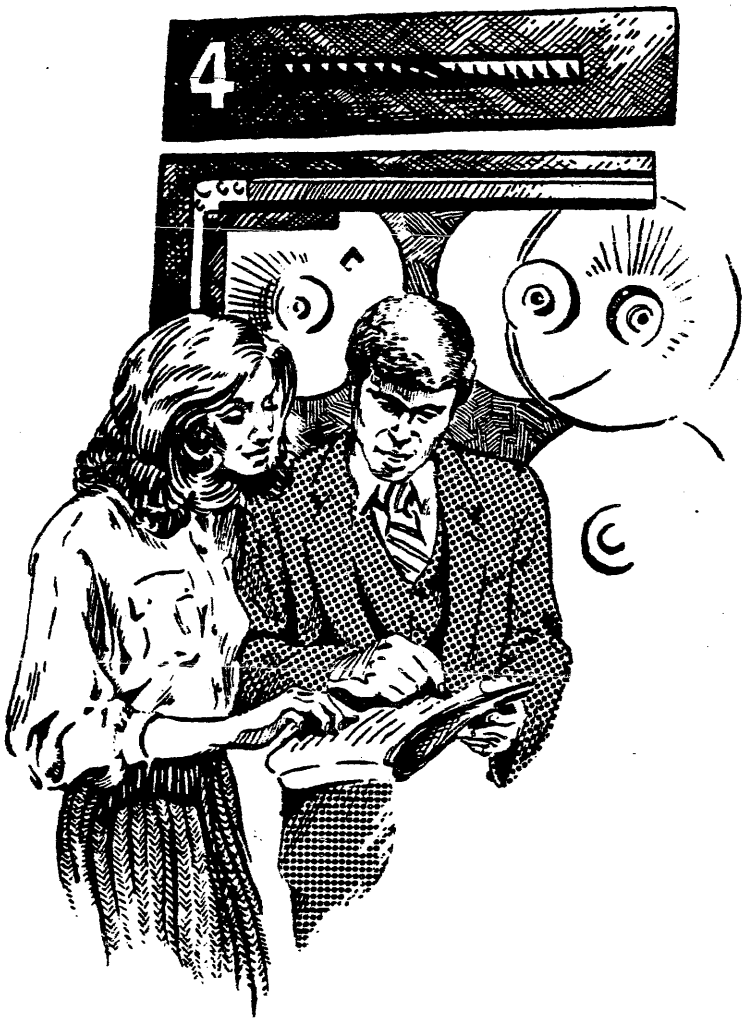


Level 68

Multics Subsystem Programming

Student Handbook

Course Code F15D

MAY 1981

Level 68

Multics Subsystem Programming

Student Handbook

Course Code F15D

ISSUE DATE: May 1, 1981

REVISION: 0

REVISION DATE:

Copyright (c) Honeywell Information Systems Inc., 1981

The information contained herein is the exclusive property of Honeywell Information Systems, Inc., except as otherwise indicated, and shall not be reproduced, in whole or in part, without explicit written authorization from the company.

Printed in the United States of America
All rights reserved

COURSE DESCRIPTION

F15D Multics Subsystem Programming

Duration: Five Days

Intended For: Advanced Multics PL/I programmers, familiar with standard Multics subroutines, who need to use advanced Multics subsystem writer's tools.

Synopsis: This intensive course describes how to bypass, replace, or supplement the standard Multics user interface by using system subroutines. Interprocess communication, tailoring the command environment, the message segment facility, the Multics ring mechanism, writing gates, dialing terminals to a process and writing I/O modules are among the topics covered in this course.

Interactive workshops are included to reinforce the material presented.

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

1. Use subsystem writer's subroutines.
2. Use a wide variety of facilities to create an environment tailored to the needs of a particular group of users.
3. Understand the conventions compilers should follow when creating object segments.
4. Understand how gates and I/O modules are written.

Prerequisites: Multics Concepts and Utilization (F01), Prerequisite Concepts for Programming on Multics (F10), Introductory Multics PL/I Programming (F15A), Advanced Multics PL/I Programming (F15B), PL/I Programming with Multics Subroutines (F15C) or equivalent experience.

Major Topics:
Writing I/O Modules
Interprocess Communication, Locking, and Timers
Advanced hcs_ Utilization
Program Library Management
Tailoring the Command Environment
Dialing Terminals to a Process
Message Segment Facility
Rings and Gate Writing
Data Segments, Temporary Segments
Creating an Error Table

Manuals:
MPM - Subsystem Writers' Guide (AK92)
SDN - Message Segment Facility (AN69)
PLM - Library Maintenance (AN80)

F15D TOPIC MAP

DAY	MORNING TOPICS	AFTERNOON TOPICS
1	SUBSYSTEM WRITING STORAGE SYSTEM SUBROUTINES	STORAGE SYSTEM SUBROUTINES (CONTINUED) WORKSHOP #1
2	WORKSHOP #1 (CONT) MULTICS SECURITY	THE COMMAND ENVIRONMENT WORKSHOP #2
3	ADVANCED MULTICS I/O WRITING I/O MODULES WORKSHOP #3	INTERPROCESS COMMUNICATION WORKSHOP #4
4	INTERPROCESS DATA BASE SHARING INTRAPROCESS TIMER MANAGEMENT WORKSHOP #5	THE STACK AND ARGUMENT LISTS SPECIAL PROGRAMMING TECHNIQUES WORKSHOP #6
5	THE PROCESS ENVIRONMENT WORKSHOP # 7 DIAL FACILITY	DIAL FACILITY DEMONSTRATION MESSAGE SEGMENT FACILITY PROGRAM LIBRARY MANAGEMENT

CONTENTS

		Page
Topic I	Subsystem Writing	1-1
	Introduction	1-1
	Terminology	1-2
	Design Concerns	1-3
	Capabilities for Subsystem Design in Multics	1-5
	Subsystem Design Tools	1-7
	Storage System Subroutines	1-8
Topic II	Storage System Subroutines	2-1
	Obtaining Status Information	2-1
	Multisegment Files	2-8
	Temporary Segments	2-14
Topic III	Storage System Subroutines (cont)	3-1
	Star and Equal Conventions	3-1
	Area Manipulation	3-8
	Introduction	3-8
	Area Format	3-9
	Area Manipulating Subroutines	3-12
	Area Related Commands	3-16
Topic IV	Multics Security	4-1
	Introduction	4-1
	Initial ACL's	4-2
	Rings	4-4
	Introduction	4-4
	Ring Brackets	4-5
	Ring Bracket Subroutines	4-8
	Gates	4-10
	Validation Level	4-15
	Cross Ring I/O	4-17
Topic V	The Command Environment	5-1
	Introduction	5-1
	Modifying the Standard Command Environment	5-2
	Current Ready Procedure	5-6
	Current Command Processor	5-8
	Command Level Intermediary	5-10
	Some Miscellaneous cu_ Entry Points	5-12
	An Example	5-15
Topic VI	Advanced Multics I/O	6-1
	Review	6-1
	Control Orders	6-2
	Useful tty_ Control Orders	6-3

		Page
	Useful vfile_ Control Orders	6-7
	Review of IOCB's	6-9
	Synonyming	6-14
	iox_ Entry Points Used in I/O Modules	6-19
Topic VII	Writing I/O Modules	7-1
	Introduction	7-1
	Implementation Rules	7-2
	Entry Points of an I/O Module	7-4
	Example of an I/O Module	7-7
Topic VIII	Interprocess Communication	8-1
	Overview	8-1
	IPC Terminology	8-4
	IPC Protocol	8-8
	Sending Wakeups	8-10
	IPC Subroutines	8-11
	ipc_ Error Codes	8-13
	Creating and Destroying Event Channels	8-15
	Invoking an Event-Call Procedure	8-17
	Going Blocked on an Event Channel	8-18
	Reading an Event-Wait Channel	8-20
	Control Functions	8-23
	Masking or Assigning Priority to Event Channels	8-27
	An Example Using Event-Wait Channels	8-29
	An Example Using Event-Call Channels	8-35
Topic IX	Interprocess Data Base Sharing	9-1
	Introduction	9-1
	The Locking Mechanism	9-2
	The set_lock_ Subroutine	9-4
	An Example of Locking	9-6
Topic X	Intraprocess Timer Management	10-1
	Introduction	10-1
	Timer Management Terminology	10-2
	timer_manager_ Generic Arguments	10-3
	timer_manager_ Entry Points	10-5
	Blocking a Process	10-7
	Using Call Timers	10-8
	Using Wakeup Timers	10-10
	Resetting and Inhibiting Timers	10-12
	Standard System Handlers	10-15
	Two Examples Using Timers	10-16
	timer_manager_ Summary	10-22
Topic XI	The Stack and Argument Lists	11-1
	The Process Stack Segment	11-1
	The Stack Header	11-2
	The Stack Frame	11-4

		Page
	Argument List Format	11-7
	Argument Descriptors	11-10
Topic XII	Special Programming Techniques	12-1
	Calling a Procedure on the Fly	12-1
	Building Data Segments	12-4
	Creating an Error Table	12-9
Topic XIII	The Process Environment	13-1
	The Standard Process Environment	13-1
	Process Creation in the Supervisor Ring	13-3
	Process Initialization in the User Ring	13-5
	Modifying the Process Environment	13-9
	Project Administration	13-11
	Closed Subsystems	13-13
	Limited Subsystems	13-14
	New Subsystems	13-18
Topic XIV	Dialing Terminals to a Process	14-1
	Overview	14-1
	Implementation of the Dial Facility	14-2
	dial_manager_	14-3
	Dialing Terminals to a Process	14-4
	Subroutines	14-5
	The 'dial' Command	14-7
	An Example	14-8
	Dialing Out to a Terminal	14-13
	dial_manager_ Entry Points	14-14
Topic XV	Message Segment Facility	15-1
	What Is It?	15-1
	Applications	15-2
	The Message Segment	15-3
	Layered Design	15-6
	Primitive Message Segment Facility	15-8
	Extended Access	15-9
	message_segment_ Subroutine Summary	15-10
	Message Segment Facility Illustrative Example	15-12
Topic XVI	Program Library Management	16-1
	Introduction	16-1
	Organization of Program Libraries	16-2
	Naming Conventions	16-4
	A Typical Program Library	16-6
	Program Library Management Tools	16-9
	Installation Tools	16-10
	Library Descriptor Tools	16-20
Appendix A	AIM	A-1
	Concepts	A-1
	Commands and Subroutines	A-3

CONTENTS (con't)

		Page
Appendix B	Program Listings.	B-1
	user_init_admin_.	B-1
	user_real_init_admin_	B-3
	process_overseer_	B-6
	project_start_up_	B-10
	listen_	B-15
Appendix C	Encoding of Channel Names	C-1
Appendix D	Instructor Code for IPC Workshop.	D-1
Appendix E	Standard Process Overseers.	E-1
Appendix F	Gate and Message Segment Examples	F-1
Appendix G	Advanced Dial Facility Example.	G-1
Appendix W	Workshops	W-1
	Workshop One.	W-1
	Workshop Two.	W-3
	Workshop Three.	W-6
	Workshop Four	W-7
	Workshop Five	W-9
	Workshop Six.	W-12
	Workshop Seven.	W-13

STUDENT BACKGROUND

Multics Subsystem Programming (F15D)

NAME: _____ PHONE: _____

TITLE: _____

COMPANY ADDRESS: _____

_____

_____

MANAGER: _____ OFFICE PHONE: _____

INSTRUCTOR'S NAME: _____

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)

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

[ ] PL1	[ ] COBOL	[ ] FORTRAN	[ ] ASSEMBLY
[ ] JCL	[ ] OPERATIONS	[ ] GCOS	[ ] MULTICS
[ ] NPS	[ ] GRTS	[ ] CP6	[ ] OTHER

4. Detail any experience you have had which is related to the material in this course.

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:

HONEYWELL MARKETING EDUCATION

COURSE AND INSTRUCTOR EVALUATION FORM

INSTRUCTOR _____

COURSE _____

START DATE _____

LOCATION _____

STUDENT NAME _____ (OPTIONAL)

In the interest of developing training courses of high quality, and then improving on that base, we would like you to complete this questionnaire. Your information will aid us in making future revisions and improvements to this course. Both the instructor and his/her manager will review these responses.

Please complete the form and return it to the instructor upon the completion of the course. In questions 1 through 14, check the appropriate box and feel free to include additional comments. Attach additional sheets if you need more room for comments. Be objective and 'concrete' in your comments -- be critical when criticism is appropriate.

1. Considering the stated objectives of this course, rate the overall length of the course.

CAN'T JUDGE	TOO SHORT				ABOUT RIGHT				TOO LONG
0	1	2	3	4	5	6	7	8	9

COMMENTS_____

_____

_____

2. Considering the objectives, rate the technical level at which the course was taught.

CAN'T JUDGE	NOT TECH ENOUGH				ABOUT RIGHT				TOO TECH
0	1	2	3	4	5	6	7	8	9

COMMENTS_____

_____

_____

3. Considering the objectives, rate the emphasis placed on the more important topics.

CAN'T JUDGE	POOR				GOOD				EXCELLENT
0	1	2	3	4	5	6	7	8	9

COMMENTS_____

_____

_____

4. Rate the sequence in which the topics were presented.

CAN'T JUDGE	POOR				GOOD				EXCELLENT
0	1	2	3	4	5	6	7	8	9

COMMENTS_____

_____

_____

5. Rate the format and quality of the learning materials (slides, student handbooks, supplementary handouts, etc.).

CAN'T JUDGE	POOR				GOOD				EXCELLENT
0	1	2	3	4	5	6	7	8	9

COMMENTS_____

6. Rate the amount of time given for the completion of the workshops.

CAN'T JUDGE	TOO LITTLE TIME				ABOUT RIGHT				TOO MUCH TIME
0	1	2	3	4	5	6	7	8	9

COMMENTS_____

7. Rate the workshops' ability to relate back to and reinforce the material presented.

CAN'T JUDGE	POOR				GOOD				EXCELLENT
0	1	2	3	4	5	6	7	8	9

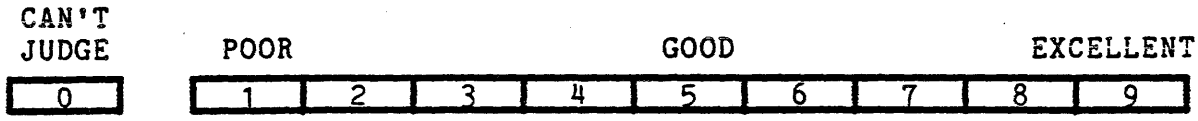
COMMENTS_____

8. Rate the physical condition of the classroom (space available, temperature, lighting, etc.).

CAN'T JUDGE	POOR				GOOD				EXCELLENT
0	1	2	3	4	5	6	7	8	9

COMMENTS_____

9. Rate the physical condition of the lab or workshop room. (systems configuration, space available, learning tools, terminals, tables, etc.).

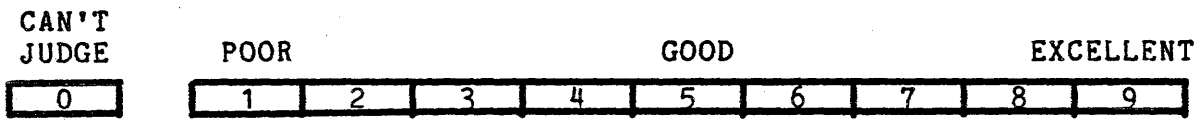


COMMENTS_____

_____

_____

10. Rate your instructor's demonstrated knowledge of the course material.

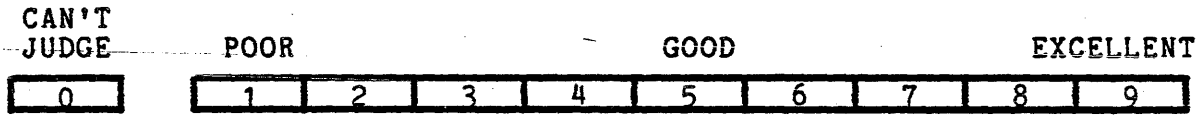


COMMENTS_____

_____

_____

11. Rate your instructor's ability to convey the technical aspects of the various topics.

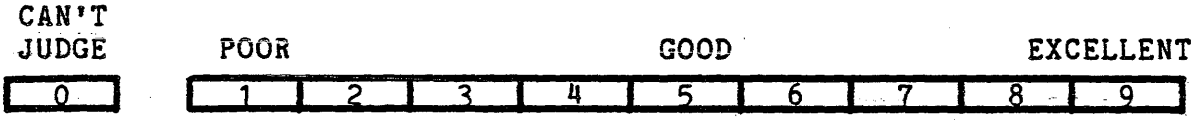


COMMENTS_____

_____

_____

12. Rate the classroom and workshop assistance given you by your instructor.



COMMENTS_____

_____

_____

13. Rate the instructor's ability to create an environment in which you felt free to ask questions.

CAN'T JUDGE	POOR				GOOD				EXCELLENT
0	1	2	3	4	5	6	7	8	9

COMMENTS _____

14. Rate the relevance of the skills learned in the course with respect to your job or further training.

CAN'T JUDGE	POOR				GOOD				EXCELLENT
0	1	2	3	4	5	6	7	8	9

COMMENTS _____

15. What did you like <u>most</u> about this course?

16. What did you like <u>least</u> about this course?

17. Other comments please:

_____

_____

_____

_____

_____

_____

_____

18. Of the following job categories, check the ones which most nearly represent the bulk of your experience, and to the right of your responses indicate the number of years you have acted in that capacity.

[] Applications Programmer. . . . _____ years

[] Field Engineering Analyst. . . _____ years

[] Manager. . . . . _____ years

[] Marketing Analyst. . . . . _____ years

[] Salesperson. . . . . _____ years

[] Secretary. . . . . _____ years

[] Systems Analyst. . . . . _____ years

[] Systems Programmer . . . . . _____ years

[] Other. . . . . _____ years

Please give "other" title _____

TOPIC I

Subsystem Writing

	Page
Introduction	1-1
Terminology	1-2
Design Concerns	1-3
Capabilities for Subsystem Design in Multics	1-5
Subsystem Design Tools	1-7
Storage System Subroutines	1-8

INTRODUCTION

- A BASIC GOAL OF THE Multics SYSTEM DESIGN PHILOSOPHY:

  TO PROVIDE A SYSTEM WHICH IS OPEN-ENDED AND CAPABLE OF SUPPORTING USER DESIGNED SUBSYSTEMS

  - TO ACHIEVE THIS, Multics
    - HAS BEEN HIGHLY MODULARIZED
      - FUNCTIONALITY LOCALIZED
      - COMPLEXITY OF ANY GIVEN MODULE MINIMIZED
    - IS MOSTLY WRITTEN IN PL/I
      - MORE EASILY READ THAN ALM
      - ENABLES ADOPTION OF SUBSYSTEMS FROM OTHER MACHINES USING PL/I
    - FEATURES A WEALTH OF TOOLS TO HELP DESIGN, IMPLEMENT, AND MAINTAIN SUBSYSTEMS

- THIS COURSE IS DESIGNED TO:

  - INTRODUCE MOST TOPICS COVERED IN THE SUBSYSTEM WRITERS' GUIDE (SWG)
  - COVER IN DETAIL SEVERAL ADVANCED TOOLS AND TECHNIQUES OFTEN USED IN WRITING SUBSYSTEMS
  - PROVIDE INSIGHT INTO HOW TO WRITE SUBSYSTEMS "THE Multics WAY"

TERMINOLOGY

- A SUBSYSTEM CAN BE DEFINED A VARIETY OF WAYS:
  - A "SYSTEM" WHICH OPERATES WITHIN THE CONFINES OF ANOTHER, LARGER SYSTEM
  - PROGRAM(S) THAT PROVIDE A SPECIAL ENVIRONMENT FOR SOME PARTICULAR PURPOSE
  - PROGRAM(S) THAT PROVIDE A NUMBER OF OPERATIONS ON SOME RESTRICTED UNIVERSE OF DATA

- EXAMPLES OF STANDARD Multics SUBSYSTEMS: qedx, ted, emacs, calc, probe, read_mail, send_mail, help, ~~abbrev~~

- A SUBSYSTEM IS SAID TO BE CLOSED IF:
  - ALL NECESSARY OPERATIONS CAN BE HANDLED WITHIN THE SUBSYSTEM
  - NO WAY EXISTS TO USE THE NORMAL MULTICS ENVIRONMENT FROM WITHIN THE SUBSYSTEM
  - EXAMPLE: THE 'fast' SUBSYSTEM

DESIGN CONCERNS

- HAS THE PROBLEM ALREADY BEEN SOLVED?

- SECURITY
  - IS SUBVERSION A REAL CONCERN?
  - WILL ACL ALONE SUFFICE, OR MUST WE RESORT TO RINGS AND AIM?
  - CLOSED SUBSYSTEM?

- HOW SHALL WE INTERFACE WITH THE STORAGE SYSTEM?
  - TEMPORARY SEGS
  - AREAS
  - PERMANENT SEGS, MSFs
  - NAME AND ADDRESS SPACE MANAGEMENT
  - USE MULTICS I/O SYSTEM?

- WILL PROCESSES NEED TO COMMUNICATE WITH EACH OTHER?

DESIGN CONCERNS

- DOES THE SUBSYSTEM HAVE A "MULTICS FLAVOR"?

- DOCUMENTATION

- SUBSYSTEM LIBRARY MAINTENANCE

CAPABILITIES FOR SUBSYSTEM DESIGN IN MULTICS

- THE OPPORTUNITIES FOR SUBSYSTEM DESIGN IN Multics ARE VIRTUALLY UNLIMITED, AND THE SUBSYSTEM DESIGNER MAY:

  - MODIFY THE COMMAND INTERFACE TO THE Multics STORAGE SYSTEM
  - MANIPULATE THE ADDRESS SPACE OF A USER PROCESS
  - MODIFY THE COMMAND ENVIRONMENT OF A USER PROCESS
  - WRITE COMMAND AND/OR ACTIVE FUNCTION PROCEDURES
  - WRITE A COMMAND PROCESSOR PROCEDURE
  - HANDLE CONTROL COMMUNICATION BETWEEN ANY NUMBER OF ASYNCHRONOUS, COOPERATING PROCESSES
  - CONTROL CONCURRENT ACCESS TO CRITICAL, SHARED DATA BASES
  - USE TIMERS
  - INTERFACE NEW I/O DEVICES, MONITOR EXISTING I/O DEVICES, ETC.
  - MODIFY, RESTRICT, OR REPLACE ENTIRELY THE PROCESS ENVIRONMENT

CAPABILITIES FOR SUBSYSTEM DESIGN IN MULTICS

- DIAL TERMINALS TO A PROCESS, OR ALLOW A PROCESS TO DIAL OUT TO A TERMINAL
- WRITE GATES
- MANIPULATE MESSAGE SEGMENTS
- CREATE, UPDATE, AND IN GENERAL, MAINTAIN PROGRAM LIBRARIES
- AND MANY, MANY OTHER THINGS

SUBSYSTEM DESIGN TOOLS

- TO ACHIEVE SOME OF THE SUBSYSTEM DESIGN TASKS MENTIONED ABOVE, THE DESIGNER HAS AVAILABLE A WIDE VARIETY OF RESOURCES INCLUDING:

  - COMMANDS AND SUBROUTINES

  - SOURCE PROGRAMS

    - WHOSE PERUSAL SHOWS THE DESIGNER HOW Multics DOES IT

    - WHICH MAY BE COPIED AND MODIFIED TO YIELD CUSTOMIZED BEHAVIOR

  - PL/1 AND ALM INCLUDE FILES

  - PROGRAM LIBRARY MAINTENANCE TOOLS

    - EXPEDITE ACCESS TO SYSTEM SOURCE, OBJECT AND INFO SEGMENTS

    - MAINTAIN USER SUBSYSTEM LIBRARIES JUST AS THEY MAINTAIN THE Multics LIBRARIES THEMSELVES

STORAGE SYSTEM SUBROUTINES

- TOPICS 2, 3 AND 4 PRESENT THE SOFTWARE WRITERS' GUIDE (SWG) SUBROUTINES USED IN MANIPULATING THE STORAGE SYSTEM

- THE FOLLOWING LIST PROVIDES A COMPARISON OF THE STORAGE SYSTEM MANIPULATING SUBROUTINES COVERED IN F15C AND F15D

  - EXCEPT WHERE NOTED F15C SUBROUTINES ARE DOCUMENTED IN THE SUBROUTINES MANUAL (AG93) AND F15D SUBROUTINES ARE DOCUMENTED IN THE SWG (AK92)

F15C	F15D
CREATING STORAGE SYSTEM ENTITIES	
hcs_$append_branch hcs_$append_branchx hcs_$append_link hcs_$create_branch_ hcs_$make_seg	
DELETING STORAGE SYSTEM ENTITIES	
delete_ hcs_$delentry_file hcs_$delentry_seg hcs_$del_dir_tree (AK92)	
OBTAINING STATUS INFORMATION	
hcs_$status_ hcs_$status_long hcs_$status_minf hcs_$status_mins	hcs_$get_author hcs_$get_bc_author hcs_$get_link_target hcs_$get_max_length hcs_$get_max_length_seg hcs_$get_safety_sw hcs_$get_safety_sw_seg (hcs_$set_max_length) (hcs_$set_max_length_seg) (hcs_$set_safety_sw) (hcs_$set_safety_sw_seg)

F15C	F15D
WORKING, DEFAULT, AND PROCESS DIRECTORIES	
change_wdir_ get_default_wdir_ (AK92) get_pdir_ get_wdir_	
MANIPULATING THE ADDRESS AND NAME SPACES	
hcs_$fs_get_path_name hcs_$fs_get_ref_name hcs_$fs_get_seg_ptr hcs_$initiate hcs_$initiate_count hcs_$make_seg hcs_$terminate_file hcs_$terminate_name hcs_$terminate_noname hcs_$terminate_seg term_$refname term_$seg_ptr term_$single_refname term_$term_ term_$unsnap	
MULTISEGMENT FILES	
	msf_manager_$acl_add msf_manager_$acl_delete msf_manager_$acl_list msf_manager_$acl_replace msf_manager_$adjust msf_manager_$close msf_manager_$get_ptr msf_manager_$open

F15C	F15D
NAMING AND MOVING DIRECTORY ENTRIES	
hcs_$chname_file hcs_$chname_seg hcs_$fs_move_file hcs_$fs_move_seg	
AFFECTING LENGTH OF ENTRIES	
adjust_bit_count_ hcs_$set_bc hcs_$set_bc_seg hcs_$truncate_file hcs_$truncate_seg	
MANIPULATING PATHNAMES	
absolute_pathname_ absolute_pathname_$add_suffix expand_pathname_ expand_pathname_$add_suffix	
MANIPULATING THE STAR AND EQUAL CONVENTION	
	check_star_name_ get_equal_name_ hcs_$star_ match_star_name_

F15C	F15D
AREAS	
get_system_free_area_(AK92)	area_info_ area_status * create_area * define_area_ release_area set_system_storage * set_user_storage *
SECURITY	
get_group_id_ get_group_id_$tag_star hcs_$add_acl_entries hcs_$add_dir_acl_entries hcs_$delete_acl_entries hcs_$delete_dir_acl_entries hcs_$fs_get_mode hcs_$list_acl hcs_$list_dir_acl hcs_$replace_acl hcs_$replace_dir_acl	cross_ring_ cross_ring_io_$allow_cross cu_$level_get (AG93) cu_$level_set (AG93) get_ring_ hcs_$add_dir_inacl_entries hcs_$add_inacl_entries hcs_$delete_dir_inacl_entries hcs_$delete_inacl_entries hcs_$get_dir_ring_brackets hcs_$get_ring_brackets hcs_$get_user_effmode hcs_$list_dir_inacl_entries hcs_$list_inacl hcs_$replace_dir_inacl hcs_$replace_inacl hcs_$set_dir_ring_brackets hcs_$set_entry_bound hcs_$set_entry_bound_seg hcs_$set_ring_brackets

* COMMANDS (INCLUDED FOR COMPLETENESS)

TOPIC II

Storage System Subroutines

	Page
Obtaining Status Information	2-1
Multisegment Files	2-8
Temporary Segments	2-14

- hcs_$get_author

  - call hcs_$get_author (dir_name, entryname, chase, author, code);

  - RETURNS Personid.Projectid.tag OF THE CREATOR OF A SEGMENT, DIRECTORY, MULTISEGMENT FILE OR LINK

- hcs_$get_bc_author

  - call hcs_$get_bc_author (dir_name, entryname, bc_author, code);

  - RETURNS Personid.Projectid.tag OF THE BIT COUNT AUTHOR OF A SEGMENT OR DIRECTORY

    - BIT COUNT AUTHOR = LAST PERSON WHO SET THE BIT COUNT

- hcs_$get_max_length, hcs_$get_max_length_seg

  - call hcs_$get_max_length (dir_name, entryname, max_length, code);

  - call hcs_$get_max_length_seg (seg_ptr, max_length, code);

  - RETURNS THE MAXIMUM LENGTH (IN WORDS) OF A SEGMENT, DIRECTORY OR LINK TARGET

  - SUBROUTINES THAT CAN CHANGE THE MAXIMUM LENGTH OF A SEGMENT

    - hcs_$set_max_length, hcs_$set_max_length_seg

    - SAME CALL ARGUMENTS AS ABOVE

    - A DIRECTORY CANNOT HAVE ITS MAXIMUM LENGTH CHANGED

    - ONCE MAX LENGTH HAS BEEN SET, AN out_of_bounds FAULT OCCURS WHEN REFERENCING BEYOND END OF SEGMENT

    - MAXIMUM LENGTH IS SET IN UNITS OF 1024 WORDS

    - REQUESTED LENGTH MAY NOT EXCEED sys_info_$max_seg_size

    - CANNOT USE TO SHORTEN SEGMENT

    - DEFAULT MAX LENGTH OF A SEGMENT IS 255K

    - stack_4 HAS INITIAL MAX LENGTH OF 64K

- hcs_$get_safety_sw, hcs_$get_safety_sw_seg

  - call hcs_$get_safety_sw (dir_name, entryname, safety_sw, code);

