETED AS DEFAULT FILE NAME /SYSTEM/RMIX86 WITH AUTO DEVICE SELECTION.

• :fφ: FILENAME = :fφ: /SYSTEM/FIILENAME
• :fφ: /FILENAME = :fφ: FILENAME
DRIVER CONFIGURATION

- SUPPLY ADDRESS PARAMETERS

- ASSEMBLE THE RESULT

EXAMPLE:

$\texttt{INCLUDE (}:\texttt{FX: B204.INC})
\%
\texttt{B204 (}:\texttt{FAFH, 128, 26})

\begin{tabular}{|c|c|c|}
\hline
DE\texttt{VICE} & SE\texttt{CTOR} & \# S\texttt{ECTORS/TRACK} \\
\hline
ADDRESS & SIZE & \\
\hline
\end{tabular}

(NOTE: THESE MACROS CHANGE FOR EACH DEVICE. SEE CHAPTER II OF THE CONFIGURATION MANUAL)
BOOTSTRAP CONFIGURATION

• SELECT DESIRED BOOTSTRAP FEATURES

• LIST BOOTSTRAP DEVICES

• CONFIGURE EACH DEVICE

• ASSEMBLE, LINK AND LOCATE THE RESULT
SELECT BOOTSTRAP FEATURES

• AUTO MACRO
  (ENABLES AUTOMATIC DEVICE SELECTION)

• CONSOLE MACRO
  (ALLOWS RUN TIME FILE SELECTION)

• MANUAL MACRO
  (ALLOWS RUN TIME DEVICE SELECTION)

• IF NO MACROS ARE USED, DEVICE AND FILE SELECTION WILL REVERT TO DEFAULTS WITH A SINGLE TRY.
LIST BOOTSTRAP DEVICES

• DEVICE MACRO
  • FOR AUTO SELECT DEVICES ARE SCANNED IN ORDER OF THE CONFIGURATION FILE

• MACRO SPECIFIES:
  • NAME OF DEVICE
  • DEVICE-UNIT NUMBER (SAME AS BIOS)
  • DEVICE INITIALIZATION ROUTINE ENTRY POINT
  • DEVICE READ ROUTINE ENTRY POINT
DRIVER CONFIGURATION
(USER SUPPLIED DRIVERS)

• YOU CREATE DEVICE $INIT AND DEVICE $READ Routines.

• ASSEMBLE WITH ENTRY POINTS AS PUBLICS

• LINK TO REST OF BOOTSTRAP Routines

(NOTE: Routines MUST BE LARGE MODEL OF COMPUTATION)
EXAMPLE BOOTSTRAP CONFIGURATION(S)

• NO DEVICE SELECTION

NAME SIMPLE

$INCLUDE (:fx: BS1.INC)

%DEVICE (wF0, ø, DEVICE INIT 215, DEVICE READ 215)

%END
EXAMPLE BOOTSTRAP CONFIGURATION(S)

• MANUAL (WITH DEVICE SELECTION)

  \$ INCLUDE (\$X: BS1.INC.)
  \% CONSOLE
  \% AUTO
  \% MANUAL
  \% DEVICE (f\%\%\%f, \%\%\%f, DEVICE INIT 204, DEVICE READ 204)
  \% DEVICE (b\%\%b, DEVICE INIT 254, DEVICE READ 254)
  \% END
EXAMPLE BOOTSTRAP CONFIGURATION(s)
(ASSEMBLE, LINK AND LOCATE)
(SIMPLE CASE)

• AFTER BOOTSTRAP CONFIGURATION FILE AND DEVICE CONFIGURATION FILE(s) ARE PREPARED

SUBMIT

:fx: BS1(DATE,ROM,RAM)

WHERE:

DATE = DATE IE 07/27/82

ROM = STARTING CODE ADDRESS
FOR STAGE 1.

RAM = STARTING ADDRESS FOR
STAGE 2.

NOTE: MODIFY :fx: BS1.CSD TO REFLECT YOUR
ARRANGEMENT BEFORE YOU SUBMIT.
EXAMPLE BOOTSTRAP CONFIGURATION
(ASSEMBLE, LINK AND LOCATE)
(COMPLEX CASE)

STEP 1. COMPILe :fx:. BCICO.P86 TO GET CONSOLe ROUTINES FOR DEVICE OR FILE SELECTION

STEP 2. ADD :fx:. BCICO.OBJ TO SUBMIT FILE LINK LIST.

STEP 3. SUBMIT :fx:. BS1( , , )
CHAPTER QUIZ

1. WHAT ARE THE THREE MODES OF LOADING?
   A. ________  B. ________  C. ________

2. WHAT ARE 2 OF THE DEVICES I CAN BOOT FROM?
   A. ________  B. ________

3. HOW DOES THE SECOND STAGE GET ON THE DEVICE?
   __________________________

4. WHAT IS THE FILE NAME FOR THE CONSOLE INTERFACE FILE?
   __________________________
CHAPTER 4

THE FILES UTILITY
WHAT IS IT?

- THE FILES UTILITY IS A PROGRAM RUNNING ON AN RMX-86/ISIS SYSTEM WHICH ALLOWS YOU TO CREATE RMX-86 FORMAT DISKETTES BEFORE YOU HAVE A WORKING USER CREATED SYSTEM.
FILES UTILITY FUNCTIONS

• FORMAT AN RMX-86 DISKETTE.
• COPY FILES FROM AN RMX-86 DISKETTE TO AN ISIS FORMAT DISKETTE.
• COPY FILES FROM AN ISIS FORMAT DISKETTE TO AN RMX-86 FORMAT DISKETTE
• DELETE FILES ON AN RMX-86 DISKETTE
• CREATE A DIRECTORY FILE ON AN RMX-86 DISKETTE
• DISPLAY THE CONTENTS OF AN RMX-86 DISKETTE DIRECTORY IN SEVERAL FORMATS
HARDWARE REQUIRED

- INTEL DEVELOPMENT SYSTEM WITH 64K RAM AND AT LEAST ONE DISK DRIVE (MDS-800, SERIES II, SERIES III, NDS-1)
- LSBC 86/12A WITH AT LEAST 192 K RAM AND AT LEAST 1 DISK DRIVE
- 957A INTELLEC TO 86/12A INTERFACE AND MONITOR
WHERE DOES THE FILES UTILITY FIT IN?

STEP 1. DEVELOP USER SOFTWARE ON THE INTELLEC SYSTEM (SERIES II, SERIES III, MDS 800)

STEP 2. TEST LOAD AND EXECUTE SOFTWARE USING THE 957A INTERFACE

STEP 3. FORMAT A BOOTABLE DISK AND LOAD TESTED SOFTWARE ONTO IT

STEP 4. PLACE BOOTSTRAP STAGE I INTO 86/12A PROM.

STEP 5. SET UP iSBC SYSTEM, LOAD DISKETTE FROM STEP 3 INTO A DRIVE AND PRESS RESET.
Files Utility Usage

- To Invoke the Files Utility

  a. Set up Hardware and Software

  b. Type

  ```sh
  submit :Fx: files (:Fx:)
  SBC861
  G
  ```
# FILES UTILITY COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>ABBREVIATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTACHDEVKE</td>
<td>AD</td>
</tr>
<tr>
<td>BREAK</td>
<td>BR</td>
</tr>
<tr>
<td>CREATEDIR</td>
<td>CD</td>
</tr>
<tr>
<td>DELETE</td>
<td>DE</td>
</tr>
<tr>
<td>DETACH</td>
<td>DT</td>
</tr>
<tr>
<td>COMMAND</td>
<td>ABBREVIATION</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>DIR</td>
<td>DI</td>
</tr>
<tr>
<td>DOWNCOPY</td>
<td>DC</td>
</tr>
<tr>
<td>FORMAT</td>
<td>FO</td>
</tr>
<tr>
<td>HELP</td>
<td>HE</td>
</tr>
<tr>
<td>UPCOPY</td>
<td>UC</td>
</tr>
</tbody>
</table>
A TYPICAL FILES UTILITY USAGE SEQUENCE

-SUBMIT :F1:FILES(:F1:)
-SBC861

ISIS-II iSBC 86/12 LOADER, V2.0

iSBC 86/12 MONITOR V2.0

.L:F1:NUCLUS
.L:F1:IOS
.L:F1:EIOS
.L:F1:FILES
.L:F1:FRGOT
.E

::F0:SUBMIT RESTORE :F1:FILES.CS(:VI:)
-SBC861

ISIS-II iSBC 86/12 LOADER, V2.0

*CONTROL-C*

.G

iRMX 86 FILES UTILITY V3.0

*FORMAT F0 LAB2 IL=5 NF=50 NAMED

*VOLUME FORMATTED - NAMED FILE OPTION

   GRANULARITY = 128
   NUMBEROFNODES = 50
   INTERLEAVE = 5
A TYPICAL FILES UTILITY USAGE SEQUENCE

*AD :F0: = F0
*DIR :F0:
   0 FILES
*CREATEDIR :F0:SYSTEM
   :F0:SYSTEM ,CREATED
*UPCOPY :F1:FIRST.LIB TO :F0:SYSTEM/RMX86
*DIR :F0:
   SYSTEM
   1 FILES
*DIR :F0:SYSTEM
   RMX86
   1 FILES
*DETACH :F0:
   :F0: ,DETACHED
*BR
A TYPICAL FILES UTILITY USAGE SEQUENCE

*BREAK* AT 1800:186A

.E

._
WARNING!!!

TO CHANGE A DISKETTE:

1. DETACH

2. CHANGE DISKETTES

3. ATTACH DEVICE (OR FORMAT)
CHAPTER QUIZ

1. TRUE-FALSE  
THE FILES UTILITY ALLOWS YOU TO  
DISPLAY THE DIRECTORY OF AN ISIS DISKETTE.

2. NAME THREE DEVICES THAT CAN BE FORMATTED BY THE  
FILES UTILITY.  
a._______  
b._______  
c._______

3. WHY CAN'T I REMOVE A DISKETTE AT ANY TIME WHILE  
I'M USING THE FILES UTILITY?

______________________________________________
WRITING DEVICE DRIVERS
FOR THE IRMX 86 I/O SYSTEM
TOPICS TO BE DISCUSSED:

- INTRODUCTION AND CONCEPTS
- DEVICE DRIVER INTERFACES
- COMMON DEVICE DRIVERS
- RANDOM ACCESS DEVICE DRIVERS
- CUSTOM DEVICE DRIVERS
- DEVICE DRIVER CONFIGURATION
REFERENCE MANUALS REQUIRED:

- IRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL
- IRMX 86 SYSTEM PROGRAMMERS REFERENCE MANUAL
- IRMX 86 CONFIGURATION GUIDE
- GUIDE TO WRITING DEVICE DRIVERS FOR THE IRMX 86 I/O SYSTEM
SYSTEM CONSTRUCTION

- THE I/O SYSTEM IS IMPLEMENTED AS A SET OF FILE
  DRIVERS AND A SET OF DEVICE DRIVERS
- YOUR APPLICATION COMMUNICATES WITH FILE DRIVERS

1. PHYSICAL
   USARTS, PRINTERS ..........

2. NAMED
   DISK, BUBBLE MEMORY.....

3. STREAM
   A PIPELINE BETWEEN TWO TASKS USING
   I/O SYSTEM CALLS
SYSTEM CONSTRUCTION

- FILE DRIVERS COMMUNICATE WITH DEVICE DRIVERS

- DEVICE DRIVERS COMMUNICATE WITH DEVICES

APPLICATION TASKS
FILE INDEPENDENT INTERFACE
FILE DRIVERS
DEVICE INDEPENDENT INTERFACE
DEVICE DRIVERS
DEVICES
INTERFACE

- The interface between your application and file drivers and between file drivers and device drivers is standard.

- This allows for:
  
  - Device independence
  - Hardware configuration changes without extensive software modifications
  - A greater range of devices can be supported
I/O DEVICE AND DEVICE DRIVERS

• EACH I/O DEVICE CONSISTS OF A CONTROLLER AND ONE OR MORE UNITS

• EACH CONTROLLER IS ASSIGNED A DEVICE NUMBER

• EACH UNIT IS ASSIGNED A UNIT NUMBER FOR THAT DEVICE AND A DEVICE UNIT NUMBER FOR ALL DEVICES IN THE I/O SYSTEM
SCHEMATIC OF SOFTWARE AT INITIALIZATION TIME

<table>
<thead>
<tr>
<th>TASKS</th>
<th>APPLICATION SOFTWARE TASKS</th>
<th>TASKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHYSICAL FILE DRIVER</td>
<td>NAMED FILE DRIVER</td>
<td>STREAM FILE DRIVER</td>
</tr>
</tbody>
</table>

CONFIGURATION INTERFACE

<table>
<thead>
<tr>
<th>DEVICE DRIVER</th>
<th>DEVICE DRIVER</th>
<th>DEVICE DRIVER</th>
<th>DEVICE DRIVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVICE CONTROLLER</td>
<td>DEVICE CONTROLLER</td>
<td>DEVICE CONTROLLER</td>
<td>DEVICE CONTROLLER</td>
</tr>
<tr>
<td>DEVICE UNIT</td>
<td>DEVICE UNIT</td>
<td>D. UNIT</td>
<td>D. UNIT</td>
</tr>
</tbody>
</table>

5-8
I/O REQUESTS

To the device driver a request is a request from the I/O system for the device to perform a certain operation.

- Read
- Write
- Seek
- Special
- Attach device
- Detach device
- Open
- Close

These requests are passed to the device driver in a segment type object.
COMPONENTS OF A DEVICE DRIVER

• AT ITS HIGHEST LEVEL A DEVICE OPERATOR CONSISTS OF FOUR PROCEDURES
  • INITIALIZE I/O
  • FINISH I/O
  • QUEUE I/O
  • CANCEL I/O

FOR EVERY I/O REQUEST THE I/O SYSTEM MAY CALL ONE OR MORE OF THESE PROCEDURES
INITIAL I/O PROCEDURE

- The I/O system calls this procedure whenever a RQ$PHYSICAL$ATTACH$DEVICE system call is made and there are currently no other units attached to this device.
FINISH I/O

- THE I/O SYSTEM CALLS THIS PROCEDURE WHENEVER A RQ$PHYSICAL$DETACH$DEVICE SYSTEM CALL IS MADE AND THERE ARE CURRENTLY NO OTHER UNITS ATTACHED TO THIS DEVICE
QUEUE I/O

• THIS PROCEDURE IS CALLED BY THE I/O SYSTEM FOR ALL USER I/O REQUESTS. THIS PROCEDURE MUST PLACE THE REQUEST ON THE REQUEST QUEUE SO THAT IT MAY BE PROCESSED WHEN APPROPRIATE.

IF DEVICE IS NOT BUSY THIS PROCEDURE MUST ALSO START THE I/O FUNCTION
CANCEL I/O

• THIS PROCEDURE IS CALLED BY THE I/O SYSTEM WHEN:

  • A RQ$A$PHYSICAL$DETACH$DEVICE CALL IS MADE WITH THE HARD DETACH OPTION SPECIFIED

  • IF THE JOB CONTAINING THE TASK THAT MADE THE I/O REQUEST SELECTED
INTERRUPT HANDLERS

- After a device has finished processing an I/O request, it sends an interrupt to the processor. At this time, the handler may service the interrupt or signal an interrupt task that will service the interrupt.

Remember that an interrupt handler is limited to the type of RMX calls that it may make.
INTERRUPT TASKS

INTERRUPT TASKS FEED THE RESULTS OF THE I/O REQUEST BACK TO THE I/O SYSTEM IF THE REQUEST IS FINISHED.

IF THE REQUEST IS NOT FINISHED THIS TASK WILL INITIATE THE NEXT STAGE OF THE REQUEST.

IF THERE ARE ADDITIONAL REQUESTS ON THE QUEUE THEN THIS TASK MUST START THE NEXT REQUEST.
DEVICE DRIVER TYPES

- COMMON DEVICE DRIVERS
  EASIEST TO IMPLEMENT

- RANDOM ACCESS DEVICE DRIVERS
  MUCH THE SAME AS COMMON DEVICES

- CUSTOM DEVICE DRIVERS
  MORE COMPLEX THAN COMMON OR RANDOM
  NEEDED FOR MORE SOPHISTICATED DEVICES
COMMON DEVICE REQUIREMENTS

• SIMPLE DEVICES - PRINTERS, USARTS

• DATA EITHER READ OR WRITTEN TO THE DEVICE DOES NOT NEED TO BE BROKEN UP INTO SPECIFIC BLOCK SIZES

• A FIRST IN/FIRST OUT QUEUE FOR THE REQUESTS IS SUFFICIENT

• ONLY ONE INTERRUPT LEVEL IS NEEDED FOR THE DEVICE
RANDOM ACCESS DEVICE DRIVER REQUIREMENTS

• DEVICES SUCH AS DISKS AND BUBBLE MEMORY

• THE DEVICE MUST SUPPORT RANDOM ACCESS SEEK

• THE I/O REQUEST MUST BE BROKEN UP INTO SPECIFIC BLOCK LENGTHS (TRACK AND SECTOR, BUBBLE PAGE)

• A FIFO QUEUE IS SUFFICIENT

• ONLY ONE INTERRUPT LEVEL IS NEEDED FOR THE DEVICE
CUSTOM DEVICE DRIVER REQUIREMENTS

• IF THE DEVICE DOES NOT FIT INTO THE CATEGORY OF EITHER COMMON OR RANDOM ACCESS THEN YOU MUST WRITE A CUSTOM DEVICE DRIVER

• ANY DEVICE THAT REQUIRES PRIORITY QUEUES

• ANY DEVICE THAT REQUIRES MORE THAN ONE INTERRUPT LEVEL

• ANY DEVICE THAT REQUIRES THE INTERRUPT HANDLER TO SERVICE MORE THAN ONE INTERRUPT BEFORE SIGNALLING THE INTERRUPT TASK
DEVICE DRIVER QUIZ #1

1. WHAT ARE THE THREE TYPES OF FILE DRIVERS?

2. APPLICATION TASKS CALL FILE DRIVERS-(TRUE - FALSE)

3. WHAT IS THE DIFFERENCE BETWEEN A DEVICE AND A UNIT?

4. WHAT OBJECT TYPE IS AN I/O REQUEST?

5. WHAT ARE THE COMPONENTS OF THE DEVICE DRIVER?

6. WHEN IS THE INITIALIZE I/O PROCEDURE CALLED?

7. WHAT ARE THE DIFFERENCES BETWEEN A COMMON AND A CUSTOM DEVICE DRIVER?
DEVICE DRIVER INTERFACES
• ALL DEVICE DRIVER INTERFACES ARE IN THE FORM OF DATA STRUCTURES