  - call hcs_$get_safety_sw_seg (seg_ptr, safety_sw, code);

  - RETURNS THE VALUE OF THE SAFETY SWITCH OF A DIRECTORY OR SEGMENT

  - SUBROUTINES THAT CAN CHANGE THE VALUE OF THE SAFETY SWITCH:

    - hcs_$set_safety_sw, hcs_$set_safety_sw_seg

    - SAME CALL ARGUMENTS AS ABOVE

- hcs_$get_link_target

  - call hcs_$get_link_target (dir_name, entryname, link_dir_name,
    link_entryname, code);

  - RETURNS THE TARGET PATHNAME OF A LINK

OBTAINING STATUS INFORMATION

- ON THE FOLLOWING PAGES IS AN EXAMPLE USING SOME OF THE SUBROUTINES PROVIDING STATUS INFORMATION
  - IT ALSO SERVES AS A REVIEW OF SOME ITEMS INTRODUCED IN F15C
    - WRITING A COMMAND
    - USING ioa_ AND com_err_
  - IN YOUR FIRST WORKSHOP YOU WILL BE ASKED TO ENHANCE THIS PROGRAM

```
STATUS: proc;
dcl   cu_$arg_count entry (fixed bin),
      cu_$arg_ptr entry (fixed bin, ptr, fixed bin, fixed bin (35)),
      hcs_$status_minf entry (char (*), char (*), fixed bin (1),
      fixed bin (2), fixed bin (24), fixed bin (35)),
      hcs_$get_safety_sw entry (char(*),char(*),bit (1),fixed bin (35)),
      hcs_$get_max_length entry (char (*), char (*), fixed bin (19),
                                 fixed bin (35)),
      hcs_$get_author entry (char (*), char (*), fixed bin (1), char (*),
                                 fixed bin (35)),
      expand_pathname_ entry (char(*),char(*),char(*),fixed bin (35)),
     (ioa_, com_err_) entry options (variable);
dcl   nargs fixed bin;
dcl   argl fixed bin;
dcl   argp ptr;
dcl   arg char (argl) based (argp);
dcl   dir char (168);
dcl   entry char (32);
dcl   code fixed bin (35);
dcl   type fixed bin (2),
      bc fixed bin (24),
      author char (32),
      max_length fixed bin (19),
      safety_sw bit (1),
      ME char (6) static init ("STATUS") options (constant);
dcl   error_table_$wrong_no_of_args ext fixed bin (35);

/* VERIFY NUMBER OF ARGUMENTS */
          call cu_$arg_count (nargs);
          if nargs ^= 1 then do;
               call com_err_ (error_table_$wrong_no_of_args, ME);
               return;
          end;

/* PROCESS SEGMENT NAME ARGUMENT */
          call cu_$arg_ptr (1, argp, argl, code);
          call expand_pathname_ (arg, dir, entry, code);
          if code ^= 0 then call ERROR;

/* FIND OUT WHAT TYPE OF BRANCH IT IS */
          call hcs_$status_minf (dir, entry, 0, type, bc, code);
          if code ^= 0 then call ERROR;
```

```
/* TELL THE USER */
          if type = 2 & bc ^= 0 then
             call ioa_ ("^a is a ^i component multisegment file",
             entry, bc);
          else call ioa_ (
             "^a is a ^[link^;segment^;directory^]
              ^[with bit count ^i^;^s^]",
              entry, type+1, (type = 1), bc);

/* GET OTHER INFORMATION AND REPORT IT TO THE USER */
          call hcs_$get_author (dir, entry, 0, author, code);
          if code ^= 0 then call ERROR;
          call hcs_$get_max_length (dir, entry, max_length, code);
          if code ^= 0 then call ERROR;
          call hcs_$get_safety_sw (dir, entry, safety_sw, code);
          if code ^= 0 then call ERROR;
          call ioa_ ("      It was created by ^a,
              it has a max length of ^i,
              and the safety switch is ^[on^;off^].",
              author, max_length, safety_sw);
ERROR:    proc;
               call com_err_ (code, ME);
               goto FINISH;
          end;

FINISH: end STATUS;
```

```
!STATUS STATUS.pl1
STATUS.pl1 is a segment
              with bit count 23256
      It was created by NDibble.MEDmult.a,
              it has a max length of 261120,
              and the safety switch is off.

!STATUS <
NDibble is a directory

      It was created by Initializer.SysDaemon.z,
              it has a max length of 65536,
              and the safety switch is on.


!STATUS test_file
test_file is a 2 component multisegment file
      It was created by NDibble.MEDmult.a,
              it has a max length of 65536,
              and the safety switch is off.

!lk >udd>F15D>s1 blurp

!STATUS blurp
blurp is a link

      It was created by NDibble.MED.a,
              it has a max length of 65536,
              and the safety switch is off.


!sml STATUS.pl1 40960


!STATUS STATUS.pl1
STATUS.pl1 is a segment
              with bit count 23292
      It was created by NDibble.MEDmult.a,
              it has a max length of 40960,
              and the safety switch is off.


!STATUS **
STATUS: Entry not found.
```

MULTISEGMENT FILES

- MULTISEGMENT FILES ARE:
  - FILES THAT USE MORE THAN ONE SEGMENT FOR STORAGE
  - COMPOSED OF ONE OR MORE COMPONENTS, EACH IS A SEGMENT, AND IS IDENTIFIED BY AN UNSIGNED INTEGER
  - VIEWED BY MANY MULTICS SUBROUTINES AS DIRECTORIES
  - USED FOR LARGE LISTINGS, INDEXED FILES, ETC.
  - MANAGED BY THE msf_manager_ SUBROUTINE
- MANIPULATING A MULTISEGMENT FILE REQUIRES USE OF A MULTISEGMENT FILE CONTROL BLOCK
  - THE CONTROL BLOCK FOR A MULTISEGMENT FILE IS CREATED AND MAINTAINED BY THE msf_manager_ IN THE USER'S PROCESS DIRECTORY
- LOCATIONS IN A MULTISEGMENT FILE ARE SPECIFIED BY A PATHNAME, COMPONENT NUMBER AND WORD OFFSET WITHIN THE COMPONENT

- msf_manager_$open

  - call msf_manager_$open (dir_name, entryname, fcb_ptr, code);

  - CREATES A FILE CONTROL BLOCK IN SYSTEM FREE STORAGE AND RETURNS A FILE CONTROL BLOCK POINTER

  - THE MSF NEEDN'T EXIST (A FCB IS STILL ALLOCATED)

  - THE fcb_ptr IS USED BY ALL FUTURE CALLS TO msf_manager_

- msf_manager_$get_ptr

  - call msf_manager_$get_ptr (fcb_ptr, component, create_sw, seg_ptr, bc, code);

  - RETURNS A POINTER TO A SPECIFIED COMPONENT IN THE MSF

  - COMPONENT IS AUTOMATICALLY CREATED, IF create_sw = "1"b

  - IF THE FILE IS A SINGLE SEGMENT FILE AND A COMPONENT GREATER THAN 0 IS REQUESTED, THE SEGMENT IS CONVERTED INTO A MSF

```
!pr MSF.pl1

 MSF: proc;
      dcl msf_manager_$open entry (char(*), char(*),
                                   ptr, fixed bin(35));
      dcl msf_manager_$get_ptr entry (ptr, fixed bin, bit(1),
                           ptr, fixed bin(24), fixed bin(35));
      dcl hcs_$initiate entry (char(*), char(*), char(*),
              fixed bin(1), fixed bin(2), ptr, fixed bin(35));
      dcl code fixed bin (35);
      dcl (fcb_ptr, seg_ptr) ptr;
      dcl bc fixed bin (24);
      dcl sysprint file;
      dcl ioa_ entry() options(variable);

      call hcs_$initiate (">udd>MED>nd>F15D", "test_file", "",
                          0, 0, seg_ptr, code);

      call ioa_ ("^p^/", seg_ptr);
      /* PROBE BREAKPOINT SET HERE */

      call msf_manager_$open (">udd>MED>nd>F15D", "test_file",
                              fcb_ptr,code);

      call msf_manager_$get_ptr (fcb_ptr, 0, "0"b, seg_ptr, bc,code);

      call ioa_ ("^/Component 0 starts at ^p", seg_ptr);

      call msf_manager_$get_ptr (fcb_ptr, 1, "1"b, seg_ptr, bc,code);

      call ioa_ ("^/Component 1 starts at ^p", seg_ptr);

 end MSF;

!create test_file

!ls test_file

  Segments = 1, Lengths = 0.

  r w    0  test_file

!MSF

  503|0
  Stopped after line 17 of MSF. (level 7)

!..lrn 503

     503 >udd>MED>nd>F15D>test_file
```

```
!continue
  Component 0 starts at 503¦0
  Component 1 starts at 501¦0

!lrn 503 501
     503 >udd>MED>nd>F15D>test_file>0
     501 >udd>MED>nd>F15D>test_file>1

!ls test_file
  Multisegment-files = 1, Lengths = 1.
  r w    1  test_file
```

- msf_manager_$adjust

  - call msf_manager_$adjust (fcb_ptr, component, bc, switch, code);

  - OPTIONALLY SETS THE BIT COUNT, TRUNCATES AND TERMINATES A COMPONENT

  - SWITCH HAS 3 BITS

    - IF BIT 1 IS ON THE BIT COUNT IS SET (BIT COUNT OF ALL COMPONENTS < component SET TO sys_info$max_seg_size)

    - IF BIT 2 IS ON THE COMPONENT IS TRUNCATED

    - IF BIT 3 IS ON THE COMPONENT IS TERMINATED

  - ALL COMPONENTS WITH NUMBERS GREATER THAN THE GIVEN COMPONENT ARE DELETED

- msf_manager_$close

  - call msf_manager_$close (fcb_ptr);

  - TERMINATES ALL COMPONENTS OF THE MSF, FREES THE FILE CONTROL BLOCK, AND SETS fcb_ptr NULL

msf_manager_ ACL ENTRY POINTS ARE SIMILAR TO hcs_ ACL ENTRY POINTS

hcs_	msf_manager_
list_acl	acl_list
replace_acl	acl_replace
add_acl_entries	acl_add
delete_acl_entries	acl_delete

TEMPORARY SEGMENTS

- TEMPORARY SEGMENTS
  - RESIDE IN THE PROCESS DIRECTORY
  - ARE MANAGED AS A POOL
  - HAVE A NAME OF THE FORM:

    <unique_name>.temp.<seg_number>
  - ARE HEAVILY USED BY MANY COMMANDS, SUCH AS qedx

- TEMPSEG POOLING ENABLES THE USE OF THE SAME TEMPSEG MORE THAN ONCE DURING THE LIFE OF A PROCESS, RESULTING IN A REDUCED COST TO THE PROCESS

- THE list_temp_segments COMMAND GIVES DETAILED INFORMATION ABOUT THE STATE OF A PROCESS' TEMPORARY SEGMENT POOL

TEMPORARY SEGMENTS

- FOUR SUBROUTINES MANIPULATE TEMPORARY SEGMENTS:

  - get_temp_segments_

    - call get_temp_segments_ (program_name, ptrs, code);

    - RETURNS POINTERS TO TEMPORARY SEGMENTS FOR A SPECIFIED PROGRAM

    - CALLER SUPPLIES

      - NAME OF REQUESTING PROGRAM

      - AN ARRAY OF POINTERS WHOSE EXTENT EQUALS THE NUMBER OF TEMPSEGS DESIRED

    - SEE ALSO get_temp_segment_

- release_temp_segments_

  - call release_temp_segments_ (program_name, ptrs, code);

  - USED TO RETURN TEMPORARY SEGMENTS TO THE FREE POOL (SO THAT THEY MAY BE REUSED, IF DESIRED)

  - CALLER SUPPLIES

    - NAME OF PROGRAM "OWNING" THE TEMPSEGS

    - ARRAY OF POINTERS TO THE TEMPSEGS TO BE RETURNED TO POOL

  - THE TEMPORARY SEGMENTS BEING 'RETURNED' ARE NOT DELETED

  - IF RELEASE IS SUCCESSFUL, POINTERS ARE NULLED

  - ANY ATTEMPT TO RELEASE TEMPSEGS NOT "OWNED" BY REQUESTOR RESULTS IN error_table_$argerr; PASSED POINTERS ARE UNCHANGED.

  - SEE ALSO release_temp_segment_

```
! pr DEMO_TEMP_SEGS.pl1

  DEMO_TEMP_SEGS: proc;
  dcl get_temp_segments_ entry (char(*), (*) ptr, fixed bin(35));
  dcl release_temp_segments_ entry (char(*), (*) ptr, fixed bin(35));
  dcl error_table_$argerr fixed bin(35) ext static;
  dcl ioa_ entry options (variable);
  dcl p_array(3) ptr;
  dcl code fixed bin(35);
  call get_temp_segments_("requestor_1", p_array, code);
  call ioa_ ("Check the following tempseg segnos:^/^(^2x^p^)", p_array);
  call release_temp_segments_ ("requestor_2", p_array, code);
  if code = error_table_$argerr then call ioa_
          ("requestor_2 may not free segments owned by requestor_1.");
  call ioa_
   ("Pointers after a bad call to release_temp_segments are:^/^(^2x^p^)",
         p_array);
  call release_temp_segments_ ("requestor_1", p_array, code);
  end DEMO_TEMP_SEGS;


! list_temp_segments
         12 Segments,  11 Free

  !BBBJHmQJDkmGxW.temp.0315  command_processor_


! DEMO_TEMP_SEGS
  Check the following tempseg segnos:
    344|0  354|0  355|0
  requestor_2 may not free segments owned by requestor_1.
  Pointers after a bad call to release_temp_segments are:
    344|0  354|0  355|0


! lrn 344 354 355
     344 >process_dir_dir>!BcDBdwpbBBBBBB>!BBBJHmQJGFjKqg.temp.0344

     354 >process_dir_dir>!BcDBdwpbBBBBBB>!BBBJHmQJkkPNPb.temp.0354

     355 >process_dir_dir>!BcDBdwpbBBBBBB>!BBBJHmQJkkkWlf.temp.0355


! list_temp_segments
         12 Segments,  11 Free

  !BBBJHmQJDkmGxW.temp.0315  command_processor_
```

TOPIC III

Storage System Subroutines (cont)

	Page
Star and Equal Conventions	3-1
Area Manipulation	3-8
Introduction	3-8
Area Format	3-9
Area Manipulating Subroutines	3-12
Area Related Commands	3-16

STAR AND EQUAL CONVENTIONS

- MOTIVATION

  - THE WRITER OF A SUBSYSTEM OR COMMAND MUST DECIDE IF HE WILL ALLOW THE USER TO SPECIFY ENTRYNAMES THAT USE THE STAR AND EQUAL CONVENTIONS

  - RECALL THAT ENTRYNAMES USING THESE CONVENTIONS CONTAIN THE CHARACTERS "*", "?", "=" OR "%"

  - THE FOLLOWING SUBROUTINES ARE USED TO PROCESS SUCH ENTRYNAMES

    - hcs_$star_

    - get_equal_name_

    - check_star_name_

    - match_star_name_

- hcs_$star_

  - call hcs_$star_ (dir_name, star_name, star_select_sw, area_ptr, star_entry_count, star_entry_ptr, star_names_ptr, code);

  - FOR A GIVEN DIRECTORY, RETURNS AN ARRAY OF ENTRYNAMES THAT MATCH A GIVEN STARNAME

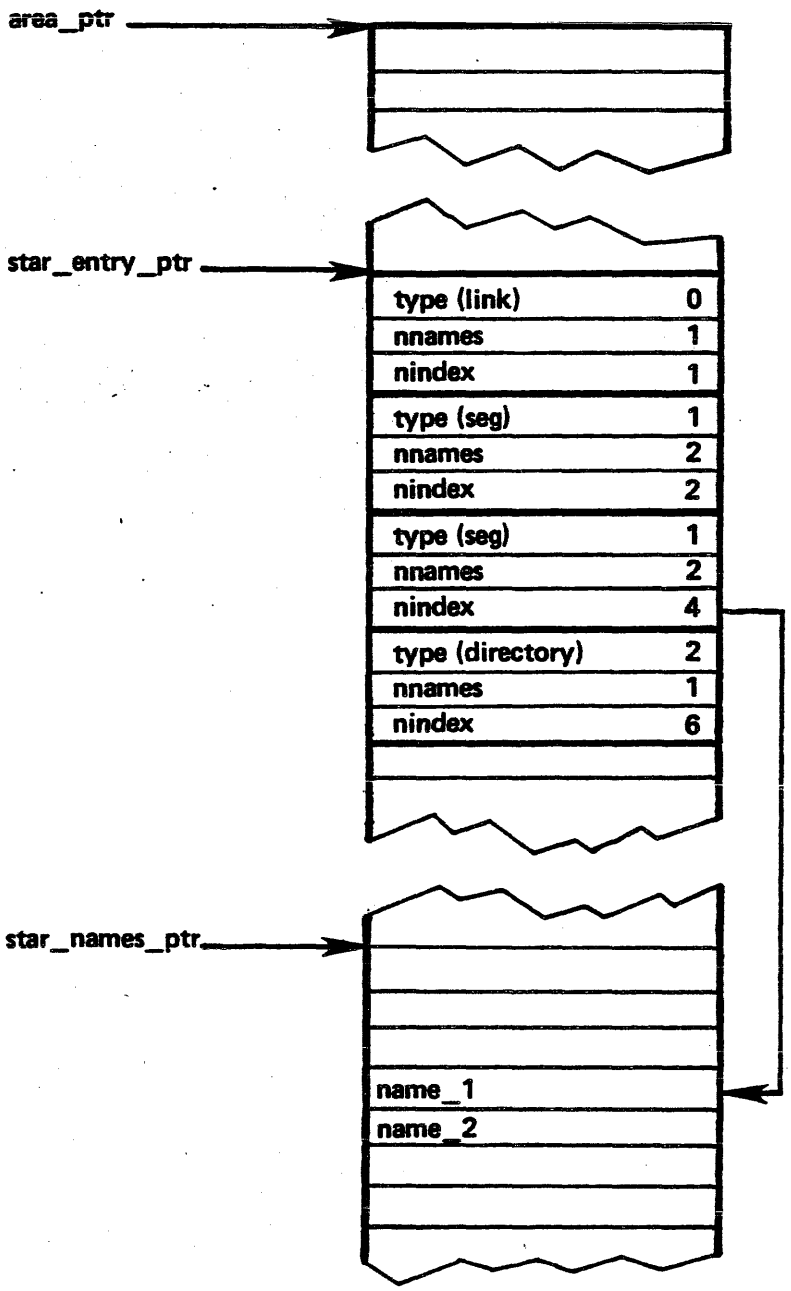
  - star_select_sw DICTATES OPERATION:

    1 - LINK NAMES ONLY

    2 - SEGS AND DIRS ONLY
    (MSF'S COME BACK AS DIRS)

    3 - SEGS, DIRS, AND LINKS

  - USER PROVIDES AN AREA FOR RETURNED ENTRYNAMES AND RELATED INFORMATION


area_ptr
star_entry_ptr
type (link) 0
nnames 1
nindex 1
type (seg) 1
nnames 2
nindex 2
type (seg) 1
nnames 2
nindex 4
type (directory) 2
nnames 1
nindex 6
star_names_ptr
name_1
name_2

```
!print lspl1.pl1

lspl1: proc;
dcl  get_system_free_area_ entry returns (ptr);

dcl  hcs_$star_ entry (char (*), char (*), fixed bin (2),
     ptr, fixed bin, ptr, ptr, fixed bin (35));

dcl 1 star_entries (star_entry_count) aligned based (star_entry_ptr),
      2 type fixed binary (2) unsigned unaligned,
      2 nnames fixed binary (16) unsigned unaligned,
      2 nindex fixed binary (18) unsigned unaligned;

dcl star_names (sum (star_entries (*).nnames))
                char (32) based (star_names_ptr);

dcl star_entry_count fixed binary,
    star_entry_ptr pointer,
    star_names_ptr pointer,
    code fixed bin (35),
    ioa_ entry options (variable);

dcl (i, j) fixed bin;

          call hcs_$star_ (">udd>MED>nd>F15D", "**.pl1",
               2, get_system_free_area_ (), star_entry_count,
               star_entry_ptr, star_names_ptr, code);
          call ioa_ ("^i segments match **.pl1:^/",
                  star_entry_count);
          do i = 1 to star_entry_count;
               if star_entries (i).type = 1 then do;
                    do j = star_entries (i).nindex to
                           star_entries(i).nindex +
                           star_entries(i).nnames - 1;
                         call ioa_ ("^[^2x^]^a",
                              (j ^= star_entries (i).nindex),
                               star_names (j));
                    end;
               end;
          end;
          free star_names_ptr->star_names;
          free star_entry_ptr->star_entries;
     end lspl1;
```

```
!lspl1
13 segments match **.pl1:

decls.incl.pl1
listen_decls.incl.pl1
listen.pl1
put_message.pl1
set_new_command.pl1
command_interceptor_.pl1
process_overseer_.pl1
user_real_init_admin_.pl1
release.pl1
  rl.pl1
get_to_cl_.pl1
cookie.pl1
bound_prog.pl1
lspl1.pl1
  list_pl1.pl1
```

- OTHER hcs_$star_ RELATED ENTRY POINTS RETURN ADDITIONAL INFORMATION ABOUT ENTRIES

  - hcs_$star_dir_list_

  - hcs_$star_list_

  - THESE RETURN INFORMATION SUCH AS WHEN LAST MODIFIED, WHEN LAST USED, MODE, RAW MODE, RECORD LENGTH, BIT COUNT, ETC.

- get_equal_name_

  - call get_equal_name_ (entryname, equal_name, target_name, code);

  - CONSTRUCTS A TARGET NAME FROM AN ENTRYNAME AND AN EQUALNAME

  - EXAMPLE

ENTRYNAME	EQUAL NAME	TARGET NAME
a.b.c	new.=.=	new.b.c
abc.def.ghi	=.%%.5	abc.de.5

- check_star_name_$path

  - call check_star_name_$path (path, code);

  - CHECKS THE ENTRYNAME PORTION OF A PATHNAME TO SEE IF IT HAS BEEN FORMED ACCORDING TO THE RULES FOR CONSTRUCTING STAR NAMES

  - RETURNED CODES:

    0 - ENTRYNAME VALID BUT ISN'T A STAR NAME

    1 - ENTRYNAME VALID AND IS A STAR NAME

    2 - ENTRYNAME IS **, *.**, OR **.*

    error_table_$badstar

  - USED, FOR EXAMPLE, BEFORE CALLING hcs_$star_

- check_star_name_$entry

  - call check_star_name_$entry (entryname, code);

  - SAME AS check_star_name_$path, HOWEVER, ONLY REQUIRES AN ENTRYNAME AS INPUT

- match_star_name_

  - call match_star_name_ (entryname, star_name, code);

  - INDICATES WHETHER OR NOT entryname MATCHES star_name

AREA MANIPULATION

INTRODUCTION

- AREAS ARE
  - STORAGE REGIONS MANAGED BY THE AREA MANAGEMENT FACILITY
  - OFTEN USED TO PASS INFO BACK AND FORTH BETWEEN USER PROCESSES AND THE SUPERVISOR
  - OFTEN (BUT NOT ALWAYS) FOUND IN PROCESS DIRECTORY SEGMENTS, NAMED <unique>.area.linker
- WHY USE AREAS?
  - EMPTYING AN ENTIRE AREA (USING THE 'empty' BUILTIN) IS EASIER THAN USING SEVERAL free STATEMENTS
  - CAN allocate IN PERMANENT SEGS AND HAVE AREA MANAGER DO ALL THE BOOK KEEPING FOR USER
  - GIVES USEFUL OPTIONS LIKE EXTENSIBILITY, ZERO ON FREE, ETC.
  - SOME SUBROUTINES REQUIRE POINTERS TO AREAS AS ARGUMENTS
  - PL/1 OFFSETS ARE USABLE ONLY IN AREAS

AREA FORMAT

- AREAS MAY BE DIVIDED INTO 4 TYPES BASED ON THE TWO FOLLOWING CRITERIA
  - EXTENSIBILITY
    - SOME AREAS ARE LIMITED TO THE SIZE OF A SEGMENT (NON-EXTENSIBLE AREAS)
    - OTHERS CAN "GROW" INTO TEMP SEGMENTS IN THE PROCESS DIRECTORY (EXTENSIBLE AREAS)
  - FREEING OF SPACE WITHIN AN AREA FOR REUSE
    - SOME AREAS HAVE BLOCKS OF FREED SPACE MAINTAINED IN LINKED LISTS AVAILABLE FOR REUSE (FREEING AREAS)
    - OTHERS DO NOT REUSE FREED SPACE IN THE AREA -- ALL ALLOCATIONS ARE DONE IN "VIRGIN AREA" (NO-FREEING AREAS)
  - NO-FREEING AREAS ARE OBVIOUSLY HANDLED MUCH FASTER BY THE AREA MANAGER
- ALL AREAS HAVE 24 WORD HEADERS
  - EXTENSIBLE AREAS HAVE AN ADDITIONAL 12 WORD BLOCK ALLOCATED IN THE AREA (CONTAINS INFORMATION NEEDED BY THE AREA MANAGER TO EXTEND THE AREA)

AREA FORMAT

- FREEING AREAS ARE MADE UP OF:

  1. A HEADER THAT CONTAINS "THREAD HEADS" POINTING TO LINKED LISTS OF FREE BLOCKS (SPACE PREVIOUSLY USED AND THEN FREED)

  2. LINKED LISTS OF FREE BLOCKS (BLOCKS ARE PUT IN LIST BASED ON SIZE)

     FIRST LIST 8 TO 14 WORDS
     SECOND LIST 16 TO 30 WORDS
     THIRD LIST 32 TO 62 WORDS
     ...
     LAST LIST STARTS AT 2**16 WORDS

  3. USED BLOCKS OF WORDS (EVEN WORD BOUNDARIES)

  4. VIRGIN SPACE

- EACH BLOCK STARTS WITH 2 WORDS OF MANAGEMENT INFORMATION SUCH AS SIZE AND A POINTER TO THE AREA HEADER

- THE NEXT WORD OF AN EMPTY BLOCK CONTAINS OFFSETS TO THE PREVIOUS AND NEXT BLOCKS IN THE LINKED LIST

AREA FORMAT

- WHEN SPACE IS FREED THE AREA MANAGER:
  - LOOKS AT THE FIRST 2 WORDS IN THE BLOCK TO DETERMINE SIZE OF THE BLOCK
  - MERGES SMALLER ADJACENT BLOCKS IF POSSIBLE
  - THREADS THE FREED BLOCKS ONTO THE APPROPRIATE LIST

- NOTE

  AREA MANAGER DOES NOT UPDATE BIT COUNT

- get_system_free_area_

  - THIS FUNCTION RETURNS A POINTER TO THE BASE OF THE PROCESS DIRECTORY SEGMENT CONTAINING THE 'system free' AREA FOR THE RING IN WHICH IT IS CALLED

  - USER MAY USE THIS AREA AS HE/SHE PLEASES

```
dcl A area based (get_system_free_area_() );
dcl get_system_free_area_ entry returns (ptr);
dcl alpha based (beta);
dcl beta pointer;

     allocate alpha in (A) set (beta);


     /* WARNING -- DO NOT SET "A" = empty();  */
```

AREA MANIPULATING SUBROUTINES

- define_area_

  - call define_area_ (info_ptr, code);

  - INITIALIZES AN AREA

  - USED TO CONTROL SPECIAL AREA MANAGEMENT FEATURES:

    - EXTEND: ENABLES AREA TO GROW BEYOND MAX SIZE SET INTO TEMPSEGS (INSTEAD OF SIGNALLING THE area CONDITION)

    - ZERO ON ALLOCATION

    - ZERO ON FREEING

    - IGNORE ALL free REQUESTS (FOR DEBUGGING PURPOSES)

    - SET MAX SIZE TO SPECIFIED VALUE (0 modulo 8)

  - USES AN INFORMATION STRUCTURE FOUND IN area_info.incl.pl1

  - REGION BEING INITIALIZED

    - IS POINTED TO BY area_info.areap

    - IS AUTOMATICALLY ACQUIRED FROM PROCESS DIRECTORY TEMPSEG POOL IF area_info.areap = null()

AREA MANIPULATING SUBROUTINES

- release_area_

  - call release_area_ (area_ptr);

  - CLEANS UP AN AREA AFTER IT IS NO LONGER NEEDED

    - RETURNS ANY TEMPSEGS TO THE POOL

- area_info_

  - call area_info_ (info_ptr, code);

  - FILLS IN THE USER-ALLOCATED area_info STRUCTURE (CALLER MUST SET area_info.areap)

```
dcl  area_infop ptr;

dcl 1 area_info aligned based (area_infop),
    2 version fixed bin,
    2 control aligned like area_control,
    2 owner char (32) unal,
    2 n_components fixed bin,
    2 size fixed bin (18),
    2 version_of_area fixed bin,
    2 areap ptr,
    2 allocated_blocks fixed bin,
    2 free_blocks fixed bin,
    2 allocated_words fixed bin (30),
    2 free_words fixed bin (30);

dcl 1 area_control aligned based,
    2 extend bit (1) unal,
    2 zero_on_alloc bit (1) unal,
    2 zero_on_free bit (1) unal,
    2 dont_free bit (1) unal,
    2 no_freeing bit (1) unal,
    2 system bit (1) unal,
    2 pad bit (30) unal;
```

AREA FORMAT

AREA RELATED COMMANDS

- AREA-RELATED COMMANDS (DOCUMENTED IN SWG)

  - create_area

    - PERFORMS define_area_'S TASKS, GIVEN A VIRTUAL POINTER TO AN AREA TO BE CREATED

  - set_system_storage, set_user_storage

    - ENABLE A USER-CREATED AREA TO BE USED INSTEAD OF DEFAULT 'system free' OR 'user free' AREAS

    - AREA SPECIFIED

      - MUST BE ZERO_ON_FREE OR ZERO_ON_ALLOC

      - SHOULD BE EXTENSIBLE

    - USEFUL FOR ISOLATING BUGS WHICH ARE INADVERTENTLY DESTROYING INFORMATION IN EITHER 'system free area' OR 'user free area'

  - area_status

    - COMMAND INTERFACE TO area_info_

YOU ARE NOW READY FOR WORKSHOP #1

TOPIC IV

Multics Security

	Page
Introduction	4-1
Initial ACL's	4-2
Rings	4-4
Introduction	4-4
Ring Brackets	4-5
Ring Bracket Subroutines	4-8
Gates	4-10
Validation Level	4-15
Cross Ring I/O	4-17

INTRODUCTION

- Multics HAS THREE ACCESS CONTROL MECHANISMS
  - THE ACCESS CONTROL LIST MECHANISM (ACL's)
  - THE ACCESS ISOLATION MECHANISM (AIM)
    SEE APPENDIX A
  - THE RING MECHANISM

INITIAL ACL'S

- THERE MAY BE, ASSOCIATED WITH EVERY DIRECTORY, TWO TYPES OF INITIAL ACL CONTROL LISTS (ONE FOR INFERIOR SEGMENTS, ONE FOR INFERIOR DIRECTORIES)

- FOR EVERY SUBROUTINE THAT MANIPULATES ACL'S THERE IS A CORRESPONDING SUBROUTINE THAT MANIPULATES INACL'S

ACL ENTRY POINTS	INACL ENTRY POINTS
hcs_$add_acl_entries	hcs_$add_inacl_entries
hcs_$add_dir_acl_entries	hcs_$add_dir_inacl_entries
hcs_$delete_acl_entries	hcs_$delete_inacl_entries
hcs_$delete_dir_acl_entries	hcs_$delete_dir_inacl_entries
hcs_$list_acl	hcs_$list_inacl
hcs_$list_dir_acl	hcs_$list_dir_inacl
hcs_$replace_acl	hcs_$replace_inacl
hcs_$replace_dir_acl	hcs_$replace_dir_inacl

INITIAL ACL'S

- ALL INACL ENTRY POINTS REQUIRE SPECIFICATION OF A RING NUMBER

- SEGMENT INACL APPLIES TO MSF'S

- SEE THE COMMANDS

sis	sid
dis	did
lis	lid

  ALL OF WHICH ACCEPT A -ring CONTROL ARGUMENT

RINGS

INTRODUCTION

- INTRAPROCESS ACCESS IS CONTROLLED BY THE RING MECHANISM

- TYPICAL APPLICATIONS
  - PROTECTION OF SUPERVISOR FROM USER PROGRAMS
  - PROTECTION OF SUBSYSTEM DATA BASE FROM DIRECT ACCESS

- CLARIFICATION OF MISCONCEPTION
  - ALL SEGMENTS ARE NOT "IN" JUST ONE RING
  - SEGMENTS MAY "SPAN" SEVERAL RINGS

RING BRACKETS

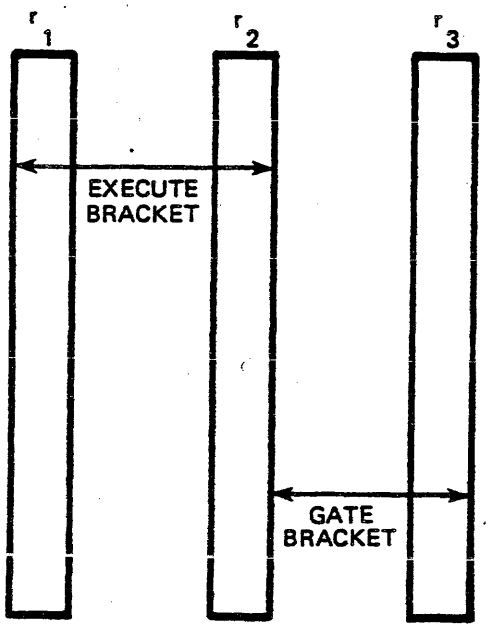
- EACH SEGMENT HAS ASSOCIATED WITH IT 3 RING BRACKET NUMBERS (THESE NUMBERS DETERMINE THE RING BRACKETS FOR THAT SEGMENT)

- RING BRACKETS DEFINE IN WHICH RING A USER CAN READ, WRITE, CALL OR EXECUTE A SEGMENT

  - THE FIRST TWO NUMBERS DELIMIT THE EXECUTE BRACKET

  - THE SECOND AND THIRD NUMBERS DELIMIT THE GATE BRACKET

EXECUTE AND GATE BRACKETS



RING BRACKETS

- RING BRACKET NUMBERS ARE EXPRESSED r1,r2,r3

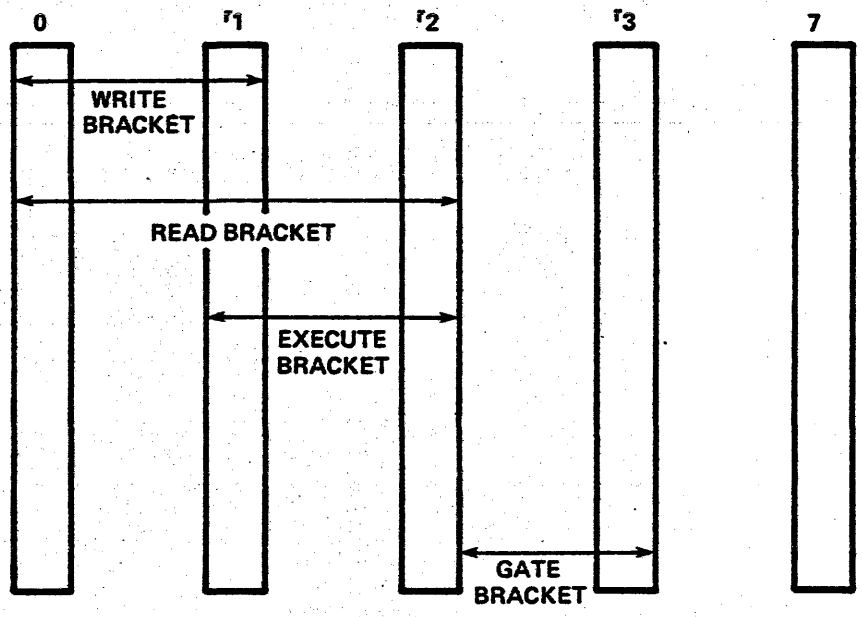
  ASSUMING A RING 4 USER, SPECIFY THE SEGMENTS FOR WHICH THE USER IS IN THE EXECUTE AND/OR GATE BRACKET

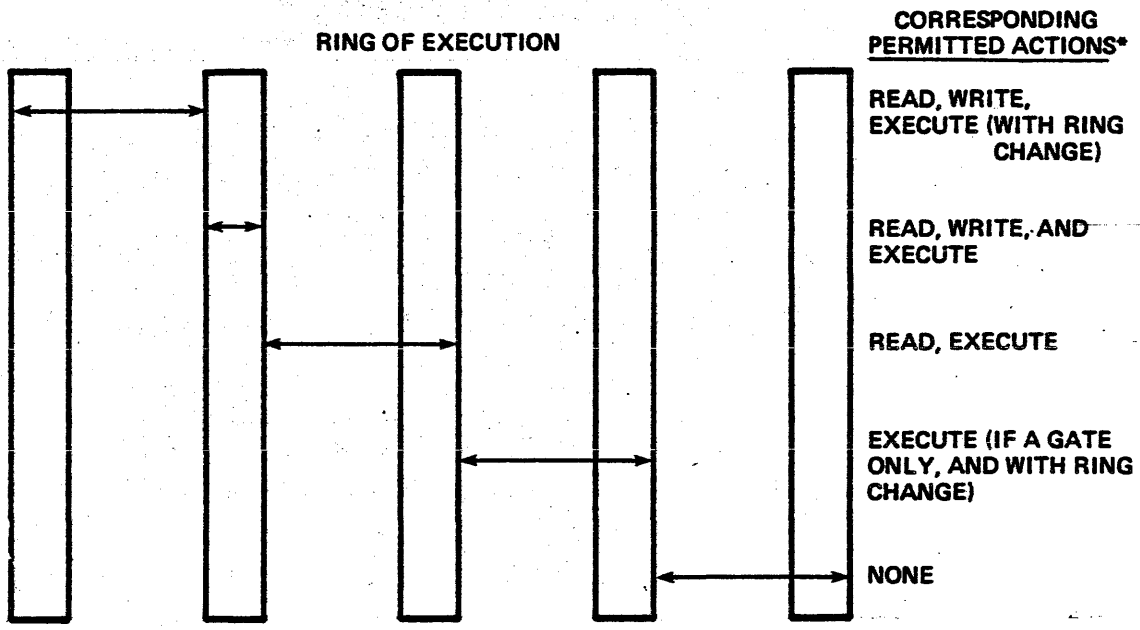
r1	r2	r3	EXECUTE	GATE
4	4	4		
0	5	5		
0	0	5		
5	5	5		
1	1	1		
3	3	4		

- TYPICAL RING BRACKETS
  - USER SEGMENTS 4,4,4
  - SYSTEM COMMANDS AND SUBROUTINES 1,5,5 OR 0,5,5
  - SYSTEM GATES 1,1,5 OR 0,0,5 (hcs_)

- NOTICE THAT TO EXECUTE hcs_ THE USER MUST BE IN RING 0
  - HOWEVER, USERS ARE USUALLY THOUGHT OF AS BEING IN RING 4
  - USERS ACTUALLY TRAVEL IN AND OUT OF THE RING STRUCTURE

RING MECHANISM SUMMARY





* SUBJECT, OF COURSE, TO ACL AND AIM

RING BRACKET SUBROUTINES

- get_ring_

  current_ring = get_ring_();

  RETURNS THE USER'S CURRENT RING OF EXECUTION

- hcs_$get_ring_brackets

  call hcs_$get_ring_brackets (dir_name, entryname, rb, code);

  RETURNS (r1,r2,r3) FOR A SPECIFIED SEGMENT

- hcs_$get_dir_ring_brackets

  call hcs_$get_dir_ring_brackets (dir_name, entryname, drb, code);

  RETURNS (r1,r2) FOR A SPECIFIED DIRECTORY

RING BRACKET SUBROUTINES

- hcs_$set_ring_brackets and hcs_$set_dir_ring_brackets

  - SAME CALL ARGUMENTS AS CORRESPONDING ENTRY POINT ON PREVIOUS PAGE

  - SETS THE RING BRACKETS OF A SPECIFIED SEGMENT OR DIRECTORY

  - THE RING BRACKETS MUST BE >= THE CURRENT VALIDATION LEVEL OF THE CALLING PROCESS

  - SEE ALSO THE set_ring_brackets (srb), set_dir_ring_brackets (sdrb), lset_ring_brackets (lsrb) AND lset_dir_ring_brackets (lsdrb) COMMANDS

- DEFINITION OF A GATE: ONLY POINT AT WHICH A PROCEDURE IN AN OUTER RING CAN TRANSFER TO A PROCEDURE IN AN INNER RING
  - IDENTIFIED BY PRESENCE OF GATE BRACKET (r2 < r3)
  - CHANGES USER'S RING OF EXECUTION
- GATES ARE "CREATED" BY:
  - USING alm MACROS
  - AFTER COMPILATION:
    - THE RING BRACKETS ARE SET TO THAT OF A GATE
    - THE ENTRY BOUND IS SET (DISCUSSED BELOW)

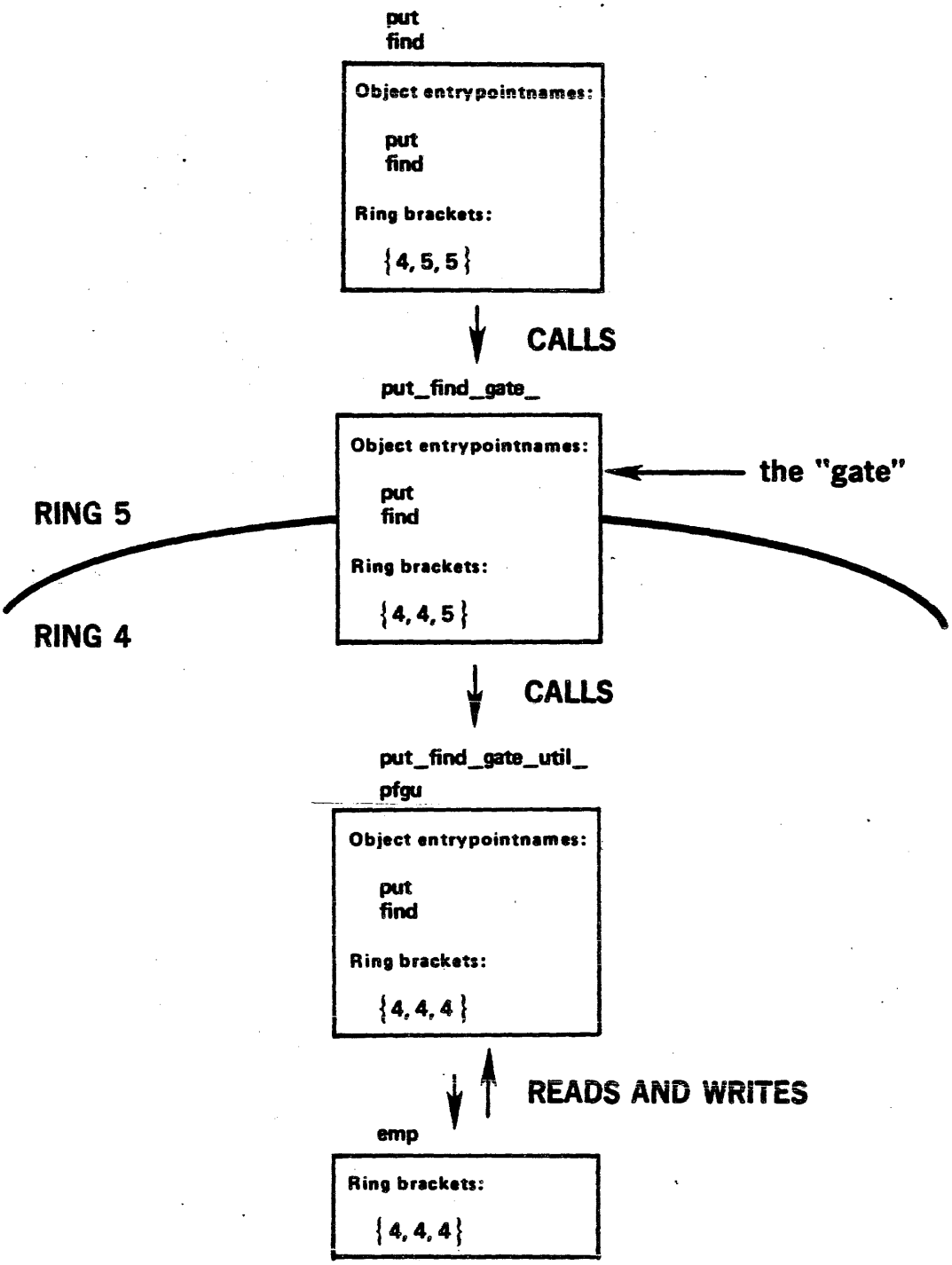
- WHY GATES SHOULD BE WRITTEN IN alm

  - PREVIOUSLY IT WAS POSSIBLE TO ARTIFICIALLY JUMP INTO INNER CODE OF A GATE (POTENTIAL BREACH OF SECURITY)

  - alm ENABLES CAREFUL CONTROL OF OBJECT SEGMENT FORMAT (ONLY TRANSFER INSTRUCTIONS ARE PLACED AT BASE OF SEGMENT)

  - THE WRITER OF THE GATE SETS THE ENTRY POINT BOUND OF THE PROGRAM EQUAL TO THE END OF THE TRANSFER INSTRUCTIONS

CROSSING RING BOUNDARIES



- USE hcs_$set_entry_bound OR hcs_$set_entry_bound_seg TO SET ENTRY POINT BOUND

  - call hcs_$set_entry_bound (dir_name, entryname, entry_bound, code);

  - call hcs_$set_entry_bound_seg (seg_ptr, entry_bound, code);

  - SETS A HARDWARE ENFORCED LIMIT ON ENTRY POINT OFFSET

  - IF entry_bound IS 0 THE MECHANISM IS DISABLED

  - ENTRY BOUND MAINTAINED IN CONTAINING DIRECTORY AND BUILT INTO THE SEGMENT DESCRIPTOR WORD (SDW) WHEN THE SEGMENT IS MADE KNOWN

  - OBJECT SEGMENT ITSELF IS UNCHANGED

EXAMPLE

```
!print bound_prog.pl1 1

bound_prog: proc;
dcl  hcs_$set_entry_bound entry (char (*), char (*),
     fixed bin (14), fixed bin (35));
dcl  code fixed bin (35);

          call hcs_$set_entry_bound (">udd>F15D>doodle",
               "bound_prog", 10, code);
     end bound_prog;

r 14:02 0.066  2

!bound_prog
r 14:02 0.081  3

!bound_prog

Error:  Attempt by cu_!373
(>system_library_1>bound_command_loop_)
to access >udd>F15D>doodle>bound_prog!16
which is beyond the entry bound for the gate.
r 14:02 0.164 23 level 2
```

RINGS

VALIDATION LEVEL

- POTENTIAL PROBLEM:
  - RING OF EXECUTION KEPT IN REGISTER IN THE PROCESSOR
  - RING OF EXECUTION KEEPS CHANGING
  - ASSUME A SEGMENT IS BEING CREATED
  - HOW DO SYSTEM SUBROUTINES ASSIGN PROPER RING BRACKETS?

- VALIDATION LEVEL
  - MEANS BY WHICH INNER RING (CALLED) PROCEDURE "KNOWS" THE LEVEL OF PRIVILEGE OF THE OUTER RING (CALLING) PROCEDURE
  - VALIDATION LEVEL CAN BE CHANGED
  - CANNOT BE SET LOWER THAN RING OF EXECUTION
  - VALIDATION LEVEL CHANGE USED FOR EXAMPLE:
    - TO CREATE A MAILBOX
    - BY A SUBSYSTEM WISHING TO CREATE A SEGMENT IN INNER RING

- cu_$level_get (AG93)

  - call cu_$level_get (level)

  - RETURNS THE CURRENT VALIDATION LEVEL

  - PRIMARILY USED PRIOR TO A CALL TO cu_$level_set TO SAVE THE CURRENT VALIDATION LEVEL

- cu_$level_set (AG93)

  - call cu_$level_set (level)

  - ALLOWS THE CALLER TO CHANGE THE CURRENT VALIDATION LEVEL

  - NEW LEVEL MUST BE >= CURRENT RING OF EXECUTION

- hcs_$get_user_effmode

  - call hcs_$get_user_effmode (dir_name, entryname, user_id, ring, mode, code);

  - RETURNS THE EFFECTIVE MODE FOR THE SPECIFIED RING

CROSS RING I/O

- AN ATTEMPT TO DO "CROSS RING I/O" USUALLY RESULTS IN A FATAL PROCESS ERROR
  - REASON: IOCB'S ARE PER RING
  - TYPICAL EXAMPLE: CALLING com_err_ IN AN INNER RING
- "CROSS RING I/O" IS ALLOWED USING THE FOLLOWING
  - cross_ring_
    - AN I/O MODULE WHICH ALLOWS AN OUTER RING TO ATTACH A SWITCH (BASICALLY AS A SYNONYM) TO A PREEXISTING SWITCH IN AN INNER RING, AND TO PERFORM I/O OPERATIONS BY FORWARDING I/O FROM THE ATTACHMENT IN THE OUTER RING THROUGH A GATE TO THE INNER RING
    - AN INNER RING SWITCH MUST BE ATTACHED WHILE IN THE INNER RING BEFORE cross_ring_ CAN BE USED TO ATTACH OUTER RING SWITCH
  - cross_ring_io_$allow_cross
    - call cross_ring_io_$allow_cross (switch_name, ring, code);
    - CALL MUST BE MADE IN THE INNER RING BEFORE THE OUTER RING ATTEMPTS TO ATTACH TO THIS SWITCH WITH cross_ring_

```
!pat
  user_i/o              tty_ -login_channel
            stream_input_output
  user_input            syn_ user_i/o
  user_output           syn_ user_i/o
  error_output          syn_ user_i/o


!pr cross.pl1 1

 cross:   proc;
    dcl gate$allow entry;
    dcl iox_$get_chars entry (ptr, ptr, fixed bin(21),
                              fixed bin(21), fixed bin(35));
    dcl iox_$attach_name entry (char(*), ptr, char(*), ptr,
                                fixed bin(35));
    dcl iox_$open entry (ptr, fixed bin, bit (1) aligned,
                         fixed bin (35));
    dcl com_err_ entry() options(variable);
    dcl code fixed bin (35);
    dcl iocb ptr;
    dcl buffer char (20);

    call gate$allow;

    call iox_$attach_name ("outer", iocb, "cross_ring_ file 4",
                                          null(), code);

    call iox_$open (iocb, 3, "0"b, code);

 end cross;


!pr gate.alm 1

          include   gate_macros
          gate_info
          gate      allow,allow,allow,0
          end
```

```
!pr allow.pl1 1

  allow:  proc;
    dcl cross_ring_io_$allow_cross entry (char(*), fixed bin,
                                          fixed bin(35));
    dcl iox_$attach_name entry (char(*), ptr, char(*), ptr,
                                          fixed bin(35));
    dcl iox_$open entry (ptr, fixed bin, bit(1) aligned, fixed bin(35));
    dcl code fixed bin(35);
    dcl com_err_ entry() options(variable);
    dcl iocb ptr;
    dcl null builtin;
    dcl cu_$level_set entry (fixed bin);
    dcl cu_$level_get entry (fixed bin);
    dcl old_level fixed bin;
    dcl get_ring_ entry() returns(fixed bin(3));

    call cu_$level_get (old_level);

    call cu_$level_set (get_ring_());

    call iox_$attach_name ("file", iocb, "vfile_ >udd>MED>nd>gate>file",
                           null(), code);

    call cross_ring_io_$allow_cross ("file", 5, code);

    call cu_$level_set (old_level);

 end allow;


!st [wd] -rb
  5, 5


!st file cross gate allow -rb
       >udd>MED>NDibble>gate>file
  4, 4, 4
       >udd>MED>NDibble>gate>cross
  4, 5, 5
       >udd>MED>NDibble>gate>gate
  4, 4, 5
       >udd>MED>NDibble>gate>allow
  4, 4, 4
```

CROSS RING I/O

```
!cross


!pat
  user_i/o              tty_ -login_channel
            stream_input_output
  user_input            syn_ user_i/o
  user_output           syn_ user_i/o
  error_output          syn_ user_i/o
  outer                 cross_ring_ file 4  stream_input_output


!io put_chars outer "line 1"
!io put_chars outer "line 2"
!io position outer -1
!io get_line outer
  io_call:7 characters returned.line 1


!pr file
  print: Incorrect access on entry. >udd>MED>NDibble>gate>file
```

- TWO MAJOR POINTS TO REMEMBER

  - WORKING DIRECTORIES ARE PER RING

  - MUST SET VALIDATION LEVEL TO INNER RING BEFORE CREATING INNER 'IOCB'

TOPIC V

The Command Environment

	Page
Introduction	5-1
Modifying the Standard Command Environment	5-2
Current Ready Procedure	5-6
Current Command Processor	5-8
Command Level Intermediary	5-10
Some Miscellaneous cu_ Entry Points	5-12
An Example	5-15

- THE SUBSYSTEM DESIGNER HAS THE CAPABILITY OF MODIFYING SEVERAL DIFFERENT ASPECTS OF THE COMMAND ENVIRONMENT

- THE cu_ (COMMAND UTILITY) SUBROUTINE ({1,5,5} PROCEDURE WRITTEN IN alm) IS THE TOOL USED BY SUBSYSTEM DESIGNERS TO ACCOMPLISH THE FOLLOWING BASIC TASKS:

  - WRITING COMMAND OR ACTIVE FUNCTION PROCEDURES

    cu_$arg_count
    cu_$arg_ptr
    cu_$af_arg_count
    cu_$af_return_arg
    cu_$af_arg_ptr

  - MODIFYING THE STANDARD COMMAND ENVIRONMENT

  - WRITING A COMMAND PROCESSOR

    cu_$generate_call

- THE STANDARD COMMAND ENVIRONMENT IS PROVIDED TO ALLOW THE USER TO PROCESS HIS COMMAND REQUESTS, EXECUTE HIS PROGRAMS, AND SO ON

- THE BASIC COMMAND ENVIRONMENT HAS THE FOLLOWING CHARACTERISTICS:

  - 'listen_' IS INVOKED AT PROCESS START UP TIME AND IS ALWAYS RETURNED TO FOLLOWING THE EXECUTION OF A COMMAND (OR FOLLOWING A PROGRAM ABORT)

    - THIS "LISTENER" ACCEPTS INPUT FROM THE USER'S TERMINAL AND PASSES SUCH INPUT TO THE "CURRENT COMMAND PROCESSOR" FOR FURTHER PROCESSING

    - EVERY TIME CONTROL RETURNS BACK TO THE "LISTENER", THE 'listen_' PROGRAM INVOKES THE "CURRENT READY PROCEDURE"

  - THE "CURRENT COMMAND PROCESSOR" PROCESSES THE INPUT LINE TYPED BY THE USER AND PASSED TO IT BY THE LISTENER

    - THE STANDARD COMMAND PROCESSOR (command_processor_) FIRST DOES SUCH THINGS AS EXPANDING OUT ANY ITERATION LOOPS (PARENTHESES) AND EVALUATING ACTIVE FUNCTIONS (BRACKETS)

    - THE STANDARD COMMAND PROCESSOR THEN DEVELOPS A CALL TO THE APPROPRIATE COMMAND OR USER PROGRAM

  - WHENEVER A QUIT OR OTHER "UNCLAIMED" SIGNAL (CONDITION NOT HANDLED BY USER) ARISES A "NEW LEVEL" OF THE LISTENER IS INVOKED BY "REENTERING COMMAND LEVEL"

    - WHEN THE DEFAULT HANDLER DECIDES TO REENTER COMMAND LEVEL, THE "CURRENT COMMAND LEVEL INTERMEDIARY" WILL BE INVOKED

- STANDARD COMMAND ENVIRONMENT CONCEPTS

  - CURRENT READY PROCEDURE

    - IS INVOKED WHEN LISTENER (OR ANY OTHER PROCEDURE) CALLS cu_$ready_proc

    - IS, BY DEFAULT, print_ready_message_, WHICH MERELY PRINTS THE READY MESSAGE

    - MAY BE SET BY A CALL TO cu_$set_ready_procedure

    - MAY BE DETERMINED BY A CALL TO cu_$get_ready_procedure

  - CURRENT COMMAND PROCESSOR

    - IS INVOKED WHEN LISTENER (OR ANY OTHER PROCEDURE) CALLS cu_$cp

    - IS, BY DEFAULT, command_processor_

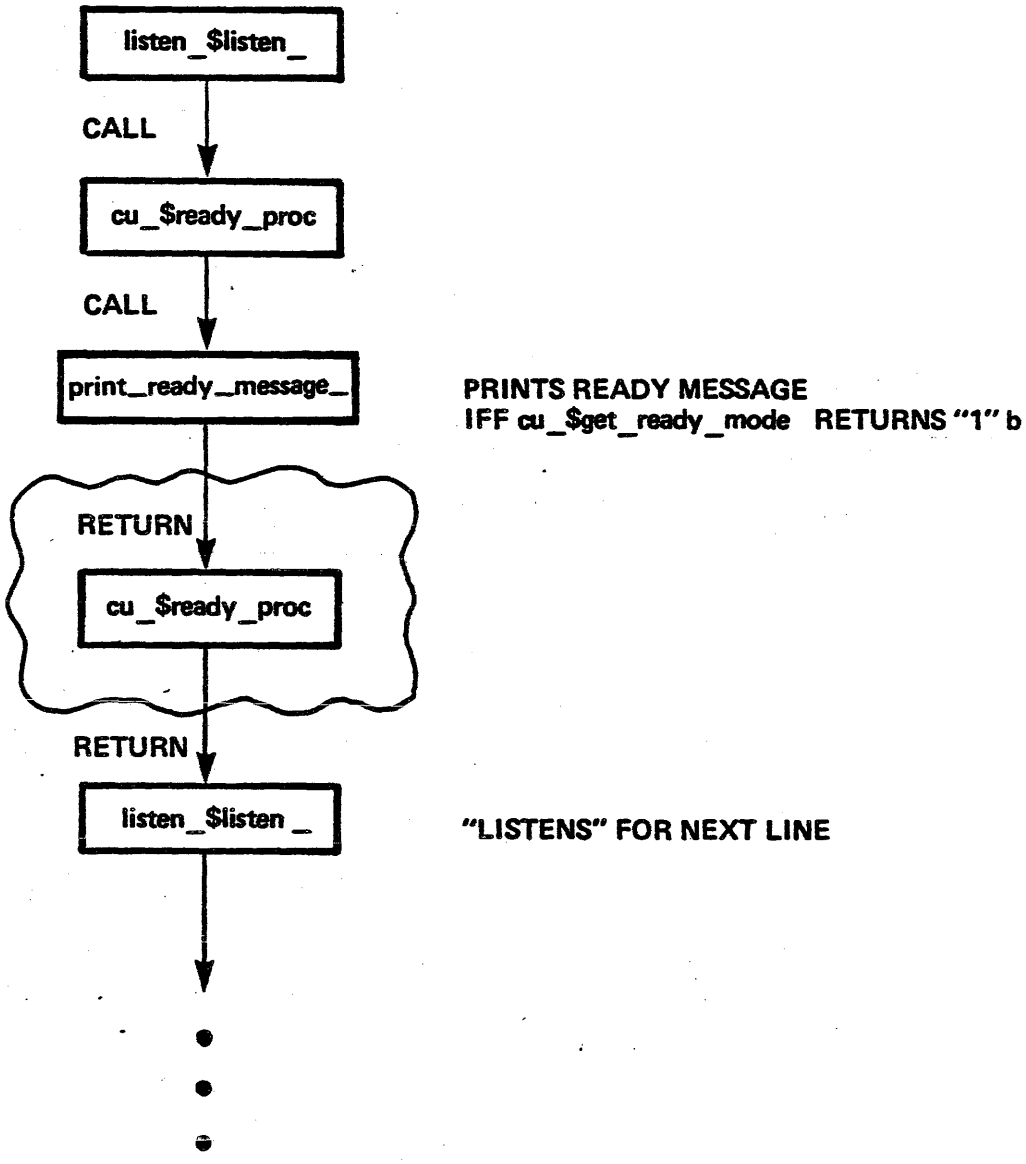
    - MAY BE SET BY A CALL TO cu_$set_command_processor (NOTE THE 'abbrev' COMMAND)