• THERE ARE TWO I/O SYSTEM INTERFACES
  DEVICE-UNIT INFORMATION BLOCKS - DUIBS
  I/O REQUEST/RESULT SEGMENTS - IORS

• DEVICE INTERFACES DEPEND ON THE DRIVER TYPE FOR BOTH COMMON AND RANDOM ACCESS DEVICES. THE COMMON DEVICE INFORMATION BLOCK IS USED. OTHER DEVICE INTERFACE STRUCTURES ARE USER DEFINED
DEVICE UNIT INFORMATION - DUB

This structure has the following format:
DECLARE DEV$UNIT$INFO$BLOCK STRUCTURE (  

| NAME (14) | BYTE, NAME USED IN ATTACHDEVICE |
| FILE $DRIVERS | WORD, WHAT FILE DRIVERS CAN BE USED |
| FUNCTS | BYTE, WHAT FUNCTIONS ARE SUPPORTED |
| FLAGS | BYTE, FOR DENSITY AND SIDE SPEC ON DISKS |
| DEV$GRAN | WORD, FOR DISKS MIN I/O SIZE |
| LOW$DEV$SIZE | WORD, THE SIZE OF THE DEVICE IN BYTES |
| HIGH$DEV$SIZE | WORD, |
| DEVICE | BYTE, THE I/O SYSTEM DEVICE NUMBER |
| UNIT | BYTE, UNIT NUMBER FOR THIS DEVICE |
| DEV$UNIT | WORD, THE DEVICE UNIT NUMBER |
DEVICE UNIT INFORMATION - DUIB (CONTINUED)

INIT$IO WORD, PROCEDURE ADDRESSES
FINISH$IO WORD,
QUEUE$IO WORD,
CANCEL$IO WORD,
DEVICE$INFO$P POINTER, TO DEVICE INFO
UNIT$INFO$P POINTER, TO UNIT INFO
UPDATE$TIME$OUT WORD, NUMBER OF SYS TIME UNITS
NUM$BUFFERS WORD, NUM BUFFERS FOR PAD DEVICE
PRIORITY BYTE, PRI FOR I/O SERVICE TASK
USING DUIBS

- THE I/O SYSTEM USES THE DUIB TO INVOKE THE DEVICE DRIVER PROCEDURES WHENEVER AN I/O REQUEST IS MADE.

- WHEN AN ATTACH DEVICE CALL IS MADE THE I/O SYSTEM WILL SCAN THE DUIB TABLES FOR A NAME MATCH.

EXAMPLE:

```
CALL RQ$A$PHYSICAL$ATTACH$DEVICE(@6,'STREAM'),
                 2,RMBX,@STATUS);
```

THERE MUST BE A DUIB FOR THE DEVICE NAME 'STREAM' AND IT MUST HAVE STREAM FILE DRIVER CAPABILITY.
DEVICE DRIVER INTERFACES
ATTACHING DEVICES

CALL RQ$A$PHYSICAL$ATTACH$DEVICE (UNITA, ...)

CALL RQ$A$PHYSICAL$ATTACH$DEVICE (UNITB, ...)

CALL RQ$A$PHYSICAL$ATTACH$DEVICE (UNITC, ...)

CALL RQ$A$PHYSICAL$ATTACH$DEVICE (UNITD, ...)

CALL RQ$A$PHYSICAL$ATTACH$DEVICE (UNITE, ...)
DUPLICATION DEVICES

• You may duplicate device and unit numbers in separate DUIBS in order to have different characteristics for the same device.

• For example if you have a disk drive that can have different sector sizes and you might want to have one instance for 128 byte sectors and one for 256 bytes. To do this you duplicate the DUIB with the exception of the name and DEV\$GRAN field.
The I/O Request/Result structure has the following format:

```
DECLARE IORS STRUCTURE C

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS</td>
<td>WORD</td>
<td>Condition code for the operation</td>
</tr>
<tr>
<td>UNIT$STATUS</td>
<td>WORD</td>
<td>If status is E$10 then unit status should be set</td>
</tr>
<tr>
<td>ACTUAL</td>
<td>WORD</td>
<td>The actual amount of data transferred</td>
</tr>
<tr>
<td>ACTUAL$FILL</td>
<td>WORD</td>
<td>Reserved</td>
</tr>
<tr>
<td>DEVICE</td>
<td>WORD</td>
<td>The device number (same as DUIB)</td>
</tr>
<tr>
<td>UNIT</td>
<td>BYTE</td>
<td>The unit number (same as DUIB)</td>
</tr>
<tr>
<td>FUNCT</td>
<td>BYTE</td>
<td>The function to be performed</td>
</tr>
<tr>
<td>SUB $ FUNCT</td>
<td>WORD</td>
<td>Used for special calls</td>
</tr>
<tr>
<td>LOW$DEV$LOC</td>
<td>WORD</td>
<td>The device location in bytes, for random access devices</td>
</tr>
<tr>
<td>HIGH$DEV$LOC</td>
<td>WORD</td>
<td>Access devices this is the sector and track</td>
</tr>
</tbody>
</table>
```
DECLARE IORS STRUCTURE
(Continued)

BUFF$P  POINTER,  WHERE THE DATA IS TO BE READ FROM OR WRITTEN TO
COUNT    WORD, HOW MUCH, IF RANDOM ACCESS THIS WILL ALWAYS BE IN DEVICE GRAN. MULTIPLES
COUNT$FILL   WORD,  RESERVE
AUX$P  POINTER,  USED FOR SPECIAL CALLS
LINK$FOR POINTER,  LINKED LIST FOR I/O REQUEST QUEUES
LINK$BACK  POINTER,
RESP$MBX  WORD, THE RESPONSE MAILBOX FOR THIS REQUEST
DONE BYTE,  I/O REQUEST STATUS
FILL BYTE, RESERVE
CANCEL$ID WORD); THE REQUEST I.D. FOR THE REQUEST
# COMMON DEVICE INFORMATION INTERFACE

- This structure is used for all common and random access device drivers.

```plaintext
DECLARE COMMON$DEV$INFO STRUCTURE(

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL</td>
<td>WORD,</td>
<td>The interrupt level used for this device</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>BYTE,</td>
<td>The initial priority of the interrupt task</td>
</tr>
<tr>
<td>STACK$SIZE</td>
<td>WORD,</td>
<td>The additional amount of stack that your process needs, (not static data)</td>
</tr>
<tr>
<td>DATA$SIZE</td>
<td>WORD,</td>
<td>The amount of data space that your device driver needs, (not static data)</td>
</tr>
<tr>
<td>NUM$UNITS</td>
<td>WORD,</td>
<td>How many units are with this device</td>
</tr>
<tr>
<td>DEVICE$INIT</td>
<td>WORD,</td>
<td>Your init procedure</td>
</tr>
<tr>
<td>DEVICE$FINISH</td>
<td>WORD,</td>
<td>Your finish procedure</td>
</tr>
<tr>
<td>DEVICE$START</td>
<td>WORD,</td>
<td>Your start procedure</td>
</tr>
<tr>
<td>DEVICE$STOP</td>
<td>WORD,</td>
<td>Your stop procedure</td>
</tr>
<tr>
<td>DEVICE$INTERRUPT</td>
<td>WORD</td>
<td>Your interrupt procedure</td>
</tr>
</tbody>
</table>
```

You may append to this structure any information that your device needs, such as I/O addresses...
RANDOM ACCESS DEVICE UNIT INFORMATION BLOCKS

- FOR RANDOM ACCESS DEVICE YOU MUST HAVE A UNIT INFORMATION BLOCK

DECLARE

RAD$UNIT$INFO$BLOCK

STRUCTURE (  

TRACK$SIZE WORD, THE SIZE IN BYTES OF A TRACK

MAX$RETRY WORD, THE MAX NUMBER OF RETRIES TO BE PERFORMED BY THE I/O SYSTEM

RESERVED WORD):

YOU MAY APPEND TO THIS STRUCTURE AND INFORMATION BY THE DEVICE
WRITING DEVICE DRIVER
GENERAL RULES

• IF PL/M 86 IS USED TO WRITE DEVICE DRIVERS THEN THE COMPACT MODEL OF COMPILATION MUST BE USED.

• IF ASM86 IS USED THEN IT MUST BE WRITTEN TO INTERFACE TO COMPACT PL/M 86 PROCEDURES

• THE I/O SYSTEM CODE CAN NEVER EXCEED 64K OF CODE
WRITING COMMON AND RANDOM ACCESS DEVICE DRIVERS
There are certain parameters passed to each device driver procedure:

- **DUIP$P** - A pointer to the DUIB structure for the device

- **D$DATA$P** - A pointer to the data object that was declared in the common device information block

- **IORS$P** - A pointer to the I/O request segment

- **STATUS$P** - A pointer to the I/O system status word
I/O SYSTEM SUPPLIED PROCEDURES

- INIT$IO
- FINISH$IO
- QUEUE$IO
- CANCEL$IO

USER SUPPLIED PROCEDURE

- A DEVICE INITIALIZATION PROCEDURE
- A DEVICE FINISH PROCEDURE
- A DEVICE START PROCEDURE
- A DEVICE STOP PROCEDURE
- A DEVICE INTERRUPT PROCESSING PROCEDURE
THE ADDRESSES OF YOUR DEVICE DRIVER PROCEDURE MUST BE PLACED IN THE COMMON DEVICE INFORMATION BLOCK FOR THE DEVICE
DEVICE INITIALIZATION PROCEDURE

THE INIT$IO PROCEDURE CALL THIS PROCEDURE TO INITIALIZE THE DEVICE

THE FORM OF THE CALL IS:

CALL DEVICE$INIT(DUIB$P, O$P, STATUS$P);

YOU MUST INITIALIZE YOUR DEVICE AND ANY VARIABLES AND SET THE STATUS WORD TO INDICATE THE SUCCESS OR FAILURE OF THIS PROCEDURE

IF YOUR DEVICE DOES NOT NEED ANY INITIALIZATION THEN YOU MAY USE THE DEFAULT$INIT PROCEDURE SUPPLIED BY THE I/O SYSTEM
DEVICE FINISH PROCEDURE

THE FINISH$IO PROCEDURE CALLS THIS PROCEDURE AFTER THE LAST REQUEST HAS BEEN PROCESSED

THE FORM OF THE CALL IS:

CALL DEVICE$FINISH(DUIB$P, D$DATA$P);

YOU MUST DO ANY FINAL PROCESSING FOR YOUR DEVICE WHEN THIS PROCEDURE IS CALLED

IF YOUR DEVICE DOES NOT NEED ANY FINAL PROCESSING THEN YOU MAY USE THE DEFAULT$FINISH PROCEDURE SUPPLIED BY THE I/O SYSTEM
DEVICE START PROCEDURE

Both QUEUE$IO AND THE INTERRUPT TASK CALL THIS PROCEDURE IN ORDER TO START AN I/O FUNCTION

QUEUE$IO CALLS THIS PROCEDURE WHEN A REQUEST IS MADE AND THERE ARE NO REQUESTS ON THE QUEUE

THE INTERRUPT TASK CALLS THIS PROCEDURE WHEN AN I/O REQUEST IS COMPLETED AND THERE ARE ADDITIONAL REQUESTS IN THE QUEUE

THE FORM OF THE CALL IS:

CALL DEVICE$START(IORS$P, DUIB$,
                  D$DATA$P);
DEVICE START PROCEDURE REQUIREMENTS

• START THE DEVICE PROCESSING THE REQUEST

• RECOGNIZE INVALID REQUESTS

• IF DATA TRANSFERS OCCUR THEN UPDATE THE IORS. ACTUAL FIELD

• IF AN ERROR OCCURS UPDATE THE IORS, STATUS AND IORS.UNIT$STATUS FIELDS

• IF THE REQUEST IS COMPLETE SET THE IORS. DONE FIELD TO TRUE
DEVICE STOP PROCEDURE

THIS PROCEDURE IS CALLED TO STOP THE I/O DEVICE FROM PERFORMING THE CURRENT I/O FUNCTION

THE FORM OF THE CALL IS:

CALL DEVICE$STOP(IORS$P, DUIB$P, D$DATA$P);

IF YOUR DEVICE GUARANTEES THAT ALL I/O REQUESTS WILL FINISH WITHIN A REASONABLE AMOUNT OF TIME THEN YOU MAY USE THE DEFAULT$STOP PROCEDURE
DEVICE INTERRUPT PROCEDURE

THE DEVICE INTERRUPT TASK CALL THIS PROCEDURE WHEN AN INTERRUPT HAS BEEN GENERATED BY THE DEVICE.

THE FORM OF THE CALL IS:

```c
CALL DEVICE$ INTERRUPT(IORS$P, DUIB$P, D$DATA$P);
```

YOUR INTERRUPT PROCEDURE MUST DETERMINE IF THE REQUEST IS FINISHED AND SET THE IORS.DONE FIELD TRUE IF IT IS.

IF IT IS NOT COMPLETE YOU MUST INITIATE THE NEXT STEP IN THE PROCEDURE.
DEVICE INTERRUPT PROCEDURE

EXAMPLE:

1. Your application task made a $RA$READ call to a disk.

2. Your start procedure initiated a seek request for a disk drive to position the head over the proper track.

3. The device generated an interrupt to signal the completion of the seek function.

4. The interrupt procedure started the read function on the disk.

5. The disk generated an interrupt when the data transfer was complete.

6. The interrupt procedure set the IORS, actual field and the IORS.DONE field to indicate the request was complete.
COMMON AND RANDOM ACCESS DEVICE DRIVER QUIZ

1. THE MINIMUM NUMBER OF PROCEDURES THAT YOU MUST WRITE IS?

2. HOW DOES THE I/O SYSTEM KNOW WHEN THE REQUEST IS COMPLETE?

3. HOW DOES THE DEVICE DRIVER INFORM THE I/O SYSTEM OF THE SUCCESS OR FAILURE OF A REQUEST?

4. HOW DOES THE I/O SYSTEM KNOW WHAT DEVICE DRIVER PROCEDURES TO CALL?

5. HOW DOES A DEVICE DRIVER KNOW WHAT THE I/O PORT ADDRESSES ARE FOR ITS DEVICE?
WRITING CUSTOM DEVICE DRIVERS
CUSTOM DEVICE DRIVER PROCEDURES

- INIT$IO - DEVICE INITIALIZATION PROCEDURE
- FINISH$IO - DEVICE FINISH PROCEDURE
- QUEUE$IO - DEVICE QUEUE I/O REQUEST PROCEDURE
- CANCEL$IO - DEVICE CANCEL I/O PROCEDURE

YOU MUST WRITE THESE PROCEDURES AND AN INTERRUPT TASK AND HANDLER IF NEEDED
INIT$IO PROCEDURE

THIS IS CALLED BY THE I/O SYSTEM WHEN THE FIRST ATTACH DEVICE CALL IS MADE.

THE FORM OF THIS CALL IS:

    CALL INIT$IO(DUIB$P, D$DATA$P, STATUS$P);

DUIB$P    - A POINTER TO THE DUIB FOR THE DEVICE TO BE INITIALIZED

D$DATA$P  - A POINTER TO THE WORD WHERE YOU MUST STORE THE TOKEN FOR A SEGMENT OBJECT IF NEEDED BY YOUR DEVICE.

      THIS SEGMENT MAY CONTAIN DATA SUCH AS A REGION TOKEN FOR THE QUEUE, A POINTER TO THE FIRST IORS ON THE QUEUE AND A TOKEN FOR AN INTERRUPT TASK IF NEEDED.

STATUS$P  - A POINTER TO A WORD WHERE YOU MUST STORE THE RESULTS OF THIS CALL

NOTE: IF NO DATA OBJECT IS NEEDED YOU MUST RETURN ZERO AS A TOKEN.
A possible flow for this procedure might be:

1. CREATE A SEGMENT FOR A DATA OBJECT

2. CREATE A REGION FOR ACCESS TO A QUEUE

3. CREATE AN INTERRUPT TASK FOR THE DEVICE

4. SET THE QUEUE TO EMPTY

5. INITIALIZE THE DEVICE HARDWARE AND ANY VARIABLES NEEDED

6. IF ALL WENT WELL THEN SET STATUS TO E$OK
FINISH I/O PROCEDURE

THE I/O SYSTEM CALLS THIS PROCEDURE AFTER THE LAST DETACH DEVICE CALL IS MADE ON THIS DEVICE

THE FORM OF THE CALL IS:

```c
CALL FINISH$IO(DUIB$P, D$DATA$T);
DUIB$P      - A pointer to the DUIB for this device unit
D$DATA$T    - A token for the data object segment
```

THE FINISH I/O PROCEDURE MUST DO ANY FINAL PROCESSING ON THE DEVICE IF NEEDED AND DELETE ANY OBJECT THE INIT$IO PROCEDURE CREATED

(SEgment, REGION, RESET INTERRUPT TASK,...)
QUEUE$IO PROCEDURE

This procedure is called for every request to the device driver.

The form of the call is:

```c
CALL QUEUE$IO(IORS$T, DUIB$P, D$DATA$);
```

- IORS$T - a token for the I/O request segment

This procedure must do the following:

1. If the device is busy place the request on the queue

2. If the device is not busy then start the I/O function

3. If the request can be completed without placing the IORS on the queue then set the IORS.DONE field to true

Note: Whenever accessing the queue you must first gain access to it by receiving control of the region that protects it.
CANCEL $ IO

This procedure is called by the I/O system whenever a hard detach device system call is made or a job is deleted that still has requests pending.

The form of the call is:

CALL CANCE$IO(CANCEL$ID, DUIB$P, D$DATA$T);

CANCEL$ID - the ID for requests that are to be removed from the queue.

This procedure must remove any request from the queue that contain the cancel ID value.
IMPLEMENTING A I/O REQUEST QUEUE

WHEN WRITING CUSTOM DEVICE DRIVERS YOU MUST HAVE SOME SORT OF QUEUE FOR INCOMING REQUESTS.

THE IORS SEGMENT CONTAINS TWO FIELDS THAT ALLOW FOR A LINKED LIST

IORS.LINK$FOR, IORS.LINK$BACK

THESE TWO POINTER VALUES CAN BE USED TO IMPLEMENT A QUEUE

IF IN YOUR DATAOBJECT YOU HAVE A VALUE CALLED FIRST$IORS THAT IS SET TO ZERO TO INDICATE AN EMPTY QUEUE.