    - MAY BE DETERMINED BY A CALL TO cu_$get_command_processor

- CURRENT COMMAND LEVEL INTERMEDIARY
  - IS INVOKED WHEN DEFAULT ERROR HANDLER (OR ANY OTHER PROCEDURE) CALLS cu_$cl
  - IS, BY DEFAULT FOR INTERACTIVE PROCESSES, get_to_cl_$unclaimed_signal WHICH REENTERS COMMAND LEVEL VIA A CALL TO listen_$release_stack
  - MAY BE SET BY A CALL TO cu_$set_cl_intermediary
  - MAY BE DETERMINED BY A CALL TO cu_$get_cl_intermediary

This Page Intentionally Left Blank

WHERE THE CURRENT READY PROCEDURE FITS IN



- MANIPULATING THE READY PROCEDURE

  - cu_$ready_proc (AG93)

    - CALLS THE CURRENT READY PROCEDURE

  - cu_$set_ready_procedure (AG93)

    - ESTABLISHES THE SPECIFIED PROCEDURE AS THE CURRENT READY PROCEDURE

  - cu_$get_ready_procedure (AG93)

    - RETURNS A NULL ENTRY VALUE IF THE CURRENT READY PROCEDURE IS THE DEFAULT (print_ready_message_)

    - OTHERWISE, RETURNS THE PL/1 ENTRY VALUE OF THE CURRENT READY PROCEDURE

  - cu_$set_ready_mode (AG93)

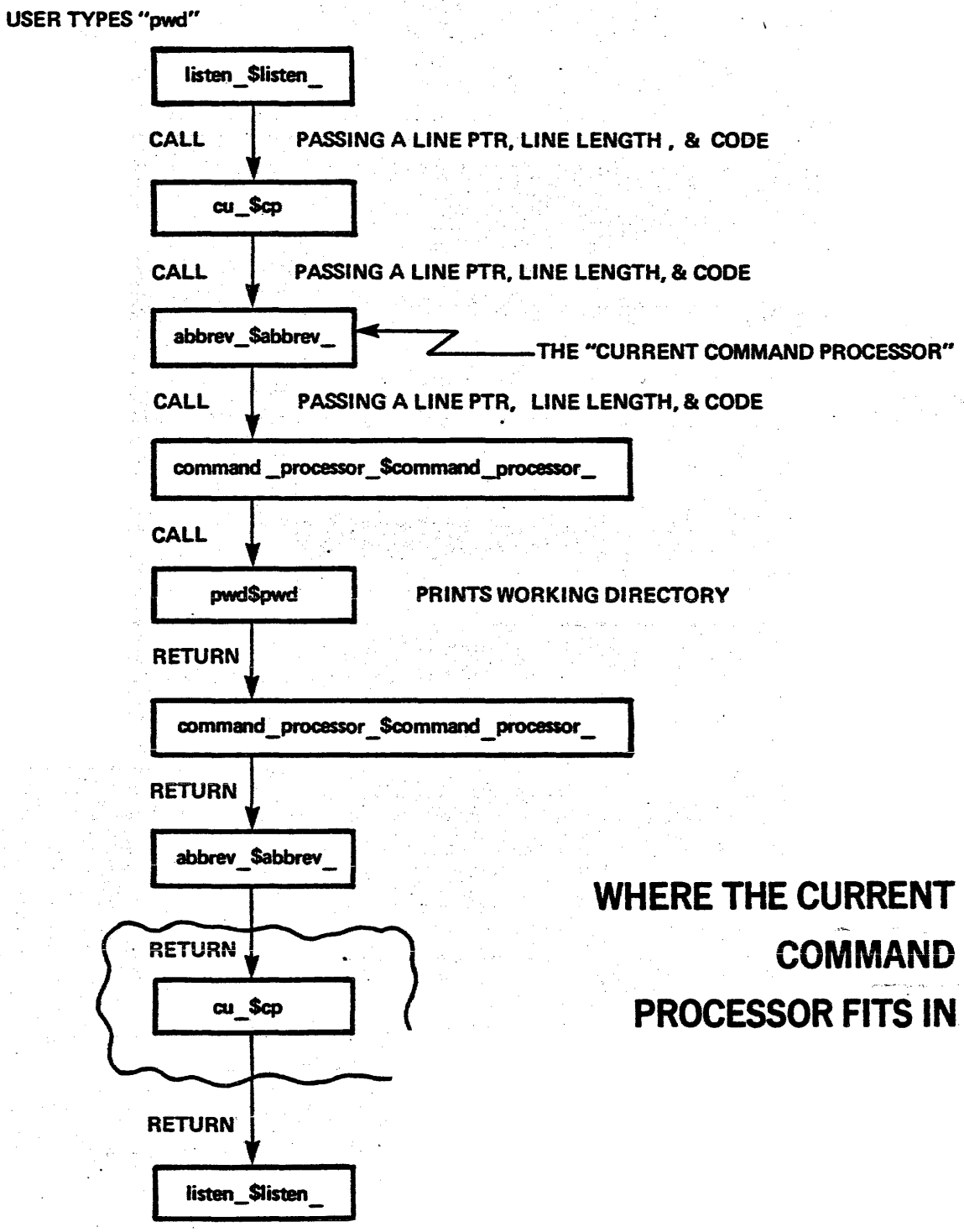
    - SETS OR RESETS THE "STATIC READY MODE" SWITCH

    - THE CURRENT READY PROCEDURE CAN (BUT NEEDN'T) CHECK THIS SWITCH TO SEE WHETHER TO PRINT OR NOT (NOTE THE 'ready_on', AND 'ready_off' COMMANDS)

  - cu_$get_ready_mode (AG93)

    - RETURNS THE VALUE OF THE "STATIC READY MODE" SWITCH

CURRENT COMMAND PROCESSOR



WHERE THE CURRENT COMMAND PROCESSOR FITS IN

- MANIPULATING THE CURRENT COMMAND PROCESSOR

  - cu_$cp (AG93)

    - INVOKES THE CURRENT COMMAND PROCESSOR, PASSING TO IT AN INPUT LINE POINTER, LINE LENGTH AND CODE

    - BESIDES THE LISTENER, USED ALSO BY SUBSYSTEMS HONORING AN "e" OR ".." REQUEST

  - cu_$set_command_processor (AG93)

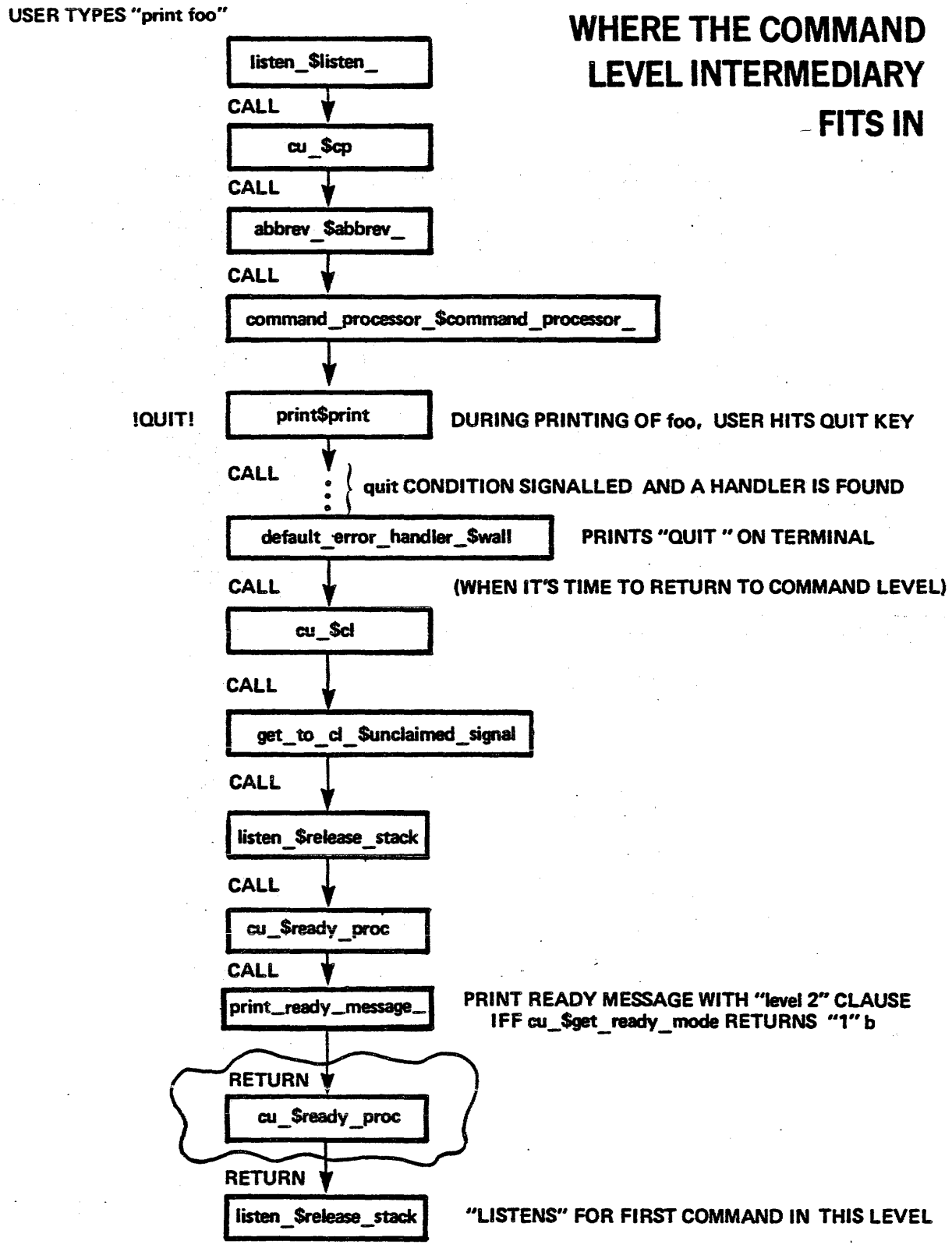
    - ESTABLISHES THE SPECIFIED PROCEDURE AS THE CURRENT COMMAND PROCESSOR (BY SPECIFYING AN ENTRY VALUE)

  - cu_$get_command_processor (AG93)

    - RETURNS A NULL ENTRY VALUE IF THE CURRENT COMMAND PROCESSOR IS THE DEFAULT (command_processor_$command_processor_)

    - OTHERWISE RETURNS THE ENTRY VALUE OF THE CURRENT COMMAND PROCESSOR

COMMAND LEVEL INTERMEDIARY



- MANIPULATING THE COMMAND LEVEL INTERMEDIARY

  - cu_$cl (AG93)

    - INVOKES THE CURRENT COMMAND LEVEL INTERMEDIARY

    - CALLED BY THE STANDARD ERROR HANDLERS

  - cu_$set_cl_intermediary (AG93)

    - ESTABLISHES THE SPECIFIED PROCEDURE AS THE CURRENT COMMAND LEVEL INTERMEDIARY

    - NOTE THAT AN INTERMEDIARY IS USED IN ABSENTEE PROCESSES TO FORCE PROCESS TERMINATION "WHEN AN ATTEMPT IS MADE TO REENTER COMMAND LEVEL"

  - cu_$get_cl_intermediary (AG93)

    - RETURNS A NULL ENTRY VALUE IF THE CURRENT COMMAND LEVEL INTERMEDIARY IS THE DEFAULT (get_to_cl_$unclaimed_signal)

    - OTHERWISE RETURNS THE ENTRY VALUE OF THE CURRENT COMMAND LEVEL INTERMEDIARY

SOME MISCELLANEOUS CU ENTRY POINTS

- cu_$decode_entry_value (AG93)

  - call cu_$decode_entry_value (entry_value, ep_ptr, env_ptr);
  - EXTRACTS THE POINTER COMPONENTS OF A PL/I ENTRY VALUE
  - USEFUL FOR DETERMINING IF AN ENTRY VALUE IS NULL
  - NOTE: RECENTLY REPLACED BY codeptr AND environmentptr BUILTINS

- cu_$arg_list_ptr (AG93)

  - RETURNS A POINTER TO THE ARGUMENT LIST STRUCTURE PASSED TO THE CALLER
  - GENERALLY USED BY SUBROUTINES WHICH ARE CALLED WITH A VARYING NUMBER OF ARGUMENTS OF VARYING DATA TYPES (ioa FOR INSTANCE), TO ALLOW EXAMINATION OF THE ARGUMENT LIST DIRECTLY
  - SEE ALSO decode_descriptor_ IN AK92

SOME MISCELLANEOUS CU ENTRY POINTS

- cu_$arg_ptr_rel (AG93)
  - REMINISCENT OF cu_$arg_ptr
  - ALLOWS A PROCEDURE TO REFERENCE THE nth ARGUMENT PASSED TO ANOTHER PROCEDURE, GIVEN A POINTER TO THAT OTHER PROCEDURE'S ARGUMENT LIST
  - QUESTION: HOW WOULD A PROCEDURE OBTAIN THE arglist_ptr OF ANOTHER PROCEDURE?
    - IT COULD BE PASSED SUCH A POINTER
    - IT COULD LOOK IN STACK FRAME OF OTHER PROCEDURE

- GENERATING A CALL GIVEN AN ENTRY VALUE

  - cu_$generate_call (AG93)

    - GENERATES A STANDARD CALL TO THE SPECIFIED PROCEDURE (DESIGNATED BY AN ENTRY VALUE) WITH A SPECIFIED ARGUMENT LIST

    - DESIGNED PRIMARILY TO BE USED BY COMMAND PROCESSORS THAT CALL A COMMAND WITH AN ARGUMENT LIST BUILT FROM A COMMAND LINE INPUT FROM A TERMINAL

    - IS PREFACED BY A CALL TO hcs_$make_entry, WHICH ACCEPTS PATHNAMES

AN EXAMPLE

```
!print change_cl.pl1 1

change_cl: proc;

dcl  codeptr builtin,
     cu_$get_cl_intermediary entry (entry),
     cu_$set_cl_intermediary entry (entry),
     cu_$get_command_processor entry (entry),
     ioa_ entry options (variable);

dcl  var_entry entry variable,
     my_intermediary entry;


/* FIND OUT THE CURRENT COMMAND PROCESSOR */
          call cu_$get_command_processor (var_entry);
          call ioa_ ("Current command processor is ^p", codeptr (var_entry));


/* FIND OUT THE CURRENT INTERMEDIARY */
          call cu_$get_cl_intermediary (var_entry);
          call ioa_ ("Current intermediary is ^p", codeptr (var_entry));

/* NOW SET MY OWN INTERMEDIARY */
          call cu_$set_cl_intermediary (my_intermediary);

     end change_cl;
```

AN EXAMPLE

```
r 19:26 0.132 0

!print my_intermediary.pl1 1

my_intermediary: proc;

dcl  get_to_cl_$unclaimed_signal entry;
dcl  ioa_ entry options (variable);

          call ioa_ ("TYPE 'start' TO RESTART PROCESS
TYPE 'release' OR 'rl' TO DISCARD STACK HISTORY");
          call get_to_cl_$unclaimed_signal;
     end my_intermediary;


r 19:26 0.069 0

!change_cl
Current command processor is 305|2676
Current intermediary is 77777|1
r 19:27 0.089 0

!lrn 305

   305 >sss>bound_full_cp_
do
response
ab
exec_com
r 19:27 0.105 0

!dcn >sss>bound_full_cp_ 2676
  2676    abbrev|334

r 19:28 0.096 2
```

AN EXAMPLE

```
!(QUIT)
QUIT
TYPE 'start' TO RESTART PROCESS
TYPE 'release' OR 'rl' TO DISCARD STACK HISTORY
r 19:29 0.168 4 level 2

!probe

Condition quit raised at block|154 (level 6).
stack
 13          simple_command_processor|12211
 12          command_processor_|11014
 11          abbrev_|7507
 10          release_stack|7755
  9          unclaimed_signal|27010
  8          my_intermediary (line 8)
  7          wall|2602
  6          block|154                                          quit
  5          tty_get_line|5763
  4          audit_get_line|5073
  3          listen_|7666
  2          project_start_up_|41673
  1          user_init_admin_|42376 (alm)
!q
r 19:29 0.980 81 level 2

!release
r 19:29 0.044 0

!change_cl
Current command processor is 305|2676
Current intermediary is 456|26
r 19:29 0.046 0

!lrn 456

   456 >udd>MED>NDibble>my_intermediary
my_intermediary
r 19:30 0.047 0
```

YOU ARE NOW READY FOR WORKSHOP #2

TOPIC VI

Advanced Multics I/O

	Page
Review	6-1
Control Orders	6-2
Useful tty_ Control Orders	6-3
Useful vfile_ Control Orders	6-7
Review of IOCB's	6-9
Synonyming	6-14
iox_ Entry Points Used in I/O Modules	6-19

REVIEW

- I/O SYSTEM BASIC CHARACTERISTICS:
  - LOGICAL INPUT/OUTPUT REQUESTS ARE USED RATHER THAN DEVICE-SPECIFIC PHYSICAL REQUESTS
  - DEVICE INDEPENDENCE IS ACHIEVED VIA THE Multics I/O SWITCH MECHANISM
  - ALL I/O REQUESTS ARE DIRECTED TO A "SWITCH", WHICH IS "ATTACHED" BY A DEVICE-DEPENDENT PROGRAM, CALLED AN I/O MODULE, TO A PARTICULAR DEVICE OR FILE
  - THE SUPPORTING DATA STRUCTURE OF A SWITCH IS AN I/O CONTROL BLOCK (IOCB)

- ALL I/O OPERATIONS CAN BE PERFORMED AT THREE BASIC LEVELS:
  - LANGUAGE LEVEL - 'open', 'close', 'get', 'read', 'put', 'write'
  - COMMAND LEVEL - THE 'io_call' COMMAND
  - SUBROUTINE LEVEL - THE 'iox_' SUBROUTINE

CONTROL ORDERS

- CONTROL OPERATIONS ARE ONE EXAMPLE OF EXTENDED POWER OF iox_ OVER LANGUAGE I/O

- SOME I/O MODULES SUPPORT 'control' OPERATIONS AND SOME DO NOT

- THEY ARE INVOKED BY A CALL TO iox_$control

- COMPLETE DESCRIPTIONS OF THE I/O MODULES, AND THE CONTROL ORDERS THEY SUPPORT CAN BE FOUND IN EITHER THE "MPM - Subroutines", ORDER NUMBER AG93, THE "MPM - Peripheral Input/Output", ORDER NUMBER AX49, OR THE "Communications Input/Ouput" MANUAL, ORDER NUMBER CC92.

- THE SUBSYSTEM DESIGNER MAY WANT TO MAKE USE OF SOME OF THE 'control' OPERATIONS SUPPORTED BY THE tty_ AND vfile_ I/O MODULES

USEFUL TTY CONTROL ORDERS

- tty_ SUPPORTS THE FOLLOWING CONTROL ORDERS (DOCUMENTED IN CC92)

  - abort
    resetread
    resetwrite

    - THESE FLUSH BOTH THE INPUT AND OUTPUT INTERMEDIATE BUFFERS (abort), THE INPUT BUFFER (resetread) OR THE OUTPUT BUFFER (resetwrite) FOR AN OPENED SWITCH

  - hangup

    - DISCONNECTS THE TELEPHONE LINE CONNECTION OF THE TERMINAL, IF POSSIBLE

  - listen

    - SENDS A WAKEUP TO THE PROCESS ONCE THE LINE ASSOCIATED WITH THIS DEVICE IDENTIFIER IS DIALED UP (SEE THE DISCUSSION OF 'DIALING TERMINALS TO A PROCESS')

  - terminal_info

    - RETURNS INFORMATION ABOUT THE DEVICE TO WHICH THE MODULE IS ATTACHED IN A USER-SUPPLIED STRUCTURE
      - ANSWERBACK-DERIVED TERMINAL ID
      - TERMINAL TYPE
      - LINE TYPE
      - BAUD RATE

USEFUL TTY CONTROL ORDERS

- quit_enable
  quit_disable

  - CAUSE 'quit' SIGNAL PROCESSING TO BE ENABLED OR DISABLED FOR THIS DEVICE (FOR EXAMPLE, THE STANDARD PROCESS CREATION CYCLE PROGRAMS ENABLE QUITS ONLY ONCE CONTROL HAS PASSED OUT TO THE USER RING, TO PREVENT A 'quit' FROM ALLOWING A PROCESS TO GAIN CONTROL IN AN INNER RING)

  - NOTE: EVEN IF TERMINAL QUITS ARE DISABLED, IT IS POSSIBLE TO SIGNAL 'quit' IN A USER PROGRAM, OR VIA THE 'signal' COMMAND

USEFUL TTY CONTROL ORDERS

- start
  - CAUSES A WAKEUP TO BE SIGNALLED ON THE EVENT CHANNEL ASSOCIATED WITH THIS DEVICE- THE REQUEST IS USED TO RESTART PROCESSING ON A DEVICE WHOSE WAKEUP MAY HAVE BEEN LOST OR DISCARDED (PERHAPS BECAUSE OF BEING INTERRUPTED BY AN ipc_ OR timer_manager_ CALLED ROUTINE)

- printer_off
  printer_on
  - CAUSE THE PRINTER MECHANISM OF THE TERMINAL TO BE TEMPORARILY DISABLED OR REENABLED (IF IT IS PHYSICALLY POSSIBLE FOR THE TERMINAL TO DO SO) - THIS MAY BE USEFUL FOR SUCH THINGS AS ACCEPTING PASSWORDS

- set_delay
  get_delay
  - SET, OR RETURN, THE NUMBERS OF DELAY CHARACTERS ASSOCIATED WITH THE OUTPUT OF CARRIAGE MOTION CHARACTERS
  - DEFAULT VALUES CAN BE USED, OR DELAY CHARACTERS CAN BE SPECIFIED FOR VERTICAL AND HORIZONTAL NEW-LINE OUTPUTS, FORMFEEDS, AND THE LIKE

- set_editing_chars
  get_editing_chars
  - CHANGE, OR FIND OUT, WHAT CHARACTERS ARE BEING USED FOR EDITING INPUT
  - THE ERASE AND KILL CHARACTERS CAN BE SET TO ANY CHARACTERS DESIRED SUBJECT TO SOME RESTRICTIONS (E.G., NO CARRIAGE-MOVEMENT CHARACTERS, NOT 'NUL' OR SPACE, AND SO ON)

USEFUL TTY CONTROL ORDERS

- set_input_translation
  get_input_translation

  - SETS, OR READS, A TABLE WHICH IS USED FOR TRANSLATION OF TERMINAL INPUT TO ASCII

- set_output_translation
  get_output_translation

  - SETS, OR READS, A TABLE WHICH IS USED FOR TRANSLATING ASCII CHARACTERS TO THE CODE TO BE SENT TO THE TERMINAL

- set_input_conversion
  get_input_conversion

  - WRITE OR READ A TABLE WHICH IS USED IN CONVERTING INPUT TO IDENTIFY ESCAPE SEQUENCES AND CERTAIN SPECIAL CHARACTERS

- set_output_conversion
  get_output_conversion

  - WRITE OR READ A TABLE USED IN FORMATTING OUTPUT TO IDENTIFY CERTAIN KINDS OF SPECIAL CHARACTERS (SUCH AS NEWLINE, CARRIAGE RETURN, BACKSPACE, AND SO ON)

- set_special
  get_special

  - WRITE OR READ A TABLE THAT SPECIFIES 1-3 CHARACTER SEQUENCES TO BE SUBSTITUTED FOR CERTAIN OUTPUT CHARACTERS, AND CHARACTERS WHICH ARE TO BE INTERPRETED AS PARTS OF ESCAPE SEQUENCES ON INPUT

USEFUL VFILE_ CONTROL ORDERS

- vfile_ SUPPORTS THE FOLLOWING CONTROL ORDERS
  - read_position
    - RETURNS THE ORDINAL POSITION OF THE NEXT RECORD (OR BYTE) AND THAT OF THE END OF THE FILE (RELATIVE TO THE FILE BASE)
  - seek_head
    - USED FOR FILES OPENED FOR KEYED-SEQUENTIAL-INPUT OR KEYED-SEQUENTIAL-UPDATE
    - LOCATES THE FIRST RECORD WITH A KEY WHOSE HEAD HAS THE SPECIFIED RELATION (=, >=, >) WITH A GIVEN SEARCH-KEY
    - USEFUL FOR APPLICATIONS WHICH MUST LOCATE THE FIRST RECORD THAT, FOR INSTANCE, BEGINS WITH "B"
    - APPLICATION: MDBM USES THIS WHEN LOCATING TUPLES GIVEN ONLY LEADING PORTION OF KEY
  - file_status
    - RETURNS VARIOUS ITEMS OF INFORMATION ABOUT THE FILE
    - SEE vfs COMMAND AND vfile_status_ SUBROUTINE

USEFUL VFILE CONTROL ORDERS

- get_key
  add_key
  delete_key
  reassign_key

  - MANIPULATE THE KEYS IN AN INDEXED FILE DIRECTLY (THUS ALLOWING SEVERAL KEYS TO BE ASSOCIATED WITH A GIVEN DATA RECORD, REASSIGNING THE DATA DESCRIPTOR OF A KEY TO ANOTHER DATA DESCRIPTOR, AND SO ON)

REVIEW OF IOCB'S

- RECALL:

  - AN 'IOCB' IS A STANDARD DATA STRUCTURE

  - IT IS THE PHYSICAL REALIZATION OF A SWITCH

  - THEY ARE FOUND IN THE USER'S PROCESS DIRECTORY

  - AN 'IOCB' IS CREATED BY iox_ WHEN A SWITCHNAME IS USED IN AN "ATTACH STATEMENT" OR "ATTACH COMMAND" FOR THE FIRST TIME IN A PROCESS

  - ONCE AN 'IOCB' IS CREATED, IT LIVES THROUGHOUT THE PROCESS (UNLESS EXPLICITLY DELETED)