WHEN A REQUEST NEEDS TO BE QUEUED YOU CAN SET THIS FIELD TO POINT TO THE FIRST IORS AND THE LINK FIELDS OF THE IORS TO POINT BOTH FORWARD AND BACK IN THE QUEUE.
REQUEST QUEUE

FIRST IORS ON QUEUE
LINK$FOR
LINK$BACK

SECOND IORS ON QUEUE
LINK$FOR
LINK$BACK

THIRD IORS ON QUEUE
LINK$FOR
LINK$BACK

LAST IORS ON QUEUE
LINK$FOR
LINK$BACK

...
INTERRUPT TASKS

INTERRUPT TASKS ARE USED TO RESPOND TO THE INTERRUPT GENERATED BY THE DEVICE.

THE INTERRUPT TASK MUST DO THE FOLLOWING:

1. SERVICE THE INTERRUPT

2. DETERMINE IF THE REQUEST IS COMPLETE

3. IF COMPLETE, GAIN ACCESS TO THE QUEUE
   - REMOVE THE IORS FROM THE QUEUE.
   - SET THE IORS.DONE FIELD TO TRUE.
   - SEND THE IORS TO THE MAILBOX IN IORS.RMBX.
   - IF THE QUEUE IS NOT EMPTY THEN START THE NEXT REQUEST.

4. IF THE REQUEST IS NOT COMPLETE THEN INITIATE THE NEXT PROCESS.
QUIZ #3 - CUSTOM DEVICE DRIVERS

1. WHAT IS THE PURPOSE OF THE DATA OBJECT?

2. WHEN IS THE CANCEL I/O PROCEDURE CALLED?

3. IS IT POSSIBLE TO USE DEFAULT I/O PROCEDURES WITH CUSTOM DEVICE DRIVER?

4. WRITE A PROCEDURE TO PLACE REQUESTS ON THE QUEUE AND ONE TO REMOVE A REQUEST FROM THE QUEUE.
   (ASSUME THAT YOU ALREADY HAVE ACCESS TO THE QUEUE)
LINKING DEVICE DRIVERS TO THE I/O SYSTEM

AFTER YOU HAVE WRITTEN YOUR DEVICE DRIVER CODE YOU MUST LINK IT TO THE I/O SYSTEM.

THE FOLLOWING COMMAND CAN BE USED TO ACCOMPLISH THIS:

```bash
LINK 86
    :FO: IOS.LIB(ISTART),   &
    :F1: ITABLE.OBJ,        &
    :F1: IDEVCF.OBJ,        &
    :F1: DRIVER.OBJ,        &
    :FO: IOOPT1.LIB,        &
    :FO: IOS.LIB,           &
    :FO: RPIFC.LIB,         &
    TO: F1: IOS.LNK         (LINKER OPTIONS)
```
CONFIGURING INTO THE I/O SYSTEM

To configure your device drivers into the I/O system you must add the necessary device driver interface structures to the file IDEVCF .A86.

This consists of adding DVIIB's for each device unit and the required common and unit info. blocks as needed.
CHAPTER 7

EXTENDED INPUT/OUTPUT SYSTEM (EIOS)
REVIEW QUIZ

NAME 3 FILE TYPES

__________________________

__________________________

__________________________
TERMINOLOGY

- I/O USER

- USER OBJECT

- DEVICE

- DEVICE CONNECTION

- FILE

- ACCESS RIGHTS

- FILE CONNECTION
BASIC I/O SYSTEM
INTERACTION SEQUENCE

1. OBTAIN USER TOKEN USING A STRUCTURE OF USER
   ID AND ALIASES

   USERTKN = RQCREATEUSER (@ STRUCT, @ STATUS);
   /* TEST STATUS */

2. OBTAIN DEVICE CONNECTION TOKEN USING THE
   PHYSICAL DEVICE NAME

   CALL RQA PHYSICAL ATTACH DEVICE (DEV NAME, FILE
   DRIVER, RESPMBX, @ STATUS);
   /* TEST STATUS TO CHECK SYNCHRONOUS PORTION OF
      CALL */
   TKN = RQRECEIVE MESSAGE (MBX, TIME, , @ STATUS);
BASIC I/O SYSTEM
INTERACTION SEQUENCE

2. (CONTINUED)
   /* CHECK TOKEN RECEIVED. IF TYPE = 101H YOU HAVE A CONNECTION. IF TYPE = 6, YOU HAVE A PROBLEM */

3. OBTAIN FILE CONNECTION USING THE DEVICE CONNECTION TOKEN, USER TOKEN, AND A FILE NAME SUBPATH
   CALL RQA ATTACH FILE (USER, DEVTKN, SUBPATH, ,@ STATUS);
   /* TEST STATUS TO CHECK SYNCHRONOUS PORTION OF CALL */
   TKN= RQ RECEIVE MESSAGE (MBX1, TIME, ,@ STATUS);
   /* CHECK TOKEN TYPE. IF TYPE 101H YOU HAVE A FILE CONNECTION. IF TYPE = 6 YOU HAVE A PROBLEM */
BASIC I/O SYSTEM
INTERACTION SEQUENCE

4. OPEN FILE FOR USAGE USING THE FILE CONNECTION
   AND THE MODE AND SHARING METHOD
   
   CALL RQA OPEN (CONN, MODE, SHARING, RESPMBX, @ status);
   /* TEST STATUS TO CHECK SYNCHRONOUS PORTION OF CALL */
   MSGTKN = RQ RECEIVE MESSAGE (RESPMBX, TIME, , @ status);
   /* TEST STATUS FIELD OF IORS RETURNED TO CHECK
      ASYNCHRONOUS PORTION OF CALL. */

   /* FINALLY YOU CAN READ OR WRITE!! */
DIVERSION: THE CONNECTION BASIC I/O STYLE

TO ACCESS A FILE WE MUST HAVE A CONNECTION TO IT

WE GENERALLY OBTAIN THIS CONNECTION IN TWO STEPS

1. OBTAIN DEVICE CONNECTION USING:
   RQA PHYSICAL ATTACH DEVICE
   PASS DEVICE NAME
   RECEIVE TOKEN
DIVERSSION: THE CONNECTION
BASIC I/O STYLE

2. OBTAIN FILE CONNECTION USING
   RQA ATTACH FILE
   PASS PREFIX (USUALLY THE DEVICE TOKEN),
   AND SUBPATH
   RECEIVE FILE CONNECTION TOKEN

YOU NOW USE THE FILE CONNECTION TOKEN FOR
ALL FUTURE INTERACTION WITH THE FILE.
EXAMPLE CONTINUED

LET'S SAY THE DEVICE HAS BEEN ATTACHED TO AND WE HAVE ITS TOKEN.

POSSIBILITIES:

TO GET TO JILL:

PREFIX

TOKEN + SUBPATH

A/E/JILL

OR

STEP 1 TOKEN + A = NEW TOKEN

STEP 2 NEW TOKEN + E/JILL

(TRY SOME OTHERS !)
EIOS TERMINOLOGY

IN ADDITION TO THE BASIC I/O SYSTEM TERMINOLOGY WE ADD:

• LOGICAL NAMES

• I/O JOBS

• DEFAULT PREFIX AND PATH PTR PARAMETERS
THE LOGICAL DEVICE NAME

**DEFINITION:** A name attached to a physical device at configuration or run time which has more meaning to the user.

**EXAMPLES:**

<table>
<thead>
<tr>
<th>PHYSICAL</th>
<th>LOGICAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>FØ</td>
<td>:FØ:</td>
</tr>
<tr>
<td>FX1</td>
<td>:HD FLOPPY:</td>
</tr>
<tr>
<td>WD1</td>
<td>:WINNY:</td>
</tr>
<tr>
<td>FØ</td>
<td>:SYSTEM:</td>
</tr>
<tr>
<td>F1</td>
<td>:PATIENT:</td>
</tr>
</tbody>
</table>
TWO WAYS TO CREATE A LOGICAL DEVICE NAME

ONE

• USE RQA PHYSICAL ATTACH DEVICE
  PASS PHYSICAL DEVICE NAME
  RECEIVE TOKEN

• CATALOG THE TOKEN USING RQA CATALOG CONNECTION
  PASS TOKEN, LOGICAL NAME, JOB

(WITH THIS METHOD YOU CAN CATALOG THE CONNECTION IN ANY JOBS DIRECTORY)
TWO WAYS TO CREATE A LOGICAL DEVICE NAME

TWO

1. USE RQA LOGICAL ATTACH DEVICE
   PASS LOGICAL NAME, DEVICE NAME

   (LOGICAL DEVICE OBJECT IS CATALOGED IN THE ROOT
   JOB UNDER THE LOGICAL NAME)

   (NOTE: THE LOGICAL DEVICE OBJECT IS NOT A CONNECTION.
   THE EIOS WILL CREATE A DEVICE CONNECTION
   DURING THE FIRST EIOS CALL THAT USES THE
   LOGICAL NAME)
THE LOGICAL FILE NAME

DEFINITION: A NAME ATTACHED TO A FILE CONNECTION AT RUNTIME FOR USE OF USER.

EXAMPLES: : OUR_DATA: : MY_DIRECTORY: : A:

(MORE ON THIS IN A MINUTE!)
THE I/O JOB

TO USE EIOS CALLS YOUR TASK MUST BE RUNNING IN AN I/O JOB.

DEFINITION: AN I/O JOB IS AN RMX-86 JOB WITH THREE EXTRA ATTRIBUTES.

1. A CATALOG ENTRY IN ITS OWN DIRECTORY UNDER THE NAME "RQ GLOBAL" (JOB TOKEN)
2. A CATALOG ENTRY OF A CONNECTION UNDER THE NAME "$" (DEFAULT PREFIX)
3. A CATALOG ENTRY OF A USER TOKEN UNDER THE NAME $USER (DEFAULT USER)
To create an I/O job

• Create AT system configuration using the EI05 macro.

• Use the create I/O job system call during run time.

Problem: This call can only be made from a task running in an I/O job.
LOGICAL NAMES AND PATHS

DATABASE: V φ 1983/MAY
JACK: SSNO/341-0273-22
TEMP: A/VOL1/TEMP
DEFAULT PREFIX

• PURPOSE: REDUCE PROGRAMMER EFFORT AND ERRORS BY ALLOWING REFERENCE TO A DEFAULT CONNECTION (TO A FILE OR DEVICE) WHICH IS CATALOGED IN THE I/O JOB DIRECTORY.

• EXAMPLE: A PARTICULAR I/O JOB MUST FREQUENTLY ACCESS A DATA FILE. OBTAIN THE CONNECTION FOR THE FILE AND CATALOG IT IN THE I/O JOB DIRECTORY UNDER "$". AFTER THIS IS DONE ANY ATTACH FILE A CALL WITH A NULL PATH WILL AUTOMATICALLY ATTACH TO THE DATA FILE.
CREATING A LOGICAL FILE NAME

1. ATTACH TO A DEVICE

2. RECEIVE TOKEN  (OPTIONAL: CATALOG AS A LOGICAL DEVICE)

3. ATTACH TO THE DESIRED FILE
   - RQS ATTACH FILE
   - PASS PATH NAME STRING
   - RECEIVE CONNECTION

4. CATALOG THE CONNECTION
   - RQS CATALOG CONNECTION
   - PASS CONNECTION, JOB, LOGICAL NAME
PATH NAME STRING
4, COUNT'EM, 4 FLAVORS!

STRING PASSED
NULL
LOGICAL NAME ONLY
SUBPATH ONLY
LOGICAL NAME + SUBPATH

EIOS ACTION
USE DEFAULT PREFIX
USE PATH CATALOGED
DEFAULT PREFIX + SUBPATH
USE PATH CATALOGED TO
GET TO DIRECTORY THEN
FOLLOW SUBPATH FROM THERE
PATH NAME STRINGS
(EXAMPLES)

• NULL

ASSUME DEFAULT PREFIX IS:

FØ/A/B

PASS NULL FOR ATTACH FILE AND GET CONNECTION TO FØ/A/B

• LOGICAL NAME ONLY

ASSUME :DATABASE: IS THE LOGICAL NAME FOR:

WD1/TUE/SECOND/DATA

PASS THE LOGICAL STRING :DATABASE:
GET CONNECTION TO DATA FILE
PATH NAME STRINGS
(EXAMPLES)

• SUBPATH ONLY
  ASSUME DEFAULT PREFIX : FØ:
  POINTS TO FØ (DEVICE NAME)
  PASS SUBPATH Q/1979/FILE 1

• LOGICAL NAME + SUBPATH
  ASSUME LOGICAL NAME : PATIENT:
  POINTS TO FØ/1979/PATIENT
  PASS SUBPATH : PATIENT: JACK/STRANGE
  RECEIVE CONNECTION TO:
  FØ/1979/PATIENT/JACK/STRANGE
THE DEFAULT USER

to attach to a file you need

• PATH STRING

• USER ID
THE DEFAULT USER

THE EXTENDED I/O SYSTEM ATTACH CALL HAS ONLY 2 PARAMETERS, PATH PTR AND STATUS

HOW DOES THE USER ID GET PASSED?

SIMPLE, THE E10S USES THE DEFAULT USER ID OF THE JOB (IO JOB OF COURSE) THAT CONTAINS THE CALLING TASK!
EIOS INTERACTION SEQUENCE

1. ATTACH TO A DEVICE USING A LOGICAL NAME, PHYSICAL NAME AND FILE DRIVER DESIRED

   CALL RQLOGICAL ATTACHDEVICE (@ (9, ':PATIENT:'),
                              @ (2, 'FØ'), 4,
                              @ STATUS);

   FØ IS IN THE SYSTEM DUIB'S

2. ATTACH TO THE FILE SPECIFYING THE PATH NAME

   CONNTKN = RQS ATTACH FILE (@ (12, 'JONES/ROBERT'),
                            @ STATUS);
EIOS INTERACTION SEQUENCE

3. OPEN THE FILE SPECIFYING THE MODE AND NUMBER OF BUFFERS DESIRED

CALL RQS OPEN(CONNTKN, 3, 2);
EOIS CALLS OVERVIEW

- RELATING TO LOGICAL NAMES
  - RQS CATALOG CONNECTION
  - RQS LOOKUP CONNECTION
  - RQS UNCATALOG CONNECTION

- CREATE FILE OR CONNECTION
  - RQS ATTACHE FILE
  - RQS CREATE DIRECTORY
  - RQS CREATE FILE
EOIS CALLS OVERVIEW

• DATA MANIPULATION
  • RQS OPEN
  • RQS CLOSE
  • RQS READMOVE
  • RQS SEEK
  • RQS WRITEMOVE
  • RQS TRUNCATE FILE

• DEVICE RELATED CALL
  • RQS SPECIAL
EOIS CALLS OVERVIEW

• CHANGING ACCESS, RENAMING, OBTAINING STATUS
  • RQS CHANGE ACCESS
  • RQS RENAME FILE
  • RQS GET CONNECTION STATUS
  • RQS GET FILE STATUS

• DELETING FILES AND CONNECTIONS
  • RQS DELETE CONNECTION
  • RQS DELETE FILE
EOIS CONFIGURATION

• SELECT THE EOIS CALLS TO BE INCLUDED IN THE FINAL SYSTEM

• SELECT THE LOGICAL DEVICES TO BE INITIALIZED IN THE FINAL SYSTEM

• CREATE THE INITIAL I/O JOB(S) IN THE SYSTEM
## E01S Configuration

<table>
<thead>
<tr>
<th>Files</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETABLE.A86</td>
<td>SYSTEM CALLS</td>
</tr>
<tr>
<td>EDEVCF.A86</td>
<td>LOGICAL DEVICES</td>
</tr>
<tr>
<td>EJOBFCF.A86</td>
<td>% JOB</td>
</tr>
</tbody>
</table>
ETABLE A86

NAME

$INCLUDE (:F2:ETABLE.MAC)

; ; JOB INTERFACE ;
; %RQCREATEIOJOB
%RQEXITIOJOB

; ; CONFIGURATION INTERFACE ;
; %RQLOGICALATTACHDEVICE
%RQLOGICALDETAHCDEVICE

; ; SYNCHRONOUS INTERFACE ;
; %RQSCREATEFILE
%RQSATTACHFILE
%RQSDISTRIBUTECONNECTION
%RQSLOOKUPCONNECTION
%RQSCATALOGCONNECTION
%RQSUNCATALOGCONNECTION
%RQSCREATEDIRECTORY
%RQSDISTRIBUTEFILE
%RQSRENAMEFILE
%RQSCONNECTACCESS
%RQSOPEN
%RQSCLOSE
%RQSSREADMOVE
%RQSGWRITEMOVE
%RQSSSEEK
%RQSTRUNCATEFILE
%RQSGETFILESTATUS
%RQSGETCONNECTIONSTATUS
%RQSSPECIAL

END
EDEVCF.A86

$ INCLUDE

LOGICAL DEVICE SELECTION

% END.DEV.CONF16

END
$INCLUDE(:F2:EDEVCF.MAC)

; BYTE-BUCKET
; %DEV_INFO_BLOCK('BB','BB',PHYSICAL)
;
; TERMINAL
; %DEV_INFO_BLOCK('T0','T0',PHYSICAL)
;
; SHUGART 204, UNIT 0, DRIVE 0
; %DEV_INFO_BLOCK('F0','F0',NAMED)

; SHUGART 204, UNIT 1, DRIVE 1
; %DEV_INFO_BLOCK('F1','F1',NAMED)
;
; 218 WINCHESTER FLOPPY SS/SD, UNIT 0, DRIVE 0
; %DEV_INFO_BLOCK('WF0','WF0',NAMED)
;
; 218 WINCHESTER FLOPPY SS/SD, UNIT 1, DRIVE 1
; %DEV_INFO_BLOCK('WF1','WF1',NAMED)
;
; STREAM
; %DEV_INFO_BLOCK('STREAM','STREAM',STREAM)

%END_DEV_CONFIG(1024)

END
EJOBCF.A86

$ INCLUDE

% IO_USER_MACROS

% IO_JOB_MACROS

% END_IO_JOB_MACROS

END
NAME
CGROUP

$INCLUDE(:F2:EJOBCF.MAC)

; ; USER 'WORLD' DEFINITION
; ; %IO_USER('WORLD', 0FFFFH)
; ; EIOS TEST JOB
; ; %IO_JOB('T0', 'WORLD', 260H, 0FFFFH, 0:0, 0, 0, 155, 1800:0, 1A00, 0:0, 1200, 0)

%END_IO_JOB_CONFIG(40)

END

NOTE: THE CONFIGURED IO_JOB IN THE RELEASE FILE IS FOR THE HUMAN INTERFACE.
ASSEMBLING, LINKING AND LOCATING THE EIos.
(THE IS TOUGH, SO PAY ATTENTION!!)