  - THE PRINCIPAL COMPONENTS OF AN 'IOCB' ARE 'pointer' VARIABLES AND 'entry' VARIABLES

  - THERE IS ONE 'entry' VARIABLE FOR EACH I/O OPERATION, WITH THE EXCEPTION OF THE ATTACH OPERATION

  - TO PERFORM AN I/O OPERATION THRU THE SWITCH, THE APPROPRIATE ENTRY VALUE IN THE CORRESPONDING 'IOCB' IS CALLED

REVIEW OF IOCB'S

- WHEN iox_$attach_name IS CALLED IT:
  - INITIALIZES SOME OF THE ELEMENTS IN THE 'IOCB' STRUCTURE
  - CALLS <module_name>$<module_name>attach
    - THIS ENTRY POINT IN THE I/O MODULE FINISHES THE INITIALIZATION OF THE 'IOCB'
- IT IS THE RESPONSIBILITY OF THE I/O MODULE TO MAINTAIN THE ACCURACY OF THE 'IOCB'
- ONLY THE iox_ ENTRY POINTS RESULTING IN ATTACHMENT OF A SWITCH REQUIRE THE MODULE AS AN INPUT ARGUMENT
  - AFTER THAT TIME, THE 'IOCB' "POINTS TO" THE APPROPRIATE ENTRY POINTS IN THE APPROPRIATE MODULE (THE USER NEED ONLY PROVIDE A POINTER TO THE 'IOCB')

```
dcl 1 iocb aligned based,     /* I/O control block. */
    2 version fixed init(1), /* Version number of structure. */
    2 name char (32),        /* I/O name of this block. */
    2 actual_iocb_ptr ptr,   /* IOCB ultimately SYNed to. */
    2 attach_descrip_ptr ptr,/* Ptr to printable attach descrip. */
    2 attach_data_ptr ptr,   /* Ptr to attach data structure. */
    2 open_descrip_ptr ptr,  /* Ptr to printable open description. */
    2 open_data_ptr ptr,     /* Ptr to open data structure. */
    2 reserved bit (72),     /* Reserved for future use. */
    2 detach_iocb entry (ptr, fixed (35)),
              /* detach_iocb (p,s) */
    2 open entry (ptr, fixed, bit (1)aligned, fixed (35)),
              /* open(p,mode,not_used,s) */
    2 close entry (ptr, fixed (35)), /* close(p,s) */
    2 get_line entry (ptr, ptr, fixed (21), fixed (21), fixed (35)),
              /* get_line(p,bufptr,buflen,actlen,s */
    2 get_chars entry (ptr, ptr, fixed (21), fixed (21), fixed (35)),
              /* get_chars(p,bufptr,buflen,actlen,s) */
    2 put_chars entry (ptr, ptr, fixed (21), fixed (35)),
              /* put_chars(p,bufptr,buflen,s) */
    2 modes entry (ptr, char (*), char (*), fixed (35)),
              /* modes(p,newmode,oldmode,s) */
    2 position entry (ptr, fixed, fixed (21), fixed (35)),
              /* position(p,u1,u2,s) */
    2 control entry (ptr, char (*), ptr, fixed (35)),
              /* control(p,order,infptr,s) */
    2 read_record entry (ptr, ptr, fixed (21), fixed (21), (fixed (35)),
              /* read_record(p,bufptr,buflen,actlen,s) */
    2 write_record entry (ptr, ptr, fixed (21), fixed (35)),
              /* write_record(p,bufptr,buflen,s) */
    2 rewrite_record entry (ptr, ptr, fixed (21), fixed (35)),
              /* rewrite_record(p,bufptr,buflen,s) */
    2 delete_record entry (ptr, fixed (35)),
              /* delete_record(p,s) */
    2 seek_key entry (ptr, char (256) varying, fixed (21), fixed (35)),
              /* seek_key(p,key,len,s) */
    2 read_key entry (ptr, char (256) varying, fixed (21), fixed (35)),
              /* read_key(p,key,len,s) */
    2 read_length entry (ptr, fixed (21), fixed (35)),
              /* read_length(p,len,s) */;
```

```
/*  "HIDDEN" PORTION          */

    2 ios_compatibility_ptr, /* Ptr to old DIM's IOS transfer vector. */
    2 syn_inhibits bit(36),   /* Operations inhibited by SYN. */
    2 syn_father ptr,         /* IOCB immediately SYNed to. */
    2 syn_brother ptr,        /* Next IOCB SYNed as this one is. */
    2 syn_son ptr;            /* First IOCB SYNed to this one. */
```

OTHER STRUCTURES OF INTEREST

```
dcl 1 attach_description based aligned,
    2 length fixed bin(17),
    2 string char (0 refer (attach_description.length) );

dcl 1 open_description based aligned,
    2 length fixed bin(17),
    2 string char (0 refer (open_description.length) );
```

- iocb.attach_descrip_ptr

  - BY DEFINITION, IF THIS IS NULL, IOCB IS DETACHED

  - THE ATTACH DESCRIPTION OF AN IOCB SYNED TO ANOTHER IS A DESCRIPTION OF THE SYNONYMIZATION, NOT A COPY OF THE OTHER IOCB'S ATTACH DESCRIPTION

- iocb.attach_data_ptr
  - IS OPTIONALLY USED BY THE I/O MODULE TO LOCATE AN INFORMATION STRUCTURE WHOSE FORMAT AND CONTENT IS MODULE-DEPENDENT
  - I/O MODULES AREN'T ALLOWED TO STORE SUPPLEMENTARY ATTACH DATA ANYWHERE ELSE

- iocb.open_descrip_ptr
  - BY DEFINITION, IF NULL, THE IOCB IS CLOSED

- iocb.open_data_ptr
  - ANALOGOUS TO attach_data_ptr

- iocb.ios_compatibility
  - POINTS TO A MODULE THAT SIMULATES ALL FUNCTIONS OF THE OBSOLETE I/O SWITCHING MECHANISM, ios_

SYNONYMING

- SYNONYMING IS ACCOMPLISHED THRU THE USE OF THE syn_ I/O MODULE WHICH:
  - ATTACHES AN I/O SWITCH, x, AS A SYNONYM FOR ANOTHER SWITCH, y
  - THEREAFTER, PERFORMING AN OPERATION ON x (EXCEPT FOR DETACH) HAS THE SAME AFFECT AS PERFORMING IT ON y

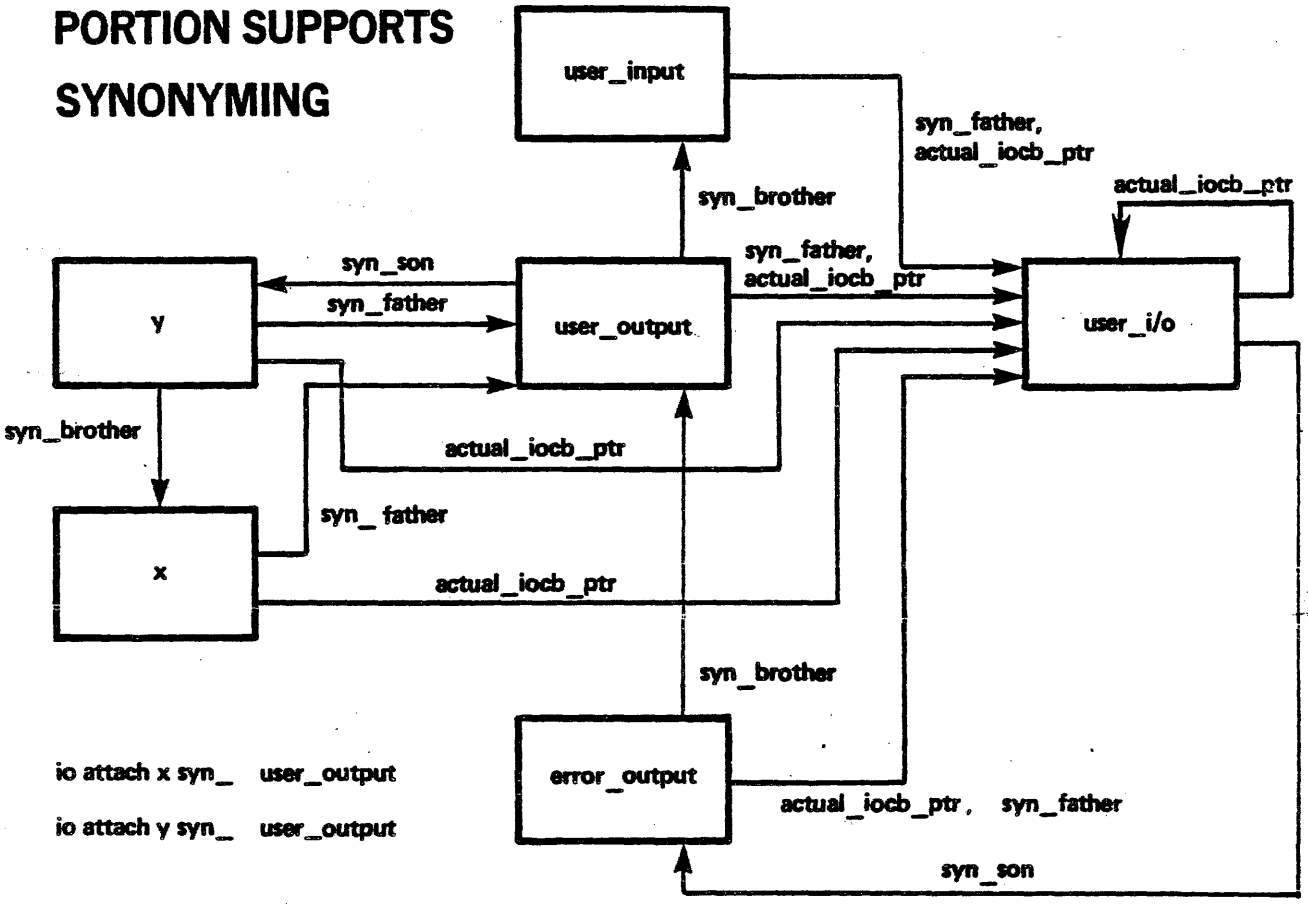
- THE ONLY WAY TO MAKE SYNONYMING BEHAVE DIFFERENTLY THAN DESCRIBED ABOVE IS TO USE THE -inhibit CONTROL ARGUMENT IN THE ATTACH DESCRIPTION
  - iocb.syn_inhibits
    - FIRST 15 BITS MAP TO iocb.open THRU iocb.read_length (detach CAN NOT BE INHIBITED)
    - WHEN A BIT IS ON, error_table_$no_operation IS RETURNED

```
!io attach y vfile_ my_file

!io open y stream_input

!io attach x syn_ y

!io print_iocb y
 IOCB "y" @ 257|27320
 SYN son is "x" @ 257|27160
 attach description: "vfile_ >udd>MEDmult>F15D>new>my_file",
                              attach data at 257|34524
 open description: "stream_input", open data at 257|27462
 detach_iocb      >sl1>bound_command_loop_$err_not_closed (271|2370)
 open              "
 close            >sss>bound_vfile_$close_file (370|55265)
 get_line         >sss>bound_vfile_$get_line_uns_file (370|722)
 get_chars        >sss>bound_vfile_$get_chars_uns_file (370|542)
 put_chars        >sl1>bound_command_loop_$err_no_operation (271|2360)
 modes             "
 position         >sss>bound_vfile_$position_uns_file (370|1142)
 control          >sss>bound_vfile_$control_uns_file (370|265)
 read_record      >sl1>bound_command_loop_$err_no_operation (271|2360)
 write_record      "
 rewrite_record    "
 delete_record     "
 seek_key          "
 read_key          "
 read_length       "


!dcn >sl1>bound_command_loop_ 2360
  2360    iox_|114
```

```
!io print_iocb x
 IOCB "x" @ 257|27160 (actual IOCB is "y" @ 257|27320)
 SYN father is "y" @ 257|27320
 attach description: "syn_ y", attach data at 257|34666
 open description: "stream_input", open data at 257|27462
 detach_iocb      >sl1>bound_command_loop_$syn_detach (271|73205)
 open             >sl1>bound_command_loop_$err_not_closed (271|2370)
 close            >sss>bound_vfile_$close_file (370|55265)
 get_line         >sss>bound_vfile_$get_line_uns_file (370|722)
 get_chars        >sss>bound_vfile_$get_chars_uns_file (370|542)
 put_chars        >sl1>bound_command_loop_$err_no_operation (271|2360)
 modes             "
 position         >sss>bound_vfile_$position_uns_file (370|1142)
 control          >sss>bound_vfile_$control_uns_file (370|265)
 read_record      >sl1>bound_command_loop_$err_no_operation (271|2360)
 write_record      "
 rewrite_record    "
 delete_record     "
 seek_key          "
 read_key          "
 read_length       "
```

```
!io attach inhibited syn_ y -inhibit close

!io print_iocb inhibited
 IOCB "inhibited" @ 257|27020 (actual IOCB is "y" @ 257|27320)
 SYN father is "y" @ 257|27320
 SYN brother is "x" @ 257|27160
 attach description: "syn_ y -inh close", attach data at 257|35402
 open description: "stream_input", open data at 257|27462
 detach_iocb      >sl1>bound_command_loop_$syn_detach (271|73205)
 open             >sl1>bound_command_loop_$err_not_closed (271|2370)
 close            >sl1>bound_command_loop_$err_no_operation (271|2360) (inh)
 get_line         >sss>bound_vfile_$get_line_uns_file (370|722)
 get_chars        >sss>bound_vfile_$get_chars_uns_file (370|542)
 put_chars        >sl1>bound_command_loop_$err_no_operation (271|2360)
 modes             "
 position         >sss>bound_vfile_$position_uns_file (370|1142)
 control          >sss>bound_vfile_$control_uns_file (370|265)
 read_record      >sl1>bound_command_loop_$err_no_operation (271|2360)
 write_record      "
 rewrite_record    "
 delete_record     "
 seek_key          "
 read_key          "
 read_length       "

!pat y x inhibited
 y                    vfile_ >udd>MEDmult>F15D>new>my_file
           stream_input
 x                    syn_ y
 inhibited            syn_ y -inh close

!io close inhibited
 io_call: Invalid I/O operation. inhibited

!pat y
 y                    vfile_ >udd>MEDmult>F15D>new>my_file
           stream_input

!io close x

!pat y x inhibited
 y                    vfile_ >udd>MEDmult>F15D>new>my_file
           (not open)
 x                    syn_ y
 inhibited            syn_ y -inh close
```

THE HIDDEN PORTION SUPPORTS SYNONYMING



IOX ENTRY POINTS USED IN I/O MODULES

- THE FOLLOWING iox_ ENTRY POINTS ARE GENERALLY OF USE TO USERS WRITING I/O MODULES:

  - iox_$propagate
    - REFLECTS MODIFICATIONS MADE TO AN ULTIMATE IOCB BACK TO ALL MEMBERS OF THE "SYNONYM FAMILY"
    - MUST BE CALLED AT CERTAIN POINTS IN THE I/O MODULE AND NOT UNDER ANY OTHER CIRCUMSTANCES
    - DOES INDEED PROPAGATE INHIBIT BITS BACKWARD TO THE SONS
  - iox_$find_iocb_n
    - USED TO FIND (ONE AT A TIME) ALL EXISTING IOCBS IN THE CALLING RING, WHETHER ATTACHED OR DETACHED
    - SEE print_attach_table COMMAND
  - iox_$look_iocb
    - RETURNS A POINTER TO THE IOCB FOR THE NAMED SWITCH IF IT EXISTS

- iox_$err_no_operation
  iox_$err_not_open
  iox_$err_not_closed
  iox_$err_not_attached

  - THESE ENTRY VALUES ARE ASSIGNED TO ENTRY VARIABLES IN THE IOCB IN ORDER TO RETURN AN ERROR CODE WHEN THAT ENTRY VARIABLE IS CALLED

  - THESE ENTRY POINTS SET THE VALUE OF THE 'code' ARGUMENT TO ONE OF THE FOLLOWING:

    error_table_$no_operation

    error_table_$not_open

    error_table_$not_closed

    error_table_$not_attached

TOPIC VII

Writing I/O Modules

	Page
Introduction	7-1
Implementation Rules	7-2
Entry Points of an I/O Module	7-4
Example of an I/O Module	7-7

RECOMMENDED READING: THE Multics USER RING I/O SYSTEM PLM (Order No. AN57) AND CHAPTER 4 IN THE SUBSYSTEM WRITERS' GUIDE.

- SOME INSTANCES IN WHICH A USER MIGHT WISH TO CREATE A NEW I/O MODULE ARE:

  - TO USE A PSEUDO DEVICE OR FILE

    - AN I/O MODULE COULD BE USED TO SIMULATE I/O TO/FROM A DEVICE OR FILE (discard_ IS AN EXAMPLE)

  - TO SUPPORT A NEW FILE TYPE, SUCH AS ONE IN WHICH RECORDS HAVE MULTIPLE KEYS

  - REINTERPRETING A FILE

    - AN I/O MODULE COULD BE DESIGNED TO OVERLAY A NEW STRUCTURE ON A STANDARD TYPE OF FILE (E.G., INTERPRETING AN UNSTRUCTURED FILE AS A SEQUENTIAL FILE BY CONSIDERING 80 CHARACTERS AS A RECORD

  - TO MONITOR A SWITCH

    - AN I/O MODULE COULD BE DESIGNED TO PASS OPERATIONS ALONG TO ANOTHER MODULE WHILE MONITORING THEM IN SOME WAY

    - SEE audit_

  - TO SUPPORT AN UNUSUAL DEVICE

    - BY WORKING THROUGH THE tty I/O MODULE IN THE 'RAW' MODE, ANOTHER I/O MODULE MIGHT TRANSMIT DATA TO/FROM A DEVICE THAT IS NOT A STANDARD Multics DEVICE

IMPLEMENTATION RULES

- SEVERAL IMPLEMENTATION RULES MUST BE FOLLOWED FOR PROPER OPERATION WITHIN THE Multics I/O ENVIRONMENT

- IN BRIEF, THE RULES ARE:

  - EXCEPT FOR THE ATTACH OPERATION, THE ENTRY DECLARATION AND PARAMETERS OF A ROUTINE THAT IMPLEMENTS AN I/O OPERATION ARE THE SAME AS THAT OF THE CORRESPONDING ENTRY IN iox_

    - FOR EXAMPLE, IT IS NOT PERMISSIBLE TO DECLARE THE ENTRY VALUE FOR THE 'put_chars' OPERATION AS ANYTHING BUT THE DECLARATION OF iox_$put_chars, WHICH IS 'entry (ptr, ptr, fixed bin(21), fixed bin(35))'

    - THE ATTACH OPERATION ACCEPTS FOUR ARGUMENTS:

      (ptr, (*)char(*) varying, bit(1) aligned, fixed bin (35))

      CORRESPONDING TO:

      (iocb_ptr, option_array, com_err_switch, code)

  - EXCEPT FOR ATTACH AND DETACH, THE "ULTIMATE" IOCB MUST BE REFERENCED USING THE VALUE OF iocb_ptr->iocb.actual_iocb_ptr - IT IS INCORRECT TO USE JUST iocb_ptr

  - IF AN I/O OPERATION CHANGES ANY VALUES IN THE ULTIMATE IOCB, THE I/O MODULE MUST CALL iox_$propagate BEFORE RETURNING

  - ALL I/O OPERATIONS MUST BE EXTERNAL ENTRY POINTS

- WHEN MODIFYING AN IOCB IPS (INTER PROCESS SIGNAL) INTERRUPT (quit, alrm OR cput) CANNOT BE TOLERATED

- I/O MODULES SHOULD MASK IPS SIGNALS AT FOLLOWS:

  - ESTABLISH AN any_other HANDLER THAT CALLS terminate_process_ IF MASKING IS IN EFFECT

  - CALL hcs_$set_ips_mask (0, mask)

  - CHANGE THE IOCB

  - CALL hcs_$reset_ips_mask (mask, mask)

ENTRY POINTS OF AN I/O MODULE

- AN I/O MODULE TYPICALLY HAS ENTRY POINTS FOR THE FOLLOWING:

  ATTACH OPERATION

  OPEN OPERATION

  CLOSE OPERATION

  DETACH OPERATION

  ANY OTHER OPERATION ENABLED BY THE ABOVE

- THE FOLLOWING IS A SIMPLIFIED SUMMARY OF THE STEPS TAKEN BY THE FIRST FOUR OF THE ABOVE ENTRY POINTS

  - TO AVOID ADDED CONFUSION, DETAILS ABOUT THE HANDLING OF CONTROL ORDERS AND MODES IS OMITTED

  - COMPLETE DOCUMENTATION OF THE FOLLOWING IS FOUND IN THE SWG

- MAJOR STEPS OF THE ATTACH OPERATION
  - SET iocb_ptr->iocb.open APPROPRIATELY
  - SET iocb_ptr->iocb.detach_iocb APPROPRIATELY
  - SET iocb_ptr->attach_descrip_ptr APPROPRIATELY
  - CALL iox_$propagate

- MAJOR STEPS OF OPEN OPERATION
  - SET actual_iocb_ptr->iocb.<operation> FOR EVEWY OPERATION THAT IS ALLOWED APPROPRIATELY
  - SET actual_iocb_ptr->open descrip_ptr APPROPRIATELY
  - CALL iox_$propagate

ENTRY POINTS OF AN I/O MODULE

- MAJOR STEPS OF CLOSE OPERATION
  - SET actual_iocb_ptr->iocb.open APPROPRIATELY
  - SET actual_iocb_ptr->iocb.detach_iocb APPROPRIATELY
  - SET actual_iocb_ptr->open_descrip_ptr TO NULL
  - CLEAN UP (SET BIT COUNTS, FREE STORAGE, ETC.)
  - CALL iox_$propagate

- MAJOR STEPS OF DETACH OPERATION
  - SET iocb_ptr->iocb.attach_descrip_ptr TO NULL
  - CALL iox_$propagate

EXAMPLE OF AN I/O MODULE

- THE FOLLOWING BEHAVES EXACTLY LIKE THE SYSTEM MODULE discard_ (IT IS NOT THE CODE FOR discard_ ALTHOUGH MUCH OF THE CODE IS VERY SIMILAR)

```
my_discard_attach: proc(iocb_ptr,option_array,com_err_switch,code);

          dcl option_array(*) char(*) varying;
          dcl buflen fixed(21);
          dcl bufptr ptr;
          dcl extend_bit bit(1) aligned;
          dcl infptr ptr;
          dcl iocb_ptr ptr;
          dcl key char(256) varying;
          dcl len fixed(21);
          dcl com_err_switch bit(1) aligned;
          dcl mode fixed;
          dcl newmode char(*);
          dcl oldmode char(*);
          dcl order char(*);
          dcl any_other condition;
          dcl blkptr ptr;
          dcl actual_iocb_ptr ptr;
          dcl code fixed(35);
          dcl mask bit(36) aligned;

%include iocb;

          dcl 1 block based (blkptr),
              2 attach_descrip aligned,
                3 length fixed bin(17) init (11),
                3 string char (11) init ("my_discard_"),
              2 open_descrip aligned,
                3 length fixed bin (17),
                3 string char (40);

          dcl free_area area based (get_system_free_area_());
          dcl get_system_free_area_ entry() returns(ptr);
          dcl com_err_ ext entry options(variable);
          dcl hcs_$set_ips_mask entry (bit(36) aligned, bit(36) aligned);
          dcl hcs_$reset_ips_mask entry (bit(36) aligned, bit(36) aligned);
          dcl iox_$propagate ext entry(ptr);
          dcl iox_$err_not_open entry() options(variable);
          dcl error_table_$bad_mode fixed(35) ext;
          dcl error_table_$not_detached fixed bin(35) ext;
          dcl error_table_$no_record fixed(35) ext;
          dcl error_table_$wrong_no_of_args fixed(35) ext;
          dcl error_table_$no_operation fixed(35) ext;
```

```
          dcl stream_output_mode fixed int static init(2);
          dcl sequential_output_mode fixed int static init(5);
          dcl keyed_sequential_output_mode fixed int static init(9);
          dcl direct_output_mode fixed int static init(12);

          dcl (addr,hbound,null,size) builtin;

     /* Start Executable Code */

       mask = "0"b;
       on any_other call handler;
       call hcs_$set_ips_mask ("0"b, mask);
       if hbound(option_array,1)>0 then
            call error (error_table_$wrong_no_of_args);
       if iocb_ptr->iocb.attach_descrip_ptr^=null() then
            call error (error_table_$not_detached);
       allocate block in (free_area);
       iocb_ptr->iocb.attach_descrip_ptr,
       iocb_ptr->iocb.attach_data_ptr = addr(attach_descrip);
       iocb_ptr->iocb.detach_iocb = my_discard_detach;
       iocb_ptr->iocb.open = my_discard_open;
       call iox_$propagate(iocb_ptr);
       call hcs_$reset_ips_mask(mask,mask);
       return;

          /* Internal procedure to handle all attach errors.
             Calls "com_err_" if the "com_err_switch" is set.
             In any case, returns to caller of attach external
             procedure with proper error code after ensuring
             that the IPS interrupt mask is restored. */

error:    proc(c);
          dcl c fixed(35);
          if mask^="0"b then call hcs_$reset_ips_mask(mask,mask);
          if com_err_switch then call
               com_err_ (c, "my_discard_");
          code = c;
          go to exit;
          end error;
exit:  return;

my_discard_detach: entry(iocb_ptr, code);
       code = 0;
       mask = "0"b;
       on any_other call handler;
       call hcs_$set_ips_mask ("0"b, mask);
       free iocb_ptr->iocb.attach_data_ptr -> block;
       iocb_ptr->iocb.attach_descrip_ptr = null();
       call iox_$propagate(iocb_ptr);
       call hcs_$reset_ips_mask(mask,mask);
       return;
```

```
my_discard_open: entry(iocb_ptr,mode,extend_bit,code);
      mask = "0"b;
      on any_other call handler;
      call hcs_$set_ips_mask ("0"b, mask);
      actual_iocb_ptr = iocb_ptr->iocb.actual_iocb_ptr;
      blkptr = actual_iocb_ptr->iocb.attach_data_ptr;

      if mode=stream_output_mode then do;
      blkptr->open_descrip.string = "stream_output";
      blkptr->open_descrip.length = 13;
      actual_iocb_ptr->iocb.put_chars = my_discard_put_chars;
      actual_iocb_ptr->iocb.modes = my_discard_modes;
      actual_iocb_ptr->iocb.control = my_discard_control;
      end;
else if mode=sequential_output_mode then do;
      blkptr->open_descrip.string = "sequential_output";
      blkptr->open_descrip.length = 17;
      actual_iocb_ptr->iocb.write_record = my_discard_write;
      end;
else if mode=keyed_sequential_output_mode then do;
      blkptr->open_descrip.string = "keyed_sequential_output";
      blkptr->open_descrip.length = 23;
      actual_iocb_ptr->iocb.write_record = my_discard_write;
      actual_iocb_ptr->iocb.seek_key = my_discard_seek_key;
      end;
else if mode=direct_output_mode then do;
      blkptr->open_descrip.string = "direct_output";
      blkptr->open_descrip.length = 13;
      actual_iocb_ptr->iocb.write_record = my_discard_write;
      actual_iocb_ptr->iocb.seek_key = my_discard_seek_key;
      end;
else do;
      call hcs_$reset_ips_mask(mask,mask);
      code = error_table_$bad_mode;
      return;
      end;

      if extend_bit then blkptr->open_descrip.string
          = blkptr->open_descrip.string||" -extend";
      actual_iocb_ptr->iocb.open_descrip_ptr = addr(open_descrip);
      actual_iocb_ptr->iocb.close = my_discard_close;
      call iox_$propagate(actual_iocb_ptr);
      call hcs_$reset_ips_mask(mask,mask);
      return;
```

```
my_discard_close: entry(iocb_ptr, code);
      code = 0;
      mask = "0"b;
      on any_other
         call handler;
      call hcs_$set_ips_mask ("0"b, mask);
      actual_iocb_ptr = iocb_ptr->iocb.actual_iocb_ptr;
      blkptr = actual_iocb_ptr->iocb.attach_data_ptr;
      actual_iocb_ptr->iocb.open_descrip_ptr = null();
      actual_iocb_ptr->iocb.detach_iocb = my_discard_detach;
      actual_iocb_ptr->iocb.open = my_discard_open;
      actual_iocb_ptr->iocb.control = iox_$err_not_open;
      actual_iocb_ptr->iocb.modes = iox_$err_not_open;
      call iox_$propagate(actual_iocb_ptr);
      call hcs_$reset_ips_mask(mask,mask);
      return;

my_discard_put_chars: entry(iocb_ptr,bufptr,buflen,code);
      code = 0;
      return;

my_discard_modes: entry(iocb_ptr,newmode,oldmode,code);
      code = 0;
      oldmode = "";
      return;

my_discard_write: entry(iocb_ptr,bufptr,buflen,code);
      code = 0;
      return;

my_discard_control: entry(iocb_ptr,order,infptr,code);
      if order = "io_call" then code = error_table_$no_operation;
      else code = 0;
      return;

my_discard_seek_key: entry(iocb_ptr,key,len,code);
      len = 0;
      code = error_table_$no_record;
      return;
```

```
          /* Internal procedure to handle faults while IPS interrupts
             are masked.  While not masked, any signals are simply
             passed on up the stack to their normal handlers.  For a
             fault while masked, the process is terminated (with the
             reason "unable to do critical I/O") because the I/O
             control blocks are in an inconsistent state, and we can
             tolerate neither spawning a command loop with interrupts
             masked nor a restart with a possibly changed mask. */

handler:   procedure;
dcl   continue_to_signal_ entry (fixed bin(35));
dcl   terminate_process_ entry (char(*), ptr);
dcl   error_table_$unable_to_do_io fixed(35) ext;
dcl   1 ti                                  aligned,
        2 version                           fixed bin init (0),
        2 code                              fixed bin (35);

      if mask ^= "0"b
      then
         do;
            ti.code = error_table_$unable_to_do_io; /* very bad trouble */
            call terminate_process_ ("fatal_error", addr (ti));
         end;
      call continue_to_signal_ (0);
   end handler;

end my_discard_attach;
```

EXAMPLE OF AN I/O MODULE

YOU ARE NOW READY FOR WORKSHOP #3

TOPIC VIII

Interprocess Communication

	Page
Overview	8-1
IPC Terminology	8-4
IPC Protocol	8-8
Sending Wakeups	8-10
IPC Subroutines	8-11
ipc_ Error Codes	8-13
Creating and Destroying Event Channels	8-15
Invoking an Event-Call Procedure	8-17
Going Blocked on an Event Channel	8-18
Reading an Event-Wait Channel	8-20
Control Functions	8-23
Masking or Assigning Priority to Event Channels	8-27
An Example Using Event-Wait Channels	8-29
An Example Using Event-Call Channels	8-35

OVERVIEW

- THE SUBSYSTEM DESIGNER IS OFTEN FACED WITH REQUIREMENTS FOR SOPHISTICATED INTERPROCESS AND INTRAPROCESS COMMUNICATIONS FACILITIES

- SUBROUTINES EXIST WHICH ALLOW THE DESIGNER TO HANDLE:

  - SYNCHRONIZATION OF SEVERAL COOPERATING PROCESSES

    - ipc FACILITATES INTERPROCESS COMMUNICATION VIA "STOP AND GO" SIGNALS BETWEEN PROCESSES

  - PROTECTION OF CONCURRENTLY ACCESSED DATA BASES

    - set_lock ALLOWS COOPERATING PROCESSES TO SHARE A CRITICAL DATA BASE IN A CONTROLLED MANNER

  - INTRAPROCESS TIMING

    - timer_manager ALLOWS A PROCESS TO MAKE USE OF SEVERAL CPU OR REAL-TIME TIMERS

OVERVIEW

- CONCEPT:
  - INTERPROCESS COMMUNICATION MEANS THAT TWO DISTINCT PROCESSES CAN PASS INFORMATION BACK AND FORTH BETWEEN THEM
    - A SIMPLE FORM OF INTERPROCESS COMMUNICATION MIGHT INVOLVE THE SHARING OF A COMMON SEGMENT BETWEEN TWO PROCESSES
    - THIS SIMPLE FORM OF COMMUNICATION DOES NOT ALLOW FOR <u>DIRECT</u> SYNCHRONIZATION, HOWEVER
  - Multics SUPPORTS A FULL INTERPROCESS COMMUNICATION FACILITY WHICH ALLOWS FOR <u>CONTROL</u> COMMUNICATION BETWEEN PROCESSES BY MEANS OF 'STOP' AND 'GO' SIGNALS

- THE INTERPROCESS COMMUNICATION MECHANISM
  - IS IMPLEMENTED BY THE ipc_ AND hcs_$wakeup SUBROUTINES
  - IS ACTUALLY MANAGED/COORDINATED BY THE Multics "TRAFFIC CONTROLLER", WHICH IS RESPONSIBLE FOR THE CREATION, DELETION, AND DISPATCHING OF PROCESSES
  - THE DISCUSSION WHICH FOLLOWS WILL NOT DWELL ON TRAFFIC CONTROLLER CONCEPTS (SEE F80B)
- EXAMPLE OF INTERPROCESS COMMUNICATION: THE 'send_message' AND 'accept_message' COMMANDS
  - THEY MAKE USE OF THE IPC FACILITY TO PASS TEXT MESSAGES FROM ONE PROCESS TO ANOTHER BY HAVING THEM PRINTED UPON THE RECEIPT OF A 'WAKEUP'

IPC TERMINOLOGY

- event
  - AN EVENT IS THE OCCURRENCE OF SOMETHING SIGNIFICANT
  - ONE PROCESS INFORMS ANOTHER THAT AN EVENT HAS OCCURRED BY CALLING hcs_$wakeup
  - A PROCESS WILL ONLY RECEIVE NOTIFICATION OF AN EVENT WHEN IT IS "BLOCKED"
  - DO NOT ASSUME THAT ONLY "SLEEPING" PROCESSES ARE SUBJECT TO "WAKING UP"

- event channel
  - IS THE ONE-WAY CONTROL PATH OVER WHICH NOTIFICATION OF THE OCCURRENCE OF EVENTS IS TRANSMITTED
  - IS MAINTAINED BY TRAFFIC CONTROLLER
  - IS CREATED ON BEHALF OF A USER IN A PARTICULAR RING

IPC TERMINOLOGY

- event-wait channel
  - AN EVENT CHANNEL WHICH, WHEN NOTIFIED OF AN EVENT, CAUSES THE PROCESS TO RESUME EXECUTION IF IT WAS "WAITING" (MORE TECHNICALLY, "BLOCKED"), OR AN EVENT CHANNEL WHICH MAY PERIODICALLY BE "POLLED" TO DETERMINE IF THE EVENT OCCURRED

- event-call channel
  - AN EVENT CHANNEL WHICH, WHEN NOTIFIED OF THE OCCURRENCE OF AN EVENT, CAUSES THE INVOCATION OF A SPECIFIED PROCEDURE IN THE PROCESS THAT CREATED THE CHANNEL

- blocked
  - A PROCESS CAN GO "BLOCKED" ON AN EVENT-WAIT CHANNEL, IN WHICH CASE IT WILL RESUME EXECUTION UPON EVENT NOTIFICATION
  - IS A PROCESS STATE IN WHICH THE PROCESS IS INACTIVE AND "LISTENING" TO AN EVENT-WAIT CHANNEL (OR CHANNELS)

- wakeup
  - A WAKEUP IS THE NOTIFICATION OF THE OCCURRENCE OF AN EVENT
  - IS SENT TO A SPECIFIC PROCESS ACROSS A SPECIFIED EVENT-WAIT OR EVENT-CALL CHANNEL

IPC TERMINOLOGY

- channel_id
  - IS THE fixed bin(71) VALUE WHICH IS USED TO UNIQUELY IDENTIFY A PARTICULAR EVENT CHANNEL
  - IS DERIVED FROM THE SYSTEM CLOCK
  - IS ASSOCIATED WITH ONE AND ONLY ONE PROCESS
- process_id
  - IS THE bit(36) VALUE WHICH UNIQUELY IDENTIFIES A PROCESS
  - IN ORDER TO SEND A WAKEUP, ONE SPECIFIES THE channel_id OF THE EVENT CHANNEL AND THE process_id OF THE PROCESS OWNING THE EVENT CHANNEL
- message
  - A 72-BIT VALUE OF ARBITRARY CONTENT CONTAINING INFORMATION WHOSE INTERPRETATION IS APPLICATION DEPENDENT

CHANNEL TYPE	OPTIONS
event-call	When a process establishes an event call channel, execution proceeds with the statement immediately following the call to create the event-call channel. The process does not (and should not) go blocked on the channel. If/when a wakeup is received, execution is momentarily interrupted and the procedure specified in the declaration is invoked. When this procedure completes, execution of the interrupted procedure continues.
event-wait	There are two options available: 1) The process can explicitly go blocked on the channel by calling ipc_$block. In this case, the statement following the call to ipc_$block will not be executed until/unless a wakeup is received on that channel. Notice that this effectively blocks the process itself. 2) The process may elect not to go blocked on this channel at all. Rather it will, from time to time, explicitly inquire as to whether or not a wakeup has been received. This polling technique is implemented via the ipc_$read_ev_chn entry point.

- SINCE IT IS NECESSARY TO KNOW THE channel_id AND process_id IN ORDER TO COMMUNICATE WITH ANOTHER PROCESS, SOME STANDARD PROTOCOL IS REQUIRED

- THE STEPS REQUIRED TO SET UP INTERPROCESS COMMUNICATION:

  - STEP 1

    - PROCESS_1 CREATES AN EVENT CHANNEL, BY WHICH ACT PROCESS_1 RECEIVES THE channel_id OF THE NEWLY CREATED EVENT CHANNEL

  - STEP 2

    - PROCESS_1 STORES THE channel_id AND ITS OWN process_id IN SOME KNOWN LOCATION IN A SHARED SEGMENT, THUS ALLOWING THESE VALUES TO BE ACCESSED BY OTHER COOPERATING PROCESSES

  - STEP 3

    - SOME OTHER PROCESS, SAY PROCESS_2, OBTAINS THE process_id AND channel_id VALUES FROM THE SHARED SEGMENT

- IT IS NOW POSSIBLE FOR PROCESS_2 TO SEND WAKEUPS TO PROCESS_1

IPC PROTOCOL

- COMMUNICATION ON AN EVENT CHANNEL IS ONE-WAY ONLY

- IF PROCESS_1 WISHES TO COMMUNICATE VIA ipc_ WITH PROCESS_2 IN THE ABOVE SCENARIO:

  - PROCESS_2 WOULD NEED ITS OWN EVENT-CHANNEL

  - IT WOULD BE NECESSARY FOR PROCESS_2 TO REPEAT THE STEPS OUTLINED ABOVE

- WHENEVER INTERPROCESS COMMUNICATION IS USED BETWEEN USER PROCESSES AND A SYSTEM PROCESS, A SUBROUTINE IS GENERALLY CALLED TO OBTAIN THE channel_id AND process_id OF THE SYSTEM PROCESS

SENDING WAKEUPS

- WAKEUPS ARE SENT FROM ONE PROCESS TO ANOTHER BY CALLING THE hcs_$wakeup ENTRY POINT

  - USAGE

    call hcs_$wakeup (process_id, channel_id, message, code);

    - process_id SPECIFIES TARGET PROCESS

    - channel_id IDENTIFIES CHANNEL THAT WAS SET UP BY TARGET PROCESS

    - message IS TWO WORDS HAVING SOME AGREED-UPON MEANING FOR THE COOPERATING PROCESSES

    - code IS A NON-STANDARD ERROR CODE (NOT IDEALLY SUITED TO com_err_)

      - = 1 IF SIGNALLING WAS CORRECTLY DONE, BUT THE TARGET PROCESS WAS IN THE STOPPED STATE (THE STATE A PROCESS IS IN JUST BEFORE THE FINAL STEP IN PROCESS TERMINATION)

      - = 2 IF AN INPUT ARGUMENT WAS INCORRECT, AND SIGNALLING WAS ABORTED

      - = 3 IF THE TARGET PROCESS WAS NOT FOUND, AND SIGNALLING WAS ABORTED

      - = error_table_$invalid_channel IF THE CHANNEL IDENTIFIER WAS NOT VALID

- EVENT CHANNELS ARE CREATED, DESTROYED, AND MANIPULATED VIA THE ipc_ SUBROUTINE

- ipc_ ENTRY POINTS MAY BE CLASSIFIED:

  - CREATING AND DESTROYING EVENT CHANNELS
    - ipc_$create_ev_chn
    - ipc_$delete_ev_chn
    - ipc_$decl_ev_call_chn
    - ipc_$decl_ev_wait_chn
  - GOING BLOCKED ON AN EVENT_WAIT CHANNEL
    - ipc_$block
  - READING AN EVENT-WAIT CHANNEL
    - ipc_$read_ev_chn

- CONTROL FUNCTIONS
  - ipc_$drain_chn
  - ipc_$cutoff
  - ipc_$reconnect
- MASKING OR ASSIGNING PRIORITY TO EVENT CHANNELS
  - ipc_$set_call_prior
  - ipc_$set_wait_prior
  - ipc_$mask_ev_calls
  - ipc_$unmask_ev_calls

IPC ERROR CODES

- ALL CALLS TO ipc_ RETURN A NONSTANDARD STATUS CODE

CODE	MEANING
0	No error.
1	A ring violation; for instance, the event channel resides in a ring that is not accessible from the caller's ring.
2	The table that contains the event channels for a given ring was not found.
3	The specified event channel was not found.
4	A logical error in using the ipc_ subroutine was encountered; for instance, waiting on an event-call channel.
5	A bad argument was passed to the ipc_ subroutine; for instance, a zero-value event channel_id.

- convert_ipc_code_

  - AN OBSCURE SUBROUTINE THAT CONVERTS A NONSTANDARD ipc_ CODE TO A SYSTEM STANDARD CODE

  - MAPPING

ipcode	returned code
0	0
1	error_table_$bad_ring_brackets
2-4	error_table_$no_message
5	error_table_$argerr

CREATING AND DESTROYING EVENT CHANNELS

- ipc CREATES EVENT-WAIT CHANNELS BY DEFAULT; HENCE, IN ORDER TO CREATE AN EVENT-CALL CHANNEL, IT IS FIRST NECESSARY TO CREATE AN EVENT-WAIT CHANNEL, AND THEN TO CHANGE IT INTO AN EVENT-CALL CHANNEL

- ENTRY POINTS TO CREATE AND DESTROY CHANNELS

  - ipc_$create_ev_chn

    - CREATES AN EVENT-WAIT CHANNEL IN THE CURRENT RING, RETURNING THE channel_id OF THE NEWLY CREATED CHANNEL, AND THE NONSTANDARD STATUS CODE

  - ipc_$delete_ev_chn

    - DESTROYS AN EVENT CHANNEL PREVIOUSLY CREATED BY THE PROCESS, REQUIRING THE channel_id OF THE CHANNEL TO BE DESTROYED AND RETURNING THE NONSTANDARD STATUS CODE

    - ONLY THE PROCESS CREATING AN EVENT CHANNEL (OR THE INITIALIZER PROCESS) MAY DESTROY IT

    - EVENT-CHANNELS ARE AUTOMATICALLY DESTROYED AT PROCESS TERMINATION TIME

CREATING AND DESTROYING EVENT CHANNELS

- ipc_$decl_ev_call_chn
  - CHANGES AN EVENT-WAIT CHANNEL INTO AN EVENT-CALL CHANNEL
  - call ipc_$decl_ev_call_chn (channel_id, proc_entry, data_ptr, priority, code);
    - channel_id IDENTIFIES EVENT-WAIT CHANNEL TO BE CHANGED
    - proc_entry IS ENTRY VALUE OF "HANDLER" TO BE INVOKED UPON RECEIPT OF WAKEUP
    - data_ptr POINTS TO A USER-DEFINED REGION CONTAINING DATA FOR proc_entry TO INTERPRET
    - priority INDICATES WHICH (OF POTENTIALLY MANY) SIMULTANEOUS EVENT-CALL CHANNEL EVENTS IN THIS RING WILL BE HONORED FIRST (THE LOWEST NUMBER IS HONORED FIRST)

- ipc_$decl_ev_wait_chn
  - USED TO CHANGE AN EVENT-CALL CHANNEL INTO AN EVENT-WAIT CHANNEL
  - IT REQUIRES THE channel_id OF AN EVENT-CALL CHANNEL AND IT RETURNS THE NONSTANDARD STATUS CODE
  - NOTE: SINCE EVENT-WAIT CHANNELS ARE CREATED BY DEFAULT, THIS IS USED ONLY TO CHANGE AN EVENT-CALL CHANNEL BACK INTO AN EVENT-WAIT CHANNEL

INVOKING AN EVENT-CALL PROCEDURE

- WHEN A PROCESS IS AWAKENED ON AN EVENT-CALL CHANNEL, CONTROL IS IMMEDIATELY PASSED TO THE "HANDLER" PROCEDURE PREVIOUSLY SPECIFIED IN THE CALL TO ipc_$decl_ev_call_chn

- "HANDLER" WILL BE CALLED WITH ONE ARGUMENT, A POINTER TO A STRUCTURE CONTAINING INFORMATION ABOUT THE WAKEUP

  - THE HANDLER SHOULD DECLARE THE FOLLOWING STRUCTURE BASED UPON THE event_call_info_ptr PARAMETER

```
dcl 1 event_call_info based (event_call_info_ptr),
    2 channel_id        fixed bin(71),
    2 message           fixed bin(71),
    2 sender            bit(36),
    2 origin,
      3 dev_signal      bit(18) unal,
      3 ring            bit(18) unal,
    2 data_ptr          ptr;
```

  - RECALL: THE VALUE OF data_ptr WAS SPECIFIED IN THE CALL TO ipc_$decl_ev_call_chn AND POINTS TO STRUCTURE OF THE USER'S CHOOSING

  - SEE >ldd>include>event_call_info.incl.pl1

GOING BLOCKED ON AN EVENT CHANNEL

- ipc_$block

  - CAUSES THE PROCESS TO GO BLOCKED ON THE SPECIFIED EVENT-WAIT CHANNEL(S)

  - IN-LINE PROGRAM EXECUTION WILL NOT PROCEED UNTIL/UNLESS A WAKEUP IS RECEIVED ON ONE OF THE CHANNELS

  - USAGE

```
call ipc_$block (wait_list_ptr, event_wait_info_ptr, code);
                      ↑                  ↑                ↑
                    INPUT              INPUT            OUTPUT
```

  - A POINTER TO THE BASE OF A USER-ALLOCATED "WAIT-LIST" STRUCTURE IS REQUIRED

    - THIS WAIT-LIST STRUCTURE CONTAINS THE channel_idS OF THE EVENT-WAIT CHANNELS TO GO BLOCKED ON (TO LISTEN TO)

```
dcl 1 wait_list              based aligned,
    2 nchan                  fixed bin,
    2 pad                    bit(36),
    2 channel_id (n refer wait_list.nchan) fixed bin(71);
```

GOING BLOCKED ON AN EVENT CHANNEL

- ipc_$block ALSO REQUIRES A POINTER TO THE BASE OF A STRUCTURE INTO WHICH IT CAN PUT INFORMATION ABOUT THE EVENT THAT FREED THE PROCESS FROM ITS BLOCKED STATE

```
dcl 1 event_wait_info aligned based(event_wait_info_ptr),
    2 channel_id     fixed bin(71),
    2 message        fixed bin(71),
    2 sender         bit(36),
    2 origin,
      3 dev_signal   bit(18) unaligned,
      3 ring         bit(18) unaligned,
    2 channel_index  fixed bin;     /* INDEX INTO
                                       wait_list.channel_id */
```

- POSSIBLE USE OF event_wait_info.sender:

  PASS sender TO get_userid_ SUBROUTINE (WHICH USES THE RESTRICTED SEGMENT >sc1>answer_table) AND IT WILL RETURN Person_id AND Project_id

- SEE >ldd>include>event_wait_info.incl.pl1

READING AN EVENT-WAIT CHANNEL

- THE PROCESS CREATING AN EVENT-WAIT CHANNEL MAY SIMPLY INQUIRE AS TO WHETHER OR NOT AN EVENT HAS OCCURRED ON A SPECIFIED EVENT-WAIT CHANNEL - THIS IS REFERRED TO AS "READING" THE EVENT-WAIT CHANNEL

- READING THE EVENT "RESETS" IT

- ipc_$read_ev_chn

  - READS THE INFORMATION ABOUT AN EVENT ON A SPECIFIED EVENT-WAIT CHANNEL IF THE EVENT HAS OCCURRED

  - REQUIRES AS INPUT THE channel_id OF THE EVENT-WAIT CHANNEL TO BE READ

  - RETURNS A VALUE INDICATING WHETHER OR NOT AN EVENT OCCURRED, AND IF AN EVENT HAS OCCURRED, RETURNS INFORMATION ABOUT THAT EVENT IN event_wait_info STRUCTURE

  - ALSO RETURNS THE NONSTANDARD STATUS CODE

```
! ll 70
! pr DEMO_READ.pl1 1

DEMO_READ: proc;

dcl
     ipc_$create_ev_chn entry (fixed bin (71), fixed bin (35)),
     ipc_$delete_ev_chn entry (fixed bin(71), fixed bin(35)),
     ipc_$read_ev_chn entry (fixed bin (71), fixed bin, ptr,
                                          fixed bin(35)),
     hcs_$wakeup entry (bit (36), fixed bin (71), fixed bin (71),
                                          fixed bin (35)),
     addr builtin,
     ioa_ entry options (variable),
     com_err_ entry options (variable),
     get_process_id_ entry returns (bit (36));

dcl
     channel_id fixed bin (71),
     message fixed bin (71),
     code fixed bin (35),
     ev_occurred fixed bin;

%include event_info;  /* My own private include file */

/* This short example illustrates the fact that reading an event-wait
   channel has the effect of 'resetting' it.      */

          call ipc_$create_ev_chn (channel_id, code);
          if code ^= 0 then call trouble;
          message = 1;
          call hcs_$wakeup (get_process_id_ (), channel_id, message,
                                                          code);
          if code ^= 0 then call trouble;
          else call ioa_ ("WAKEUP successfully completed.");
          call ipc_$read_ev_chn (channel_id, ev_occurred,
                                          addr (event_info), code);
          if code ^= 0 then call trouble;
          call ioa_ ("read_ev_chn says the event ^[has not^;has^] occu
\crred, and the message is ^i.", ev_occurred+1, event_info.message);
          call ipc_$read_ev_chn (channel_id, ev_occurred,
                                          addr (event_info), code);
          if code ^= 0 then call trouble;
          call ioa_ ("A second call of read_ev_chn says: The event ^[h
\cas not^;has^] occurred.", ev_occurred+1);

trouble:  proc;
               call com_err_ (code, "DEMO_READ", "Something unexpected
\c occurred.");
               goto bottom;
          end trouble;
```

```
bottom:
          call ipc_$delete_ev_chn (channel_id, code);
          /* Ignore bad code, if it should occur.          */
     end DEMO_READ;


! DEMO_READ
WAKEUP successfully completed.
read_ev_chn says the event has occurred, and the message is 1.
A second call of read_ev_chn says: The event has not occurred.
```

- RESETTING CHANNELS AND INHIBITING THE NOTIFICATION OF EVENTS

  - ipc_$drain

    - RESETS AN EVENT-WAIT CHANNEL SO THAT ANY PENDING EVENTS (EVENTS THAT HAVE BEEN RECEIVED AND QUEUED UP BUT NOT PROCESSED FOR THAT CHANNEL) ARE REMOVED

    - REQUIRES THE channel_id OF THE EVENT-WAIT CHANNEL

    - RETURNS THE NONSTANDARD STATUS CODE

    - OFTEN USED IN 'cleanup' HANDLERS

  - ipc_$cutoff

    - INHIBITS THE "READING" (IN THE GENERAL SENSE) OF PENDING OR FUTURE EVENTS ON A SINGLE SPECIFIED EVENT (WAIT OR CALL) CHANNEL

    - NOTE THAT MORE (NEW) EVENTS CAN BE RECEIVED (AND QUEUED UP), BUT THEY WILL NOT CAUSE THE PROCESS TO WAKE UP

    - REQUIRES THE channel_id OF THE EVENT CHANNEL TO CUTOFF, AND RETURNS THE NONSTANDARD STATUS CODE

    - AN ATTEMPT TO READ A CUTOFF CHANNEL RESULTS IN CODE 4 (A LOGICAL ERROR IN USING ipc_ WAS ENCOUNTERED)

- ipc_$reconnect

  - ENABLES THE READING OF EVENTS ON A SINGLE SPECIFIED EVENT CHANNEL FOR WHICH READING HAD PREVIOUSLY BEEN INHIBITED BY A CALL TO ipc_$cutoff

  - WHEN CALLED, ALL PENDING SIGNALS, WHETHER RECEIVED BEFORE OR DURING THE TIME READING WAS INHIBITED, ARE HENCEFORTH AVAILABLE FOR READING (IN THE GENERAL SENSE)

  - REQUIRES THE channel_id OF THE EVENT CHANNEL WHICH HAD BEEN CUTOFF, AND RETURNS THE NONSTANDARD STATUS CODE

```
!print DEMO_CUTOFF.pl1 1
DEMO_CUTOFF: proc;

/* This experiment demonstrates the cutting off and reconnection of
   a single channel.  It accomplishes this in the following steps:

   1    get wait channel
   2    issue wakeup with msg = 1
   3    cutoff channel
   4    issue wakeup with message = 2
   5    reconnect
   6    read channel twice
   7    delete channel

   ioa_ is called at strategic points to confirm ipc's behavior. */

dcl
     channel_id fixed bin (71),
     code fixed bin (35),
     i fixed bin,
     ev_occurred fixed bin,
     addr builtin,
     get_process_id_ entry returns (bit (36)),
     message fixed bin (71),
     hcs_$wakeup entry (bit (36), fixed bin (71), fixed bin (71),
                        fixed bin (35)),
     ipc_$create_ev_chn entry (fixed bin (71), fixed bin (35)),
     ipc_$delete_ev_chn entry (fixed bin (71), fixed bin (35)),
     ipc_$cutoff entry (fixed bin (71), fixed bin (35)),
     ipc_$reconnect entry (fixed bin (71), fixed bin (35)),
     ipc_$read_ev_chn entry (fixed bin (71), fixed bin, ptr,
                        fixed bin (35)),
    (com_err_, ioa_) entry options (variable),
     1 event_info aligned,
     2 channel_id fixed bin (71),
     2 message fixed bin (71),
     2 sender bit (36),
     2 origin,
     3 dev_signal bit (18) unaligned,
     3 ring bit (18) unaligned,
     2 channel_index fixed bin;
         call ipc_$create_ev_chn (channel_id, code);
         if code ^= 0 then call trouble;
         call hcs_$wakeup (get_process_id_ (), channel_id, 1, code);
         if code ^= 0 then call trouble;
         call ioa_ ("First wakeup successfully performed with msg = 1");
         call ipc_$cutoff (channel_id, code);
         if code ^= 0 then call trouble;
         call ioa_ ("Channel successfully cutoff.");
         call hcs_$wakeup (get_process_id_ (), channel_id, 2, code);
         if code ^= 0 then call trouble;
```

```
          call ioa_ ("2nd wakeup performed while channel was cutoff.");
          call ipc_$reconnect (channel_id, code);
          if code ^= 0 then call trouble;
          do i = 1 to 2;
               call ipc_$read_ev_chn (channel_id, ev_occurred,
                                      addr (event_info), code);
               if code ^= 0 then call trouble;
               call ioa_ ("^[First^;Second^] reading channel after reco
\cnnect.  The event ^[hasn't^;has^] occurred. ^[^s^;^/The message is ^
\ci.^]", i, ev_occurred+1, ev_occurred+1, event_info.message);
          end;
wrapup:
          call ipc_$delete_ev_chn (channel_id, code);
          return;

trouble:  proc;
               call com_err_ (code, "DEMO_CUTOFF", "Error not expected.");
               goto wrapup;
          end trouble;

     end DEMO_CUTOFF;


!DEMO_CUTOFF
First wakeup successfully performed with msg = 1
Channel successfully cutoff.
2nd wakeup performed while channel was cutoff.
First reading channel after reconnect.  The event has occurred.
The message is 1.
Second reading channel after reconnect.  The event has occurred.
The message is 2.
```

MASKING OR ASSIGNING PRIORITY TO EVENT CHANNELS

- SINCE THERE IS ACTUALLY SOME DELAY BETWEEN THE TIME A WAKEUP IS RECEIVED AND THE TIME THE PROCESS IS NOTIFIED, IT IS POSSIBLE FOR SEVERAL WAKEUPS TO BE PRESENT BY THE TIME A PROCESS IS AWAKENED
  - IT IS POSSIBLE TO SPECIFY RELATIVE PRIORITIES AMONG EVENT-CALL CHANNELS
  - POSSIBLE TO SPECIFY THAT EVENT-CALL CHANNELS HAVE PRIORITY OVER EVENT-WAIT CHANNELS AND VICE VERSA
  - BY DEFAULT, EVENT-CALL CHANNELS HAVE PRIORITY OVER EVENT-WAIT CHANNELS

- MANIPULATING PRIORITIES
  - ipc_$set_wait_prior
    - CAUSES EVENT-WAIT CHANNELS TO BE GIVEN PRIORITY OVER EVENT-CALL CHANNELS (THIS IS NOT THE DEFAULT)
    - ONLY EVENT CHANNELS IN CURRENT RING ARE AFFECTED

MASKING OR ASSIGNING PRIORITY TO EVENT CHANNELS

- ipc_$set_call_prior
  - CAUSES EVENT-CALL CHANNELS TO BE GIVEN PRIORITY OVER EVENT-WAIT CHANNELS (THIS IS THE DEFAULT)
  - ONLY EVENT CHANNELS IN CURRENT RING ARE AFFECTED

- ipc_$mask_ev_calls
  - CAUSES THE ipc_$block ENTRY POINT TO COMPLETELY IGNORE ALL EVENT-CALL CHANNELS IN THE CALLER'S RING (I.E., TO MASK THEM) SO THAT ANY WAKEUPS SENT ACROSS EVENT-CALL CHANNELS ARE INSTEAD QUEUED UP
  - CAUSES A "MASK COUNTER" TO BE INCREMENTED; MASKING IS IN EFFECT SO LONG AS COUNTER $\neq$ 0

- ipc_$unmask_ev_calls
  - DECREMENTS THE "MASK COUNTER"; EVENT-CALLS ARE UNMASKED (NOTICED) WHEN COUNTER = 0
  - SEVERAL EXTERNAL PROCEDURES MAY NEED TO MASK AND UNMASK. AN INCREMENTAL COUNTER PERMITS INDISCRIMINATE CALLS WITHOUT FEAR OF PREMATURE UNMASKING