SUBMIT :fx: EIos(DATE, LOC_ADR)

BEFORE DOING THIS SUBMIT YOU SHOULD PRINT
THE FILE ON A TERMINAL OR A HARD COPY TO
INSURE THAT THE FILE WILL NOT CALL FOR
RESOURCES THAT YOU DO NOT HAVE.
ADDING THE EIOS TO THE SYSTEM

• ONE JOB MACRO REQUIRED AT SYSTEM CONFIGURATION TIME.

• PARAMETERS FOR MACRO ARE FOUND IN THE iRMX-86 CONFIGURATIONS GUIDE.
CHAPTER QUIZ!

1. Give a physical device name. ____________

2. Give a logical device name. ____________

3. What are the characteristics of an I/O job?
   ____________
   ____________
   ____________

4. What is the "gotcha" in the creation of an I/O job? ________________
CHAPTER QUIZ (CONT.)

5. WHAT IS A...
   A. DEFAULT USER
   B. DEFAULT PREFIX

6. MATCH THE FOLLOWING
   A. ETABLE.A86  ____ LOGICAL DEVICES
   B. EJOBBCF.A86 ____ SYSTEM CALL SELECTION
   C. EDEVCF.A86  ____ IO JOB CREATION
THE HUMAN INTERFACE
OVERVIEW

THE HUMAN INTERFACE IS A LAYER OF THE RMX-86 SYSTEM THAT ALLOWS THE OPERATOR TO LOAD, EXECUTE AND SUBSEQUENTLY INTERACT WITH PROGRAM FILES.
RMX-86 AND THE HUMAN INTERFACE
RESIDENT/NON-RESIDENT PROGRAMS

RESIDENT: PROGRAMS LOADED AT SYSTEM RESET WHICH REMAIN IN MAIN MEMORY. (COULD BE IN ROM)

NON-RESIDENT: PROGRAMS WHICH ARE LOADED INTO MAIN MEMORY FROM SECONDARY STORAGE UPON PROGRAM OR OPERATOR COMMAND
SOME EXAMPLES

SYSTEM/RESIDENT: APPLICATION LOADER, EXTENDED I/O SYSTEM
SYSTEM/NON-RESIDENT: COPY, DIR, DELETE
USER/RESIDENT: DATA COLLECTION, INTERRUPT DRIVEN TASKS.
USER/NON-RESIDENT: DATA REDUCTION, DATA ANALYSIS PROGRAM.
HUMAN INTERFACE
SERVICES

• NON-RESIDENT COMMANDS

• RESIDENT SYSTEM SERVICES
NON-RESIDENT COMMANDS

• FILE MANIPULATION
  • ATTACH DEVICE
  • CREATE DIR
  • DETACH DEVICE
  • DOWNCOPY
  • RENAME
  • COPY
  • DELETE
  • DIR
  • FORMAT
  • UPCOPY

• GENERAL UTILITY
  • DATE
  • SUBMIT
  • DEBUG
  • TIME
HUMAN INTERFACE
COMMAND SYNTAX

COMMAND: # INPATHLIST [PREPOSITION OUTPATHLIST] [PARAMETERS]

WHERE:

INPATHLIST = ONE OR MORE FILES TO BE USED AS INPUT DURING COMMAND EXECUTION
PREPOSITION = HOW YOU WANT OUTPUT HANDLED
OUTPATHLIST = ONE OR MORE FILES TO RECEIVE OUTPUT DURING COMMAND EXECUTION
PARAMETERS = REQUESTED OPTIONAL SERVICES
PATH LISTS

PATHNAME [ , PATHNAME] . . .

EXAMPLES:

MY FILE/DATA
YOUR FILE/1979/DATA, JACKFILE/samp 1
A/B, A/C, A/D, E/Q/2
PREPOSITIONS

TO-OUTPUT TO NEW FILE
   (IF OLD FILE IS SPECIFIED, A QUERY RESULTS)

OVER-OUTPUT TO OLD FILE OVER OLD DATA
   (WHETHER OR NOT TARGET FILE EXISTS)

AFTER-OUTPUT APPENDED AFTER DATA IN TARGET FILE
   (WHETHER OR NOT TARGET FILE EXISTS)

AS - ASSOCIATES A PHYSICAL DEVICE TO A LOGICAL
   NAME (ONLY FOR THE ATTACH DEVICE COMMAND)
## CONTROL CHARACTERS

<table>
<thead>
<tr>
<th>CHARACTER</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑Z</td>
<td>END OF FILE</td>
</tr>
<tr>
<td>↑C</td>
<td>PROGRAM ABORT</td>
</tr>
<tr>
<td>↑D</td>
<td>INVOKE DEBUGGER</td>
</tr>
<tr>
<td>↑O</td>
<td>SUPPRESS/RESTORE OUTPUT</td>
</tr>
<tr>
<td>↑S</td>
<td>SUSPEND OUTPUT</td>
</tr>
<tr>
<td>↑Q</td>
<td>RESUME OUTPUT</td>
</tr>
<tr>
<td>↑X</td>
<td>DELETE CURRENT INPUT LINE</td>
</tr>
<tr>
<td>↑R</td>
<td>REPEAT CURRENT LINE OR PREVIOUS LINE IF CURRENT LINE IS EMPTY</td>
</tr>
</tbody>
</table>
COPY

COPY DATA FROM INPUT FILE(S) TO OUTPUT FILE(S)
DATE

SET OR DISPLAY CURRENT DATE

dd mmm yy
DIR

LIST THE NAMES AND ATTRIBUTES OF FILES IN A SELECTED DIRECTORY.

DIR - INPATH - TO - OVER - OUT PATH

EXTENDED - LONG - FAST - ONE - QUERY

SHORT
FORMAT

FORMAT OR REFORMAT A VOLUME ON A SECONDARY STORAGE DEVICE (DISK, DISKETTE, BUBBLE)
RESIDENT SYSTEM SERVICES

• I/O PROCESSING CALLS

• COMMAND PARSING CALLS

• MESSAGE PROCESSING CALLS

• COMMAND PROCESSING CALLS

• PROGRAM CONTROL CALL
I/O PROCESSING CALLS

• C GET INPUT CONNECTION
  PASS INPUT PATHNAME
  RETURN EIOS CONNECTION

• C GET OUTPUT CONNECTION
  PASS OUTPUT PATHNAME
  RETURN EIOS CONNECTION
COMMAND PARSING CALLS

- **CGET** INPUT PATHNAME
  RETURN PATHNAME FOR STANDARD INTO FILE

- **CGET** OUTPUT PATHNAME
  RETURN PREPOSITION AND PATHNAME FOR STANDARD OUTPUT FILE

- **CGET** PARAMETER
  RETURN NEXT PARAMETER FROM INPUT LINE AS KEYWORD NAME AND VALUE
COMMAND PARSING CALLS

- C SET PARSEBUFFER
  SWITCH TO NEW BUFFER
MESSAGE PROCESSING CALLS

• C FORMAT EXCEPTION
  PASS EXCEPTION CODE
  RETURN MESSAGE IN USER BUFFER

• C SEND COMMAND RESPONSE
  SEND MESSAGE TO COMMAND OUTPUT
  READ RESPONSE FROM COMMAND INPUT

• C SEND EO RESPONSE
  SEND MESSAGE TO ERROR OUTPUT
  READ RESPONSE FROM ERROR INPUT
COMMAND PROCESSING CALLS

- **C CREATE COMMAND CONNECTION**
  
  RETURN COMMAND CONNECTION TOKEN

- **C DELETE COMMAND CONNECTION**
  
  PASS COMMAND CONNECTION TOKEN
  
  DELETE CONNECTION

- **C SEND COMMAND**
  
  RECEIVE COMMAND LINES FROM CONSOLE
  
  SEND TO COMMAND DATA SPACE AND EXECUTE
PROGRAM CONTROL CALL

• C SET CONTROL C

SEND NEW CONTROL-C SEMAPHORE TOKEN
HOW DOES ALL OF THIS WORK?

PHASE 1. COMMAND LINE INTERPRETER PARSES THE COMMAND LINE TO BREAK OUT THE PATHNAME TO THE PROGRAM FILE.

JACK/PROG 1 ➔ :PROG:JACK/PROG 1 (FIRST)
OR :SYSTEM:JACK/PROG 1 (SECOND)

HOW DOES ALL OF THIS WORK?

PHASE 2. PROGRAM EMPLOYS HI COMMANDS TO CARRY OUT ITS OWN PROCESSING.