AN EXAMPLE USING EVENT-WAIT CHANNELS

```
!print abs_print_punch.pl1 1

abs_print_punch: proc;

dcl  ME char (15) init ("abs_print_punch") static options (constant);
dcl  hcs_$initiate entry (char (*), char (*), char (*),
     fixed bin (1), fixed bin (2), ptr, fixed bin (35)),
     get_process_id_ entry returns (bit (36)),
     ipc_$create_ev_chn entry (fixed bin (71), fixed bin (35)),
     ipc_$block entry (ptr, ptr, fixed bin (35)),
    (ioa_, com_err_) entry options (variable),
     expand_pathname_ entry (char (*), char (*), char (*),
     fixed bin (35)),
     dprint_ entry (char (*), char (*), ptr, fixed bin (35));

dcl  code fixed bin (35);
dcl 1 ipc_info based (seg_ptr),
    2 target_process_id bit (36) unal,
    2 target_chnl_id fixed bin (71);

%include dprint_arg; /* USED BY dprint_ SUBROUTINE */

dcl 1 wait_list based (wait_list_ptr),
    2 nchan fixed bin,
    2 channel_id (0 refer (nchan)) fixed bin (71);

dcl 1 event_info,
    2 channel_id fixed bin (71),
    2 message fixed bin (71),
    2 sender bit (36),
    2 origin,
      3 dev_signal bit (18) unal,
      3 ring bit (18) unal,
    2 channel_index fixed bin;

dcl (seg_ptr, wait_list_ptr) ptr;

dcl  dprint_paths$ char (168) external static;
dcl  dir char (168), entry char (32);

/* SUPPLY ipc_ PROTOCOL INFORMATION */
          call hcs_$initiate (">udd>F15dw>Auerbach", "ipc_seg1", "",
               0, 1, seg_ptr, code);
          if seg_ptr = null () then call ERROR;
          call ipc_$create_ev_chn (
               seg_ptr -> ipc_info.target_chnl_id, code);
          if code ^= 0 then call ERROR;
          seg_ptr -> ipc_info.target_process_id = get_process_id_ ();

/* NOW GO BLOCKED WAITING FOR FURTHER INSTRUCTIONS */
          allocate wait_list;
```

```
          wait_list_ptr -> wait_list.nchan = 1;
          wait_list_ptr -> wait_list.channel_id (1) =
               seg_ptr -> ipc_info.target_chnl_id;

          call ipc_$block (wait_list_ptr, addr (event_info), code);
          if code ^= 0 then call ERROR;

/* AN EVENT HAS OCCURRED - EXAMINE */
          call ioa_ ("CHANNEL ID ^i^/MESSAGE ^i^/SENDER ^.3b",
               event_info.channel_id,
               event_info.message,
               event_info.sender);
          call ioa_ ("DEV SIGNAL ^.3b^/RING ^.3b^/CHANNEL INDEX ^i",
               event_info.origin.dev_signal,
               event_info.origin.ring,
               event_info.channel_index);

          if event_info.message = 0 then goto CANCEL;
          if event_info.message = 1 then goto PRINT;
          if event_info.message = 2 then goto PUNCH;
/* INVALID MESSAGE */
          call com_err_ (0, ME, "INVALID REQUEST CODE ^i",
               event_info.message);
          return;

CANCEL:
/* NO PRINTING OR PUNCHING AT ALL */
          call ioa_ ("PRINT/PUNCH REQUEST CANCELLED");
          return;

PRINT:
/* PRINT REQUEST */
          dprint_arg.pt_pch = 1;
          dprint_arg.output_module = 1;
          dprint_arg.class = "printer";
          goto DPRINT;

PUNCH:
/* PUNCH REQUEST */
          dprint_arg.pt_pch = 2;
          dprint_arg.output_module = 3;
          dprint_arg.class = "punch";
```

```
DPRINT:
          call expand_pathname_ (dprint_paths$, dir, entry, code);
          if code ^= 0 then call ERROR;
          dprint_arg.version = 4;
          dprint_arg.copies = 1;
          dprint_arg.delete = 0;
          dprint_arg.queue = 3;
          dprint_arg.notify = 1;
          dprint_arg.heading = "";
          dprint_arg.dest = "";
          dprint_arg.nep = "0"b;
          dprint_arg.single = "0"b;
          dprint_arg.non_edited = "0"b;
          dprint_arg.truncate = "0"b;
          dprint_arg.center_top_label = "0"b;
          dprint_arg.center_bottom_label = "0"b;
          dprint_arg.lmargin = 10;
          dprint_arg.line_lth = -1;
          dprint_arg.page_lth = -1;
          dprint_arg.top_label = "";
          dprint_arg.bottom_label = "";
          call dprint_ (rtrim (dir), rtrim (entry),
               addr (dprint_arg), code);
          if code ^= 0 then call ERROR;

/* REPORT ACTION AND QUIT */
          call ioa_ ("^a>^a ^[DPRINT^;DPUNCH^] REQUEST SUBMITTED.",
               dir, entry, dprint_arg.pt_pch);
          call ioa_ ("END ^a", ME);

ERROR:    proc;
               call com_err_ (code, ME);
               goto FINIS;
          end;

FINIS:
     end abs_print_punch;
```

AN EXAMPLE USING EVENT-WAIT CHANNELS

```
r 11:12 0.122 15

!print driver_ipc.pl1 1

driver_ipc: proc;

dcl  ME char (10) init ("driver_ipc") static options (constant);
dcl  hcs_$initiate entry (char (*), char (*), char (*),
     fixed bin (1), fixed bin (2), ptr, fixed bin (35));
dcl  code fixed bin (35);
dcl  seg_ptr ptr;
dcl 1 ipc_info based (seg_ptr),
    2 his_process_id bit (36),
    2 his_chnl_id fixed bin (71);
dcl  hcs_$wakeup entry (bit (36), fixed bin (71), fixed bin (71),
     fixed bin (35));
dcl  dprint_paths$ char (168) external static;
dcl  message fixed bin (71);
dcl (ioa_, com_err_) entry options (variable);


/* PICK UP ABSENTEE'S PROCESS AND CHANNEL IDS */
          call hcs_$initiate (">udd>F15dw>Auerbach", "ipc_seg1", "",
               0, 1, seg_ptr, code);
          if seg_ptr = null () then call ERROR;

/* TELL ABSENTEE TO PRINT AND GIVE IT A PATH TO PRINT */
          dprint_paths$ = ">udd>F15dw>Auerbach>abs_print_punch.pl1";
          message = 1;

/* FIRE OFF WAKEUP SIGNAL */
          call hcs_$wakeup (ipc_info.his_process_id,
               ipc_info.his_chnl_id,
               message,
               code);
          if code ^= 0 then call ERROR;

          call ioa_ ("^[DPRINT^;DPUNCH^] REQUEST FOR ^a",
               message, dprint_paths$);
          call ioa_ ("END ^a", ME);

ERROR:    proc;
               call com_err_ (code, ME);
               goto FINIS;
          end;

FINIS:
     end driver_ipc;


r 11:12 0.020 2
```

AN EXAMPLE USING EVENT-WAIT CHANNELS

```
!print ipc_example.absin 1

abs_print_punch
logout
&quit


r 11:12 0.020 1

!ear ipc_example
27 already requested.
r 10:55 0.113 6

!driver_ipc
DPRINT REQUEST FOR >udd>F15dw>Auerbach>abs_print_punch.pl1
END driver_ipc
r 11:13 0.068 4

!ldr -long

Queue 3:    1 request.  12 total requests.

Pathname:           >udd>F15dw>Auerbach>abs_print_punch.pl1
Type:               print
Copies:             1
Time:               03/13/80  10:56 mst Thur
Delete:             no
Notify:             yes
Options:            -indent 10

r 11:13 0.138 10
```

AN EXAMPLE USING EVENT-WAIT CHANNELS

```
!print ipc_example.absout 1

Absentee user Auerbach F15dw logged in: 03/13/80  11:14 mst Thur
r 11:14 1.827 34

abs_print_punch
CHANNEL ID 98439864967772869469
MESSAGE 1
SENDER 004670305344
DEV SIGNAL 000000
RING 000004
CHANNEL INDEX 1
>udd>F15dw>Auerbach>abs_print_punch.pl1 DPRINT REQUEST SUBMITTED.
END abs_print_punch
r 11:15 2.314 27

logout

Absentee user Auerbach F15dw logged out 03/13/80  11:16 mst Thur
CPU usage 2 sec, memory usage 1.0 units

r 11:17 0.084 14

From IO.SysDaemon (printer):
printed >udd>F15dw>Auerbach>abs_print_punch.pl1
$0.06 queue 3 prtd 10115
```

AN EXAMPLE USING EVENT-CALL CHANNELS

```
!print listen.pl1 1

listen: proc;
dcl  ipc_$create_ev_chn entry (fixed (71), fixed (35)),
     get_process_id_ entry returns (bit (36)),
     hcs_$wakeup entry (bit (36), fixed bin (71), fixed (71), fixed (35)),
     iox_$control entry (ptr, char (*), ptr, fixed (35)),
     code fixed (35),
     ipc_$decl_ev_call_chn entry (fixed (71), entry, ptr, fixed,
                                  fixed (35)),
     hcs_$initiate_count entry (char (*), char (*), char (*), fixed (24),
     fixed (2), ptr, fixed (35)),
     bc fixed (24),
     iox_$user_output ext ptr,
     iox_$put_chars entry (ptr, ptr, fixed (21), fixed (35)),
    (ioa_, com_err_) entry options (variable);
dcl 1 ipc_info based (seg_ptr),
    2 process_id bit (36),
    2 channel_id fixed bin (71);
dcl  seg_ptr ptr;
dcl  ME char (6) init ("listen") static options (constant);
dcl  event_info_ptr ptr;
dcl 1 event_info based (event_info_ptr),
    2 channel_id fixed bin (71),
    2 message fixed bin (71),
    2 sender bit (36),
    2 origin,
      3 dev_signal bit (18) unal,
      3 ring bit (18) unal,
    2 data_ptr ptr;

/* INITIATE PROTOCOL PASSING SEGMENT */
          call hcs_$initiate_count (">udd>F15dw>Auerbach", "ipc_seg1",
               "", bc, 1, seg_ptr, code);
          if seg_ptr = null () then call ERROR;

/* CREATE EVENT CHANNEL AND MAKE IT A CALL CHANNEL */
          call ipc_$create_ev_chn (ipc_info.channel_id, code);
          if code ^= 0 then call ERROR;
          ipc_info.process_id = get_process_id_ ();
          call ipc_$decl_ev_call_chn (ipc_info.channel_id,
               print_msg, null (), 0, code); /* data_ptr = null */
          if code ^= 0 then call ERROR;

/* ALL DONE */
          call ioa_ ("Now listening for messages.");
          return;
```

```
print_msg: entry (event_info_ptr);
/* THIS HANDLER FOR TAKING MESSAGES DOES NOT USE THE
event_info_ptr PARAMETER PASSED TO IT.                */

/* GET POINTER TO MAILBOX SEGMENT */
          call hcs_$initiate_count (">udd>F15dw>Auerbach", "mailbox",
               "", bc, 1, seg_ptr, code);
          if seg_ptr = null () then call ERROR;

/* PRINT OUT CONTENTS OF >udd>F15dw>Auerbach>mailbox        */
          call ioa_ ("Message is:");
          call iox_$put_chars (iox_$user_output, seg_ptr,
               divide (bc, 9, 21, 0), code);
          if code ^= 0 then call ERROR;

/* RESTART ANY INTERRUPTED OUTPUT AND RETURN */
          call iox_$control (iox_$user_output, "start", null (),
               code);

ERROR:    proc;
               call com_err_ (code, ME);
               call iox_$control (iox_$user_output, "start", null (),
                    code);
               goto FINIS;
          end;

FINIS:
     end listen;


r 12:36 0.383 18

!print put_message.pl1 1

put_message: proc;

%include listen_decls;
dcl  mailbox file;
dcl 1 ipc_info based (seg_ptr),
    2 process_id bit (36),
    2 channel_id fixed bin (71);
dcl  seg_ptr ptr;

/* THE FOLLOWING PL/1 I/O STATEMENTS BUILD UP A SEGMENT
WHICH WILL BE DUMPED LATER BY THE print_msg EVENT HANDLER. */

          open file (mailbox) stream output;
          put file (mailbox) skip
               list ("Hello...this is the absentee process..");
          put file (mailbox) skip
               list ("Just wanted to prove it works!!");
```

```
          put file (mailbox) skip;
          close file (mailbox);

/* OBTAIN process_id AND channel_id SO THAT WE CAN SEND WAKEUP */
          call hcs_$initiate_count (">udd>F15dw>Auerbach", "ipc_seg1",
               "", bc, 1, seg_ptr, code);
          if seg_ptr = null () then call ERROR;

/* NOW SEND THE WAKEUP */
          call hcs_$wakeup (ipc_info.process_id, ipc_info.channel_id,
               0, code);
          if code ^= 0 then call ERROR;

ERROR:    proc;
dcl  ME char (11) init ("put_message") static options (constant);
               call com_err_ (code, ME);
               goto FINISH;
          end;

FINISH:
     end put_message;


r 12:36 0.020 1

!print pm.absin 1

put_message
logout
&quit


r 12:36 0.029 1

!defer_messages
r 12:36 0.018 5

!listen
Now listening for messages.
r 12:36 0.015 3

!ear pm
27 already requested.
r 12:36 0.126 12
```

```
!who

Multics MR8.0, load 55.0/130.0; 55 users
Absentee users 0/3

IO.SysDaemon
Backup.SysDaemon
IO.SysDaemon
IO.SysDaemon
IO.SysDaemon
GCOS.SysDaemon
Volume_Dumper.Daemon
Opr.Operator
MFreeman.SSF
Susan.NCB
Jagernauth.Multics
Irish.Doc
Downing.Multics
Wardd.Multics
Falksenj.Multics
Coppola.HFED
Casselman.HCRC
Martinson.SysMaint
Nolde.Bus-Plan
FED.VIS
Lombreglia.NCB
Lutz.GSASched
Retriever.SysDaemon
Harrison.Rapidata
Stryk.HCRC
Matheson.DEBUG
Landrum.SED
Baryza.FORD_CONV
Auerbach.F15dw
Johnson.SysAdmin
Donner.Multics
Fawcett.VIS
Message is:

Hello...this is the absentee process..
Just wanted to prove it works!!
Coflin.G66
Friedman.F15aw
Glicksman.HISCAN
Berglund.Multics
Arnwine.SiteSA
NThompson.NOPS
Chouinard.BBbench
JWilliams.SED
Student_20.F01
RBarnes.Multics
```

```
Bergum.HCRC
Gildersleeve.Multics
Tilton.MMPP
Watts.Doc
FED.VIS
Sam.SRB
Student 14.F01
Gowans.BSask
PHJones.BBbench
Troost.MMPP
Whitford.Doc
Gintell.Multics
Grimes.SMP
```

YOU ARE NOW READY FOR WORKSHOP #4

TOPIC IX

Interprocess Data Base Sharing

	Page
Introduction	9-1
The Locking Mechanism	9-2
The set_lock_ Subroutine	9-4
An Example of Locking	9-6

- CONCEPT
  - ANOTHER FORM OF INTERPROCESS COMMUNICATION INVOLVES THE SHARING OF COMMON DATA BASE (NOT TO BE CONFUSED WITH MDBM) SEGMENTS
  - BECAUSE SEVERAL PROCESSES MAY BE ATTEMPTING TO CONCURRENTLY ACCESS AND UPDATE A SHARED DATA BASE, SOME FORM OF CONTROL IS REQUIRED:
    - TO PREVENT THE DATA BASE FROM BEING LEFT IN AN INCONSISTENT STATE
    - TO PREVENT ANY PROCESS FROM OPERATING UPON PARTIALLY-UPDATED DATA

- A POSSIBLE SOLUTION
  - A "LOCKING" FACILITY: THE set_lock_ SUBROUTINE
  - COOPERATING PROCESSES OBSERVE A LOCKING PROTOCOL, IN WHICH A GIVEN PROCESS DOES NOT ACCESS A DATA BASE UNTIL IT CAN SET A LOCK WORD, AND IN WHICH A PROCESS RESETS THAT LOCK WORD WHEN IT HAS COMPLETED A CRITICAL UPDATE OR RETRIEVAL

THE LOCKING MECHANISM

- DATA BASE LOCKING IS IMPLEMENTED BY set_lock_ SUBROUTINE (AG93) (BUT SEE ALSO THE PL/1 NON-STANDARD BUILTINS stac AND stacq)

- set_lock_ PROTOCOL

  - A CALLER-SUPPLIED LOCK WORD IS USED FOR THE MUTUAL EXCLUSION OF PROCESSES

  - THIS LOCK WORD IS

    - DECLARED (bit(36) aligned) BY UPDATING PROGRAM(S)

    - ZEROED ONCE BY SOME SPECIAL INITIALIZATION PROGRAM WHEN DATABASE FIRST CREATED (THUS INDICATING IT'S UNLOCKED)

    - LOCATED

      - USUALLY IN A SPECIAL, SEPARATE SEGMENT ACCESSIBLE TO ALL COOPERATING PROCESSES

      - SOMETIMES IN THE DATABASE ITSELF

- WHEN A PROCEDURE IS ABOUT TO ENTER A CRITICAL SECTION OF CODE, IT CALLS AN ENTRY POINT IN set_lock_ WHICH ATTEMPTS TO SET THE LOCK BY PLACING THAT PROCESS'S LOCK IDENTIFIER IN THE LOCK WORD USING AN INDIVISIBLE MACHINE INSTRUCTION, stac (STORE-A CONDITIONAL)

  - stac USES A SPECIAL MAIN MEMORY REFERENCE THAT PROHIBITS SUCH REFERENCES BY OTHER PROCESSES BETWEEN THE TEST AND THE DATA TRANSFER.

  - IF THE LOCK WORD ALREADY CONTAINS SOME OTHER VALID LOCK IDENTIFIER, THE INTERPRETATION IS THAT THE DATA BASE IS LOCKED BY THAT OTHER PROCESS, AND THE CALLING PROCESS WAITS FOR THE LOCK TO BE UNLOCKED BY THAT OTHER PROCESS

- WHEN A CRITICAL SECTION OF CODE HAS BEEN COMPLETED BY THE PROGRAM, THE LOCK OUGHT TO BE UNLOCKED, ALLOWING ANOTHER PROCESS TO SET THE LOCK

- SUCCESS HINGES ON THE FOLLOWING CONVENTIONS:

  - set_lock_ IS THE ONLY PROCEDURE THAT MAY MODIFY THE LOCK WORD (WITH THE EXCEPTION OF THE PROGRAM WHICH INITIALIZES THE DATA BASE AND LOCK WORD)

  - ALL PROCESSES SHOULD CALL set_lock_$lock BEFORE ENTERING A CRITICAL SECTION OF CODE

  - ALL PROCESSES SHOULD CALL set_lock_$unlock AFTER COMPLETING A CRITICAL SECTION OF CODE

- set_lock_$lock (AG93)

  - ATTEMPTS TO PLACE LOCK IDENTIFIER OF CALLING PROCESS IN THE GIVEN LOCK WORD

  - call set_lock_$lock (lock_word, wait_time, code);

    - wait_time INDICATES THE NUMBER OF SECONDS THAT set_lock_$lock SHOULD WAIT FOR A VALIDLY LOCKED LOCK WORD TO BE UNLOCKED BEFORE RETURNING UNSUCCESSFULLY (-1 INDICATES NO TIME LIMIT)

    - ONE OF THE FOLLOWING CODES IS RETURNED:

      - 0

      - error_table_$invalid_lock_reset

      - error_table_$locked_by_this_process

      - error_table_$lock_wait_time_exceeded

- set_lock_$unlock (AG93)

  - ATTEMPTS TO RESET A GIVEN LOCK WORD TO "0"b

  - call set_lock_$unlock (lock_word, code);

  - RETURNS ONE OF THE FOLLOWING CODES:

    - 0

    - error_table_$lock_not_locked

    - error_table_$locked_by_other_process IF lock_word CONTAINED NON-ZERO VALUE NOT EQUAL TO LOCK IDENTIFIER OF THE CALLING PROCESS

```
!pwd
 >user_dir_dir>F15dw>Auerbach
 r 92:9 0.173 1

!ls -pn >udd>F15dw>Auerbach>AJAX_db

 Segments = 2, Lengths = 2.

 r w     0  lock_word
 re      2  book_seat

 Multisegment-files = 1, Lengths = 7.

 r w     7  flight_records

 r 09:29 0.064 0

!print book_seat.pl1 1

 book_seat: procedure;

 /* THIS PROGRAM UPDATES AN AIRLINES DATABASE
    WHICH IS LOCATED IN THE SAME DIRECTORY
    AS THIS PROGRAM, AND WHICH ALSO CONTAINS
     A LOCK SEGMENT */

 dcl  flight_records file;
 dcl 1 flight_rec based (rec_ptr),
     2 total_seats fixed bin,
     2 seats_booked fixed bin,
     2 seat_info (0 refer (seats_booked)),
       3 name char (20) varying,
       3 address char (30) varying;
 dcl  flight_no char (4) varying,
      date char (6) varying;
 dcl  rec_ptr ptr;

 dcl  set_lock_$lock entry (bit (36) aligned,
      fixed bin, fixed bin (35)),
      set_lock_$unlock entry (bit (36) aligned,
      fixed bin (35)),
     (ioa_, com_err_) entry options (variable),
      change_wdir_ entry (char (168) aligned, fixed bin (35));

 dcl  lock_word$ bit (36) aligned external static;
 dcl (key, cleanup) condition;

 dcl (code,
      error_table_$invalid_lock_reset external,
      error_table_$locked_by_this_process external,
      error_table_$lock_wait_time_exceeded external) fixed bin (35);

 dcl  ME char (9) init ("book_seat") static options (constant);
```

```
/* ESTABLISH 'on unit' FOR 'cleanup'
   DO NOT UNLOCK LOCK - DATA BASE MAY BE
   IN AN INCONSISTENT STATE */
        on cleanup close file (flight_records);

/* ESTABLISH 'on unit' FOR 'key' CONDITION
   REPORT ERROR AND ASK AGAIN */
        on key (flight_records) begin;
             call ioa_ ("Invalid key entered ^a", onkey ());
             goto PROMPT;
        end;


/* BEGIN UPDATE PROGRAM */
        call ioa_ ("AJAX Airlines flight booking program begins");

/* CHANGE WORKING DIR TO AIRLINES DATABASE DIRECTORY */
        call change_wdir_ (">udd>F15dw>Auerbach>AJAX_db", code);
        if code ^= 0 then do;
             call com_err_ (code, ME);
             return;
        end;

/* OPEN DATABASE */
        open file (flight_records) direct update;

/* LOCK DATABASE NOW - TRY FOR 30 SECONDS */
        call set_lock_$lock (lock_word$, 30, code);

        if code ^= 0 then
/* COULDN'T LOCK IT - FIND OUT WHY */

             if code = error_table_$lock_wait_time_exceeded
             then do;
/* DATABASE IS BUSY */
                  call ioa_ ("Database busy - try again later.");
                  goto WRAPUP;
             end;
             else
             if code = error_table_$invalid_lock_reset
             then do;
/* SOMEBODY DIDN'T UNLOCK BEFORE DYING */
                  call ioa_ ("Database has invalid lock");
                  call ioa_ ("Notify DBA - no update allowed");
                  goto WRAPUP;
             end;
             else
             if code = error_table_$locked_by_this_process then do;
```

```
/* SOMETHING IS VERY WRONG - DIE */
                    call ioa_ ("FATAL ERROR!!");
                    call ioa_ ("NOTIFY DBA IMMEDIATELY!!");
                    goto WRAPUP;
               end;
               else;
          else;

/* DATABASE IS NOW LOCKED */

PROMPT:
/* BASIC REQUEST LOOP */

          do while ("1"b);
               call ioa_ ("Enter flight_no, date for booking");
               get list (flight_no, date);
               if flight_no = "0" then goto WRAPUP;
/* STOP WHEN flight_no IS "0" */
/* TRY TO READ RECORD */
               read file (flight_records) key (flight_no||date)
                    set (rec_ptr);

/* SEE IF ANY SEATS ARE LEFT */
               if rec_ptr -> seats_booked >= rec_ptr -> total_seats
               then do;
                    call ioa_ ("Flight is booked full.");
               end;

/* OKAY - GET REST OF INFO */
               else do;
                    seats_booked = seats_booked + 1;
                    call ioa_ ("Enter name, address of cust");
                    get list (
                         flight_rec.seat_info (seats_booked).name,
                         flight_rec.seat_info (seats_booked).address);
                    rewrite file (flight_records)
                         from (rec_ptr -> flight_rec);
               end;
          end;

WRAPUP:
/* UNLOCK AND CLOSE DATABASE */
          call set_lock_$unlock (lock_word$, code);
          close file (flight_records);
          call ioa_ ("End Update Program ^a", ME);

     end book_seat;
```

AN EXAMPLE OF LOCKING

```
r 09:29 0.025 2

!AJAX_db>book_seat
AJAX Airlines flight booking program begins
Enter flight_no, date for booking
!112,800303
Flight is booked full.
Enter flight_no, date for booking
!0,0
End Update Program book_seat
```

TOPIC X

Intraprocess Timer Management

	Page
Introduction	10-1
Timer Management Terminology	10-2
timer_manager_ Generic Arguments	10-3
timer_manager_ Entry Points	10-5
Blocking a Process	10-7
Using Call Timers	10-8
Using Wakeup Timers	10-10
Resetting and Inhibiting Timers	10-12
Standard System Handlers	10-15
Two Examples Using Timers	10-16
timer_manager_ Summary	10-22

- CERTAIN SOPHISTICATED PROGRAMS MAY REQUIRE THE USE OF ONE OR MORE CPU AND/OR REAL-TIME TIMERS

- TIMER MANAGER FACILITY (TMF) OPERATES PRIMARILY IN CONJUNCTION WITH THE INTERPROCESS COMMUNICATION FACILITY, THUS ENABLING PROGRAMS TO RUN ASYNCHRONOUSLY WITHIN A PROCESS

TIMER_MANAGER

- ~~Multics~~ ALLOWS A PROCESS TO:

  - BLOCK ITSELF FOR A SPECIFIED REAL TIME PERIOD (SLEEP)

  - CALL A SPECIFIED PROCEDURE WHEN A SPECIFIED TIME INTERVAL HAS ELAPSED

  - ISSUE A WAKEUP ON A SPECIFIED EVENT-WAIT CHANNEL WHEN A SPECIFIED TIME INTERVAL HAS ELAPSED

- THE SUBROUTINE INTERFACE IS timer_manager_ (AK93)

- FOR MORE ABOUT TIME, SEE THE MPM SUBROUTINES MANUAL:

  clock_ cpu_time_and_paging_ date_time_
  decode_clock_value_ virtual_cpu_time_

TIMER MANAGEMENT TERMINOLOGY

- timer_manager_ MAKES USE OF SEVERAL CRITICAL CONCEPTS:

  - alarm

    - "alarm" IS USED TO DESIGNATE A REAL-TIME TIMER; THAT IS, A "WALL-TIME" ELAPSED TIME - WHEN AN "alarm" TIMER GOES OFF, THE "alrm" (STATIC) CONDITION IS SIGNALLED

  - cpu

    - "cpu" IS USED TO DESIGNATE A VIRTUAL CPU TIMER; WHEN A "cpu" TIMER GOES OFF, THE "cput" (STATIC) CONDITION IS SIGNALLED

  - relative time

    - A TIME MEASURED FROM THE CALL TO timer_manager_; THAT IS, A TIME MEASURED FROM THE TIME THE TIMER IS CREATED

  - absolute time

    - A TIME MEASURED FROM THE FIXED POINT IN TIME "January 1, 1901 0000 Hours"

TIMER MANAGER GENERIC ARGUMENTS

- timer_manager_ ENTRY POINTS ACCEPT A COMMON SET OF GENERIC ARGUMENTS

  - channel

    - THE EVENT CHANNEL ID (fixed bin(71)) OVER WHICH A WAKEUP IS TO BE TRANSMITTED

    - SET UP PRIOR TO INVOCATION OF A timer_manager_ ENTRY POINT

  - routine

    - THE PROCEDURE TO BE INVOKED WHEN A "CALL" TIMER GOES OFF (SPECIFIED WHEN THE TIMER IS CREATED)

    - THE PROCEDURE WILL BE PASSED TWO ARGUMENTS (UNLIKE AN EVENT-CALL PROCEDURE) AS FOLLOWS:

      - mc_ptr

        - AN ALIGNED POINTER TO THE "MACHINE CONDITIONS" AT THE TIME THE alrm OR cput CONDITION WAS SIGNALLED (SEE SECTION 7 IN MPM REFERENCE GUIDE)

      - name

        - A CHARACTER STRING INDICATING WHETHER THE TIMER WAS AN ALARM TIMER (alrm) OR A CPU TIMER (cput)

    - IS MOST OFTEN AN EXTERNAL ENTRY, BUT MIGHT BE INTERNAL (TAKE CARE!)