EXAMPLE: PROGRAM TO ENCODE A DATA FILE

GET INPUT PATHNAME
GET OUTPUT PATHNAME
GET INPUT CONNECTION
GET OUTPUT CONNECTION
PROCESS FILE
DELETE INPUT CONNECTION
DELETE OUTPUT CONNECTION
EXIT I/O JOB
CREATING A NEW CUUP
(COMMONLY USED USER PROGRAM)

1. WRITE THE PROGRAM

2. ASSEMBLE OR COMPILE THE PROGRAM

3. LINK CODE TO APPROPRIATE RMX-86 LIBRARIES
   USE BIND, NOINITCODE AND MEMPOOLEDIRECTIVES
   TO CREATE LTL OR PIC MODULE [SERIES III]

   - OR -
CREATING A NEW CUUP

3. USE LINK AND LOCATE WITH NO.INIT. CODE AND MEMPOOL DIRECTIVES TO CREATE AN ABSOLUTE MODULE. (THERE MUST BE RESERVED SPACE IN WHICH TO LOAD IT!) (SERIES III)

-OR-

3. USE LINK AND LOCATE ON A SERIES II TO CREATE AN ABSOLUTE MODULE (ONLY)
CREATING A NEW CVUP

4. Place program in an appropriately named file in either the :SYSTEM: directory or the :PROG: directory.
THE COMMAND CONNECTION
OR
THE ULTIMATE SUBROUTINE

PROBLEM: I HAVE A PROGRAM WHICH WILL
COPY, PROCESS, SORT AND FURTHER
PROCESS A FILE OF DATA, I HAVE
A SYSTEM COPY AND SORT ALREADY
AND WOULD LIKE TO USE THEM LIKE:

MY PROG
GET PARMS FROM USER
COPY

PROCESS

SORT

FINISH PROCESS

END
THE COMMAND CONNECTION

- A BOND BETWEEN YOUR PROGRAM AND THE COMMAND LINE EXECUTOR.
- USED WHEN YOUR PROGRAM WANTS TO SEND A COMMAND LINE TO BE EXECUTED.
- CAN BE ESTABLISHED ONCE AT PROGRAM START AND USED THROUGHOUT THE PROGRAM RUN.
**SEND COMMAND**

- A SYSTEM PROGRAM TO MOVE A BUFFER OF DATA (A COMMAND) TO THE COMMAND CONNECTION.

- IF BUFFER CONTAINS A CONFIGURATION CHARACTER, SEND COMMAND RETURNS IMMEDIATELY OTHERWISE IT RETURNS AFTER COMMAND IS EXECUTED.
AN EXAMPLE

OUR SORT PROGRAM.

PROGRAM INVOCATION:

FAST SORT

:FI: JACK/DATA

TO :FI: JACK/SORTED

USER COMMAND

INPUT FILE

OUTPUT FILE
AN EXAMPLE

1. GET INPUT PATHNAME INTO A PRIVATE BUFFER

2. GET OUTPUT PATHNAME INTO A PRIVATE BUFFER

3. CREATE COMMAND CONNECTION

4. FORMAT COPY COMMAND IN PRIVATE COMMAND BUFFER USING INPUT AND OUTPUT PATHNAMES

5. SEND ASSEMBLED COMMAND TO COMMAND CONNECTION

COPY PROGRAM RUNS
AN EXAMPLE

6. PROCESS COPIED DATA

7. FORMAT SORT COMMAND IN PRIVATE COMMAND BUFFER AGAIN USING INPUT AND OUTPUT PATHNAMES

8. SEND ASSEMBLED COMMAND TO THE COMMAND CONNECTION

9. DELETE COMMAND CONNECTION

10. FINISH PROCESSING AND EXIT
ANOTHER USE

SINCE THE PRIVATE COMMAND BUFFER COULD BE FILLED FROM ANY SOURCE, IMAGINE...

1. READ A FILE INTO COMMAND BUFFER

2. SEND COMMAND

3. REPEAT FOREGOING AS LONG AS "DATA" EXISTS IN THE FILE.

WHAT DOES THIS REMIND YOU OF?
HUMAN INTERFACE CONFIGURATION

- DESIGNATE PATHNAMES FOR THE LOGICAL NAMES REQUIRED BY THE HUMAN INTERFACE

- SPECIFY THE SIGN ON MESSAGE

- SPECIFY THE MAXIMUM COMMAND NAME LENGTH

- SPECIFY THE DIRECTORIES AND THE SEQUENCE THAT THE HUMAN INTERFACE WILL SEARCH THEM IN FOR USER PROGRAMS
PATHNAME - LOGICAL NAME
SPECIFICATION

- FOUR DIRECTORIES - SYSTEM
  PROG
  DEFAULT
  WORK

- LOGICAL DEVICE NAME (:FØ: IN SUPPLIED FILE)
  MUST BE CONFIGURED IN THE EXTENDED I/O SYSTEM
THE SIGN ON MESSAGE

• MAXIMUM LENGTH IS 255 CHARACTERS

• ESSENTIALLY "ANYTHING GOES!"
  (WITHIN THE BOUNDS OF GOOD TASTE, OF COURSE)

• SOME EXAMPLES
  “JACLYN SYSTEM 2000 V1.0”
  “WORDCRUSHER V2.9 JOEN MFG COPYRIGHT 1987”
COMMAND NAME LENGTH

- THEORETICALLY COULD BE $2^{16} - 1$

- HOWEVER, A SINGLE LINE (80) MAKES A BIT MORE SENSE.
DIRECTORIES AND SEARCH SEQUENCE

• A MAXIMUM OF 255 DIRECTORIES CAN BE AUTOMATICALLY SEARCHED

• USER SUPPLIES A STRING TABLE OF NAMES

• SYSTEM SEARCHES DIRECTORIES IN SEQUENCE GIVEN.

• IN ALL CASES THESE DIRECTORIES MUST BE CONFIGURED IN THE EXTENDED I/O SYSTEM (MUST EXIST BEFORE THE HUMAN INTERFACE BEGINS RUNNING)
LINKING AND LOCATING THE HUMAN INTERFACE
(ANOTHER BIG ONE)

• SUBMIT :fx: HI(DATE, LOC)

• WHERE DATE = MM/DD/YY OR DD MMM YY
  LOC = LOCATION OF HUMAN INTERFACE WHEN SYSTEM IS LOADED.
HUMAN INTERFACE PREREQUISITS

- NUCLEUS
- DEBUGGER OR TERMINAL HANDLER
- BASIC I/O SYSTEM
- EXTENDED I/O SYSTEM
- APPLICATION LOADER

IN ALL CASES ABOVE THE CALLS REQUIRED BY THE HUMAN INTERFACE MUST BE CONFIGURED.
TERMINAL HANDLER REQUIREMENTS

HUMAN INTERFACE

:TO:

BASIC % SYSTEM

RQTHNORMIN RQTHNORMOUT

TERMINAL HANDLER
TERMINAL HANDLER

REQUIREMENTS

• IF YOU WANT TO USE TC MODULE FROM HUMAN INTERFACE FOR PROGRAM CONTROL (ABORT)

MODIFY

MTH.CSD or
DB.CSD

ADD

:FX: HI.LIB(HCONTC), &
BASIC I/O SYSTEM REQUIREMENTS

• FILE DRIVERS - PHYSICAL STREAM NAMED

• DUIBS - TØ (TERMINAL DEVICE) BB (BYTE BUCKET) STREAM (STREAM FILE DEVICE) ? (ANY DISK OR BUBBLE DEVICES REQUIRED)

• DEVICE DRIVERS FOR ALL DUIBS
EXTENDED % SYSTEM REQUIREMENTS

• CONFIGURATION FILE (EDEVCF.A86) MUST INCLUDE:
  TØ
  BB
  STREAM

• I/O JOB FILE (EJOBCF.A86) MUST INCLUDE AN I/O JOB
  MACRO FOR THE HUMAN INTERFACE

• MEMORY POOL FOR EI0S MUST BE LARGE ENOUGH TO
  INCLUDE THE HUMAN INTERFACE
CHAPTER QUIZ

1. WHAT IS AN EXAMPLE OF A NON-RESIDENT USER PROGRAM?

2. GIVE 2 NON-RESIDENT USER COMMANDS

3. WHAT IS THE EFFECT OF THE AFTER PREPOSITION?

4. WHAT IS THE DIFFERENCE BETWEEN ↑O AND ↑S?
CHAPTER QUIZ!

5. WHAT 2 CALLS CAN BE USED TO GET AN INPUT CONNECTION FROM THE COMMAND LINE?

__________________

__________________

6. WHAT IS A COMMAND CONNECTION?

__________________

7. WHAT IS THE FILE FOR HUMAN INTERFACE CONFIGURATION?

__________________
INTEL WORKSHOPS

Introductory Workshops
- Introduction to Microcomputers (4 days)
- Intellic Development Systems (3 days)
- Operating Systems Fundamentals (2 days)

8080, 8085 System Design Workshops
- MCS-80/85 System Design (5 days)
- PASCAL Programming (5 days)
- PL/M Programming (5 days)
- iRMX 88.80 Operating Systems (5 days)

8086, 8088 System Design Workshops
- iAPX 86.88 System Design (5 days)
- iAPX 86.88 Advanced Assembly Language (5 days)
- ICE 86.88 and iAPX 86/21 (5 days)
- iAPX 286 Architecture (3 days)
- PASCAL Programming (5 days)
- PL/M Programming (5 days)
- iRMX 88.80 Operating System (5 days)
- iRMX 86 Operating System (5 days)
- iRMX 86 I/O Operating System (5 days)

432 Workshops
- iAPX 432 Architecture (3 days)
- Ada Programming (4 days)

Programming Language Workshops
- PASCAL Programming (5 days)
- PL/M Programming (5 days)
- iAPX 86.88 Advanced Assembly Language (5 days)
- Ada Programming (4 days)

Single Chip Microcomputer Workshops
- MCS 48/49 System Design (5 days)
- MCS-51 Microcontroller (5 days)
- 2920 Signal Processor (5 days)

Peripheral Chips Design Workshops
- Data Communication Chips (4 days)
- IEEE-488 GPIB Chips (3 days)
- Bubble Memory Design (3 days)