TIMER MANAGER GENERIC ARGUMENTS

- time
  - MANY timer_manager_ ENTRY POINTS REQUIRE THAT THE TIME (fixed bin (71)) BE SPECIFIED; alrm OR cput CONDITION IS SIGNALLED AT THAT TIME

- flags
  - MANY timer_manager_ ENTRY POINTS REQUIRE THIS bit(2) STRING, WHICH SPECIFIES HOW THE time ARGUMENT IS TO BE INTERPRETED
    - "11"b MEANS RELATIVE SECONDS
    - "10"b MEANS RELATIVE MICROSECONDS (1e-6 SECONDS)
    - "01"b MEANS ABSOLUTE SECONDS
    - "00"b MEANS ABSOLUTE MICROSECONDS (1e-6 SECONDS)

TIMER MANAGER ENTRY POINTS

- timer_manager_ ENTRY POINTS ALLOW A PROCESS TO:
  - BLOCK A PROCESS FOR A SPECIFIED REAL TIME INTERVAL
    - timer_manager_$sleep
  - CAUSE A SPECIFIED PROCEDURE TO BE INVOKED AT A SPECIFIED TIME
    - timer_manager_$alarm_call
    - timer_manager_$cpu_call
  - CAUSE A WAKEUP TO BE ISSUED ON A SPECIFIED EVENT-WAIT CHANNEL AT A SPECIFIED TIME
    - timer_manager_$alarm_wakeup
    - timer_manager_$cpu_wakeup

TIMER MANAGER ENTRY POINTS

- RESET AND INHIBIT TIMERS
  - timer_manager_$alarm_call_inhibit
  - timer_manager_$reset_alarm_call
  - timer_manager_$reset_alarm_wakeup
  - timer_manager_$cpu_call_inhibit
  - timer_manager_$reset_cpu_call
  - timer_manager_$reset_cpu_wakeup

TIMER MANAGER ENTRY POINTS

BLOCKING A PROCESS

- timer_manager_$sleep CAUSES PROCESS TO GO BLOCKED FOR A PERIOD OF REAL TIME

  - OTHER TIMERS THAT ARE ACTIVE ARE PROCESSED WHENEVER THEY GO OFF

  - HOWEVER, THE PROCEDURE ISSUING THIS CALL WILL NOT RESUME (I.E., EXECUTE ITS NEXT INSTRUCTION) UNTIL THE REAL TIME HAS BEEN PASSED

  - EXAMPLE

```
                                        ----SECONDS
                                        |
                                        v
call timer_manager_$sleep ( 30, "11"b);
                                        ^
                                        |
                          RELATIVE------
```

  - WOULD CAUSE THIS PROCESS TO GO TO "SLEEP" FOR THIRTY SECONDS

TIMER MANAGER ENTRY POINTS

USING CALL TIMERS

- ENTRY POINTS WHICH CAUSE A SPECIFIED PROCEDURE TO BE INVOKED WHEN A TIMER GOES OFF

  - timer_manager_$alarm_call

    - SETS UP A REAL-TIMER
    - A SPECIFIED ROUTINE IS CALLED WHEN THE TIMER GOES OFF
    - IT REQUIRES THE time, flags, AND routine ARGUMENTS AS INPUT
    - EXAMPLE

      ```
      call timer_manager_$alarm_call ( 80, "11"b, print_usage);
      ```

      - WOULD CAUSE A PROCEDURE CALLED print_usage TO BE INVOKED AFTER 80 SECONDS OF REAL TIME HAD ELAPSED

- timer_manager_$cpu_call
  - SETS UP A CPU TIMER WHICH WILL CAUSE A SPECIFIED PROCEDURE TO BE INVOKED WHEN A SPECIFIED INTERVAL OF CPU TIME HAS ELAPSED
  - REQUIRES THE SAME ARGUMENTS AS THE timer_manager_$alarm_call ENTRY POINT
  - EXAMPLE

```
                                        ----MICROSECONDS
                                        |
                                        v
call timer_manager_$cpu_call ( 1000, "10"b, print_cpu_usage);
                                        ^
                                        |
                                RELATIVE----
```

  - CAUSES THE PROGRAM print_cpu_usage TO BE INVOKED WHEN ONE MILLISECOND OF CPU TIME HAS ELAPSED

TIMER MANAGER ENTRY POINTS

USING WAKEUP TIMERS

- ENTRY POINTS BELOW ALLOW THE CALLER TO SPECIFY THAT A WAKEUP IS TO BE SENT ACROSS SPECIFIED EVENT-WAIT CHANNELS WHEN THE TIMER GOES OFF

  - timer_manager_$alarm_wakeup

    - SETS UP A REAL-TIME TIMER THAT ISSUES A WAKEUP ON THE EVENT-WAIT CHANNEL SPECIFIED WHEN THE TIMER GOES OFF

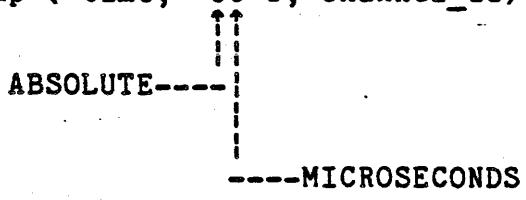
    - CALLER MAY WISH TO GO BLOCKED ON THE CHANNEL (BUT NEEDN'T)

    - THE EVENT MESSAGE PASSED IS THE STRING "alarm___"

    - REQUIRES THREE INPUT ARGUMENTS - THE time, flags, AND channel_id

    - EXAMPLE

```
call convert_date_to_binary_ ("1 hour 5 minutes", time, code);
call timer_manager_$alarm_wakeup ( time, "00"b, channel_id);
```



    - WOULD CAUSE A WAKEUP TO BE ISSUED ACROSS channel_id 65 WALL_CLOCK MINUTES FROM THE TIME convert_date_to_binary_ WAS CALLED

USING WAKEUP TIMERS

- timer_manager_$cpu_wakeup

  - OPERATES EXACTLY LIKE timer_manager_$alarm_wakeup EXCEPT THAT THE TIMER IS A CPU TIMER AND THE EVENT MESSAGE IS "cpu_time"

  - EXAMPLE

```
call timer_manager_$cpu_wakeup (50, "01"b, channel_id);
                                      ↑↑
                                      ||
                             ABSOLUTE----|
                                       |
                                       |
                                       ----SECONDS
```

    - WOULD CAUSE A WAKEUP TO BE ISSUED ON THE EVENT-WAIT CHANNEL IDENTIFIED BY channel_id AFTER A TOTAL OF 50 CPU SECONDS HAD BEEN EXPENDED (MEASURED FROM PROCESS CREATION TIME)

TIMER MANAGER ENTRY POINTS

RESETTING AND INHIBITING TIMERS

- timer_manager_ ENTRY POINTS EXIST WHICH ALLOW A PROCESS TO RESET OR INHIBIT TIMERS

- USERS OF timer_manager_ SHOULD BE AWARE OF THE PERILS OF ASYNCHRONOUS PROCESSING, AND PROGRAMS CREATING CALL OR WAKEUP TIMERS SHOULD PROVIDE AN 'on unit' FOR THE cleanup CONDITION TO RESET TIMERS

  - SUCH 'on units' GENERALLY DO NOTHING MORE THAN RESET THE TIMERS, THUS PREVENTING SUCH TIMERS FROM GOING OFF AT UNDESIRED TIMES

  - ENTRY POINTS WHICH RESET AND INHIBIT TIMERS:

    - timer_manager_$reset_alarm_call

      - RESETS, OR TURNS OFF ALL REAL-TIME TIMERS (alarm) THAT CALL THE ROUTINE SPECIFIED WHEN THEY GO OFF

      - REQUIRES ONLY ONE ARGUMENT, THE NAME OF THE ROUTINE FOR WHICH TIMERS HAVE BEEN SET

      - EXAMPLE

```
call timer_manager_$reset_alarm_call (print_usage);
```

        - THIS CALL WOULD TURN OFF ANY REAL-TIME TIMERS WHICH WOULD CALL print_usage IF THEY WENT OFF

TIMER MANAGER ENTRY POINTS

RESETTING AND INHIBITING TIMERS

- timer_manager_$reset_cpu_call
  - OPERATES IN THE SAME MANNER AS timer_manager_$reset_alarm_call EXCEPT THAT IT TURNS OFF CPU TIMERS FOR THE SPECIFIED PROCEDURE

- timer_manager_$reset_alarm_wakeup
  - TURNS OFF ALL REAL-TIME TIMERS THAT ISSUE A WAKEUP ON THE EVENT-WAIT CHANNEL SPECIFIED
  - THE ONLY INPUT ARGUMENT IS THE channel_id OF THE CHANNEL FOR WHICH TIMER WAKEUPS ARE TO BE TURNED OFF
  - EXAMPLE

```
call timer_manager_$reset_alarm_wakeup (channel_id);
```

    - THIS CALL WOULD TURN OFF ANY REAL-TIME TIMERS WHICH WOULD OTHERWISE ISSUE A WAKEUP ON THE CHANNEL IDENTIFIED BY channel_id WHEN THEY WENT OFF

- timer_manager_$reset_cpu_wakeup
  - OPERATES EXACTLY LIKE timer_manager_$reset_alarm_wakeup EXCEPT THAT IT TURNS OFF CPU TIMERS FOR THE EVENT-WAIT CHANNEL SPECIFIED

RESETTING AND INHIBITING TIMERS

- INHIBITING INTERRUPTS WHILE USING CALL TIMERS ALLOWS THE PROCESS TO ENSURE THAT THE HANDLER (THE PROCEDURE INVOKED) WILL NOT BE INTERRUPTED BEFORE IT RETURNS
  - IF SUCH HANDLERS DO NOT RETURN, THE PROCESS MAY MALFUNCTION, SINCE IT IS DANGEROUS TO INHIBIT INTERRUPTS FOR TOO LONG
  - timer_manager_$alarm_call_inhibit
    - OPERATES EXACTLY LIKE timer_manager_$alarm_call EXCEPT THAT ALL INTERRUPTS ARE INHIBITED JUST BEFORE THE HANDLER IS INVOKED
    - WHEN THE HANDLER RETURNS, ALL INTERRUPTS ARE REENABLED
  - timer_manager_$cpu_call_inhibit
    - OPERATES LIKE timer_manager_$cpu_call EXCEPT THAT ALL INTERRUPTS ARE INHIBITED WHILE THE HANDLER IS EXECUTING

TIMER MANAGER ENTRY POINTS

STANDARD SYSTEM HANDLERS

- OTHER ENTRY POINTS IN timer_manager_ SERVE AS STATIC HANDLERS FOR TWO CONDITIONS:
  - timer_manager_$cpu_time_interrupt (FOR 'cput' CONDITION)
  - timer_manager_$alarm_interrupt (FOR 'alrm' CONDITION)
- SEE THE CODE FOR user_real_init_admin_ IN APPENDIX B

```
!print cookie.pl1 1

cookie: proc;
/* THE INFAMOUS COOKIE MONSTER PROGRAM
   THIS PROGRAM USES THE TIMER MANAGER FACILITY
   TO CAUSE THE COOKIE MONSTER PROGRAM TO BE
   REINVOKED AGAIN AND AGAIN, THUS BEWILDERING
   THE CALLER */

dcl  timer_manager_$alarm_call entry(fixed bin (71), bit (2), entry),
     timer_manager_$reset_alarm_call entry (entry),
     iox_$control entry (ptr, char (*), ptr, fixed bin (35)),
    (ioa_, ioa_$nnl) entry options (variable);

dcl (quit, cleanup) condition;
dcl  next_time fixed bin (71);
dcl  null builtin;
dcl  iox_$user_io ext ptr;
dcl  answer char (6);
dcl  sysin file;
dcl  code fixed bin (35);
dcl  i static init (1);
dcl  first_time bit (1) static init ("1"b);


/* ESTABLISH 'on unit' FOR CLEANUP
   RESET ANY TIMERS THAT MAY BE AROUND */
      on cleanup call timer_manager_$reset_alarm_call (handler);

/* COME HERE IF FIRST TIME EXECUTING COOKIE */
        if first_time then do;
/* FOR INITIAL CALL, SET TIME TO 10 SECONDS */
             next_time = 10;
             first_time = "0"b;
        end;
/* WHEN COOKIE IS CALLED AGAIN,
   SET THE TIME TO 80 SECONDS */
        else next_time = 80;

/* NOW SET UP TIMER TO CALL handler AND RETURN
   IT WILL SEEM TO THE CALLER THAT ALL IS NORMAL */
      call timer_manager_$alarm_call (next_time, "11"b, handler);
        return;
```

```
/* COME HERE WHEN TIMER GOES OFF */
handler:  entry;
/* ESTABLISH 'on unit' TO TRAP HIS QUITS */
          on quit begin;
               call ioa_ ("QUIT");
               call ioa_ ("You wanna get yourself logged out??");
               i = i+1;
               go to ask_again;
          end;

/* BASIC PROMPTING LOOP
   WE READ INPUT LOOKING FOR COOKIES */
ask_again:
          if i = 1 then call ioa_ ("I want a cookie.");
          else if i = 2 then call ioa_ ("Please give me a cookie.");
          else if i = 3 then call ioa_ (
               "You had better give me a cookie.");
          else if i = 4 then call ioa_ ("I WANT A COOKIE!");
          else if i = 5 then call ioa_ (
               "PLEASE--GIMME A COOKIE!!!!");
          else do;
               call ioa_ ("I give up. You're hopeless.");
               call ioa_$nnl ("Guess I'll have to get one myself.");
               call ioa_ ("COOKIECOOKIECOOKIE");

/* AT THIS POINT I EITHER GOT A COOKIE FROM HIM
   OR FROM MYSELF - RESET i AND CALL COOKIE AGAIN */
next_set:
               i = 1;
               call cookie;
               call iox_$control (iox_$user_io, "start", null, code);
               return;
          end;

/* HERE'S WHERE I GRAB HIS INPUT LINES */
          get list (answer);
          if answer = "cookie" | answer = "COOKIE" then do;
               call ioa_ ("Thanks. I needed that. YumYumYum...");
               go to next_set;
          end;
          else do;
               i = i+1;
               go to ask_again;
          end;
     end;
```

```
!cookie
r 11:56 0.065 1

!hmu

Multics MR8.0, load 63.0/100.0; 63 users
Absentee users 0/5

r 11:56 0.058 0

I want a cookie.
!who
Please give me a cookie.
!(QUIT)

QUIT
You wanna get yourself logged out??
You had better give me a cookie.
!new_proc
PLEASE--GIMME A COOKIE!!!!
!logout
I give up. You're hopeless.
Guess I'll have to get one myself.COOKIECOOKIECOOKIE
!new_proc
r 11:57 1.829 68
```

```
!print usage.pl1 1

usage: proc (mc_ptr, name);

/* THIS PROGRAM PRINTS OUT PAGE FAULT AND PAGING DEVICE
   FAULT INFORMATION EVERY HALF-SECOND OF CPU TIME */

dcl (mc_ptr ptr,
     name char (*)) parameter;

dcl  timer_manager_$cpu_call entry (fixed bin (71), bit (2), entry),
     timer_manager_$reset_cpu_call entry (entry),
     cpu_time_and_paging_ entry (fixed bin, fixed bin (71),
     fixed bin),
     iox_$control entry (ptr, char (*), ptr, fixed bin (35)),
     ioa_ entry options (variable);

dcl  cleanup condition,
     code fixed bin (35),
     iox_$user_io external ptr;

dcl (total_pf fixed bin,
     total_pdf fixed bin,
     first_time bit (1) init ("1"b)) static;

dcl  pf fixed bin,
     time fixed bin (71),
     pdf fixed bin;


/* ESTABLISH cleanup HANDLER TO RESET TIMER */
     on cleanup call timer_manager_$reset_cpu_call (usage);

/* PERFORM INITIALIZATION IF FIRST TIME */
          if first_time then do;
               call cpu_time_and_paging_ (total_pf, time, total_pdf);
               call ioa_ ("Usage counters initialized");
               call ioa_ ("Total page faults since process began ^i",
                    total_pf);
               call ioa_ ("Total pd faults since process began ^i",
                    total_pdf);
               first_time = "0"b;
          end;

/* COMES HERE WHEN TIMER GOES OFF */
          else do;
               call cpu_time_and_paging_ (pf, time, pdf);
               call ioa_ ("In the half-second CPU interval:");
               call ioa_ ("^i page faults were taken",
                    pf - total_pf);
               call ioa_ ("^i paging device faults were taken",
                    pdf - total_pdf);
               total_pf = pf;
```

```
          total_pdf = pdf;
          call iox_$control (iox_$user_io, "start",
               null (), code);
     end;

/* FIRE UP NEXT TIMER IN EITHER CASE */
     call timer_manager_$cpu_call (500000, "10"b, usage);

     end usage;
!usage
Usage counters initialized
Total page faults since process began 703
Total pd faults since process began 0
r 12:19 0.083 4

!hmu

Multics MR8.0, load 63.0/100.0; 63 users
Absentee users 1/5

r 12:19 0.055 7

!list -first 2

Segments = 62, Lengths = 127.

re      1  usage
r w     1  usage.pl1

r 12:19 0.201 21

!cookie
r 12:19 0.052 1

!pwd
>udd>F15dw>Auerbach
r 12:19 0.035 0

I want a cookie.
!cookie
Thanks. I needed that. YumYumYum...
!whome
In the half-second CPU interval:
55 page faults were taken
0 paging device faults were taken
Auerbach.F15dw

r 12:19 0.310 30
```

```
!pl1 cookie
PL/I
In the half-second CPU interval:
60 page faults were taken
0 paging device faults were taken
In the half-second CPU interval:
41 page faults were taken
0 paging device faults were taken
In the half-second CPU interval:
63 page faults were taken
0 paging device faults were taken
r 12:20 1.876 163
```

TIMER MANAGER SUMMARY

FOR REAL-TIME TIMERS

sleep	GO BLOCKED AT THIS POINT FOR SPECIFIED TIME
alarm_call	CALL SPECIFIED ROUTINE AT SPECIFIED TIME
alarm_call_inhibit	CALL SPECIFIED ROUTINE AT SPECIFIED TIME WITH ALL INTERPROCESS SIGNALS MASKED OFF
alarm_wakeup	ISSUE A WAKEUP ON SPECIFIED EVENT-WAIT CHANNEL WHEN TIMER GOES OFF, PASSING A MESSAGE OF "alarm___"
reset_alarm_call	TURN OFF ALL TIMERS THAT CALL ROUTINE SPECIFIED WHEN MATURE
reset_alarm_wakeup	TURN OFF ALL TIMERS THAT WAKEUP SPECIFIED EVENT-WAIT CHANNEL WHEN THEY MATURE

FOR CPU-TIME TIMERS

cpu_call	CALL SPECIFIED ROUTINE AT SPECIFIED TIME
cpu_call_inhibit	CALL SPECIFIED ROUTINE AT SPECIFIED TIME WITH ALL INTERPROCESS SIGNALS MASKED OFF
cpu_wakeup	ISSUE A WAKEUP ON SPECIFIED EVENT-WAIT CHANNEL WHEN TIMER GOES OFF, PASSING A MESSAGE OF "cpu_time"
reset_cpu_call	TURN OFF ALL TIMERS THAT CALL ROUTINE SPECIFIED WHEN THEY MATURE
reset_cpu_wakeup	TURN OFF ALL TIMERS THAT WAKEUP SPECIFIED EVENT-WAIT CHANNEL WHEN THEY MATURE

YOU ARE NOW READY FOR WORKSHOP #5

TOPIC XI

The Stack and Argument Lists

	Page
The Process Stack Segment	11-1
The Stack Header	11-2
The Stack Frame	11-4
Argument List Format	11-7
Argument Descriptors	11-10

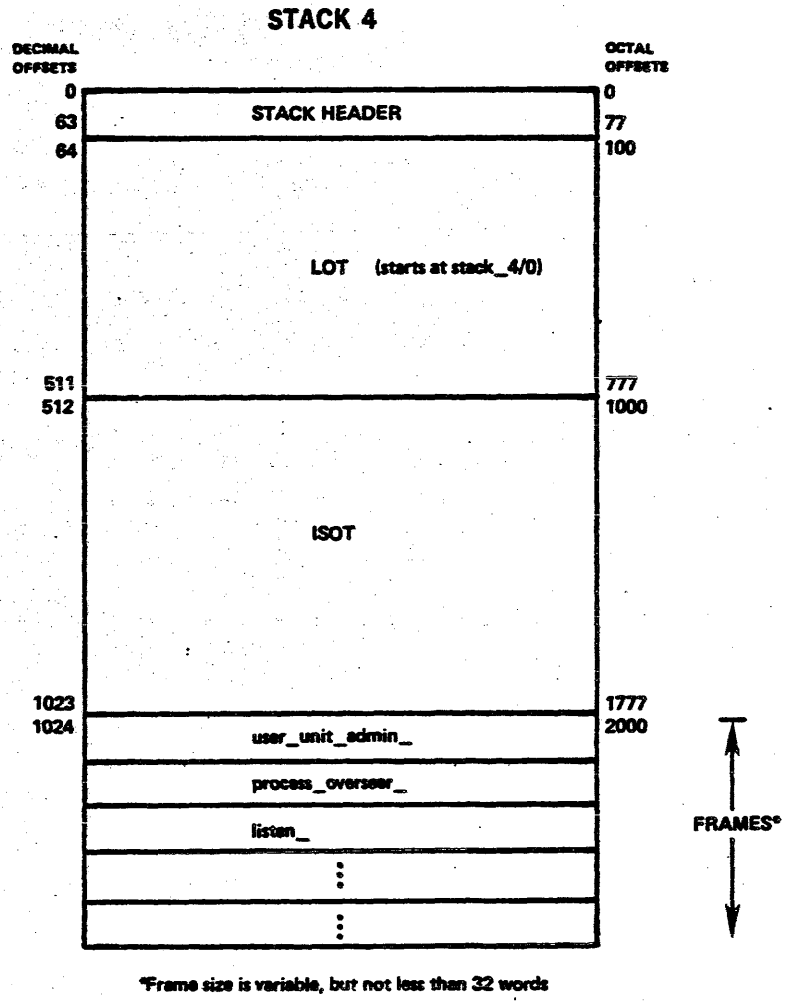
THE PROCESS STACK SEGMENT

- EXECUTION IN A STANDARD Multics PROCESS USES A STACK SEGMENT

  - THERE IS ONE STACK SEGMENT PER RING WITH ENTRYNAME 'stack_n' WHERE 'n' IS THE RING OF EXECUTION

  - EACH STACK CONTAINS A STACK "HEADER" FOLLOWED BY AS MANY STACK "FRAMES" AS ARE REQUIRED BY THE PROCESS

  - THE STACK DYNAMICALLY EXPANDS AND SHRINKS AS INDIVIDUAL FRAMES ARE "PUSHED" ONTO THE STACK (BY AN INVOKED PROCEDURE) AND "POPPED" OFF THE STACK (BY A RETURNING PROCEDURE)



THE STACK HEADER

- MOSTLY CONTAINS POINTERS TO SPECIAL CODE (STANDARD CALL, PUSH, AND RETURN OPERATORS) AND SPECIAL DATA BASES (LINKAGE OFFSET TABLE, REFERENCE NAME TABLE, ETC) WHICH ARE REQUIRED FOR THE NORMAL EXECUTION OF A PROCESS IN THE CORRESPONDING RING

```
dcl 1 stack_header       based (sb) aligned,

    2 pad1 (4)           fixed bin,
    2 old_lot_ptr        ptr,  /* OBSOLETE */
    2 combined_stat_ptr  ptr,  /* POINTS TO AREA CONTAINING
                                  SEPARATE STATIC */

    2 clr_ptr            ptr,  /* AREA_PTR FOR LINKAGE SECTION ALLOCATION */
    2 max_lot_size       fixed bin(17) unal,
    2 main_proc_invoked  fixed bin(11) unal,
    2 run_unit_depth     fixed bin(5) unal,
    2 cur_lot_size       fixed bin(17) unal,          /* IN WORDS */

    2 system_free_ptr    ptr,  /* USUALLY POINTS TO SYSTEM FREE STORAGE

    2 user_free_ptr      ptr, /* USUALLY POINTS TO
                                 <unique>.area.linker|0 */

    2 null_ptr           ptr,  /* THERE IS NO STACK FRAME
                                  PREVIOUS TO THE HEADER */
    2 stack_begin_ptr    ptr,
    2 stack_end_ptr      ptr,  /* POINTS TO NEXT USEABLE
                                  STACK FRAME */
    2 lot_ptr            ptr,  /* INITIALLY POINTS TO BASE OF STACK */

    2 signal_ptr         ptr,  /* POINTS TO SIGNALLING PROC FOR
                                    THIS RING */
    2 bar_mode_sp        ptr,  /* NEEDED BECAUSE BAR MODE PROGS CAN
                                  CHANGE THE STACK FRAME PTR REGISTER
                                  (PR6) */
    2 pl1_operators_ptr  ptr,  /* POINTS TO pl1_operators_
                                  $operator_table */
    2 call_op_ptr        ptr,  /* POINTS TO ALM CALL OPERATOR */

    2 push_op_ptr        ptr,  /* POINTS TO ALM PUSH OPERATOR */
    2 return_op_ptr      ptr,  /* POINTS TO STANDARD RETURN ALM OPERATOR */
    2 return_no_pop_op_ptr ptr,  /* POINTS TO SHORT RETURN ALM OPERATOR */
```

```
2 entry_op_ptr       ptr,  /* POINTS TO ALM ENTRY OPERATOR */
2 trans_op_tv_ptr    ptr,  /* POINTS TO A VECTOR OF SPECIAL
                              LANGUAGE OPERATORS */
2 isot_ptr           ptr,
2 sct_ptr            ptr,  /* POINTS TO SYSTEM CONDITION
                              TABLE (SCT) */
2 unwinder_ptr       ptr,  /* POINTS TO UNWINDER PROCEDURE
                              FOR THIS RING */

2 sys_link_info_ptr ptr,   /* POINTS TO *system LINK NAME
                              TABLE */
2 rnt_ptr            ptr,  /* POINTS TO REFERENCE NAME TABLE */
2 ect_ptr            ptr,  /* OBSOLETE */
2 assign_linkage_ptr ptr,  /* OBSOLETE */
2 pad3 (8)           bit (36);  /* FOR FUTURE EXPANSION */
```

- NOTE:

  set_system_storage SETS stack_header.system_free_ptr

  set_user_storage SETS stack_header.user_free_ptr

  WHEN THE NUMBER OF INITIATED SEGMENTS EXCEEDS 512, THE lot AND isot ARE COPIED TO SYSTEM FREE STORAGE

THE PROCESS STACK SEGMENT

THE STACK FRAME

- STACK FRAMES ARE VARIABLE LENGTH, AND CONTAIN BOTH CONTROL INFORMATION AND DATA FOR ACTIVE PROCEDURES

- IN GENERAL, A STACK FRAME IS ALLOCATED EXPLICITLY BY THE PROCEDURE TO WHICH IT BELONGS (ITS OWNER) WHEN THAT PROCEDURE IS INVOKED

- STACK FRAMES ARE THREADED TO EACH OTHER WITH FORWARD AND BACKWARD POINTERS, MAKING IT EASY TO TRACE THE PROCESS HISTORY

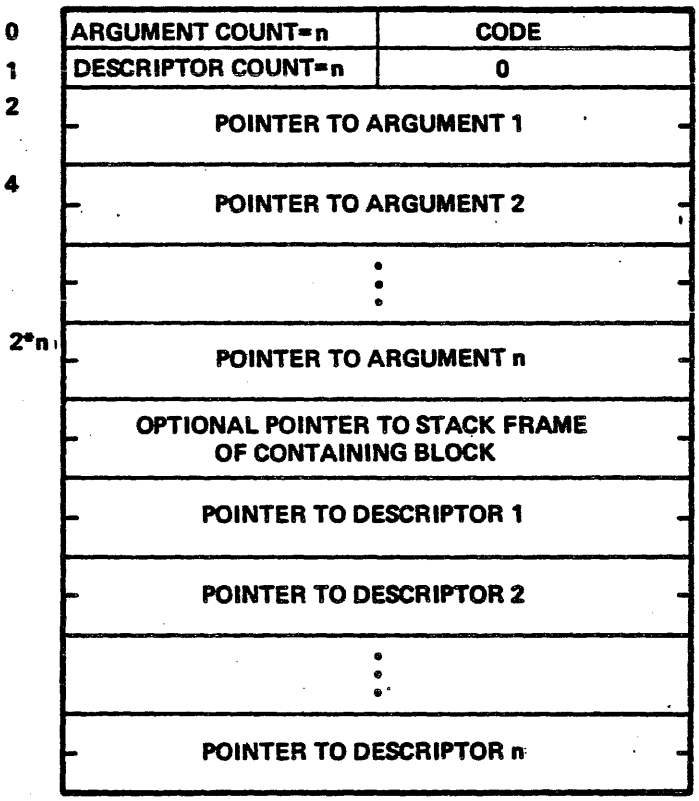
```
dcl 1 stack_frame based(sp) aligned,

    2 pointer_registers(0 : 7) ptr,  /* FOR ALM CALL PSEUDO-OP */
    2 prev_sp pointer,
    2 next_sp pointer,  /* IF EQUAL TO stack_end_ptr, POINTS TO
                           NEXT AVAILABLE STACK FRAME LOCATION */
    2 return_ptr pointer,  /* TELLS US WHERE TO RESUME EXECUTION */
    2 entry_ptr pointer,  /* POINTS TO THIS PROCEDURE'S ENTRY
                             POINT */
    2 operator_and_lp_ptr ptr,  /* POINTS TO OPERATOR SEGMENT BEING
                                   USED BY THIS PROCEDURE OR,
                                   IF ALM PROCEDURE, POINTS TO
                                   LINKAGE SECTION */
    2 arg_ptr pointer,  /* POINTS TO arg_list TO BE USED BY THIS
                           PROCEDURE */
    2 static_ptr ptr unaligned,  /* POINTS TO INTERNAL STATIC
                                    REGION */
    2 fio_ps_ptr ptr unal,  /* FOR FORTRAN I/O */
    2 on_unit_relp1 bit(18) unaligned,  /* POINTS TO A LIST OF
                                           ENABLED CONDITIONS
                                           (RELATIVE TO STACK
                                           FRAME BASE) */
    2 on_unit_relp2 bit(18) unaligned,  /* OBSOLETE */
    2 translator_id bit(18) unaligned, /* A CODED NUMBER INDICATING
                                          WHAT GENERATED THE OBJECT */
    2 operator_return_offset bit(18) unaligned,  /* USED BY SOME
                                                    pl1_operators_
                                                    FUNCTIONS; IF 0,
                                                    THEN A DEDICATED
                                                    REGISTER CONTAINS
                                                    RETURN LOCATION */
    2 x(0: 7) bit(18) unaligned,  }
    2 a bit(36),                  }  USED TO SAVE REGS WHEN AND IF THIS
    2 q bit(36),                  }  PROCEDURE DOES AN ALM CALL
    2 e bit(36),                  }
    2 timer bit(27) unaligned,
    2 pad bit(6) unaligned,
    2 ring_alarm_reg bit(3) unaligned;

          /* AUTOMATIC VARIABLES */

          /* THREADED LIST OF ON UNITS */
```

```
dcl 1 on_unit based aligned,
    2 name ptr,                 /* ptr to the condition name */
    2 body ptr,                 /* ptr to proc to handle condition */
    2 size fixed bin,           /* length of the condition name */
    2 next bit (18) unaligned,          /* rel ptr to next on unit*/
    2 flags unaligned,
      3 pl1_snap bit (1) unaligned,     /* if "1"b then call snap proc */
      3 pl1_ bit (1) unaligned,         /* "1"b indicates to use
                                           system condition handler */
      3 pad bit (16) unaligned,
    2 file ptr;                 /* ptr to file descriptor for pl1 I/O
                                   condition */
```

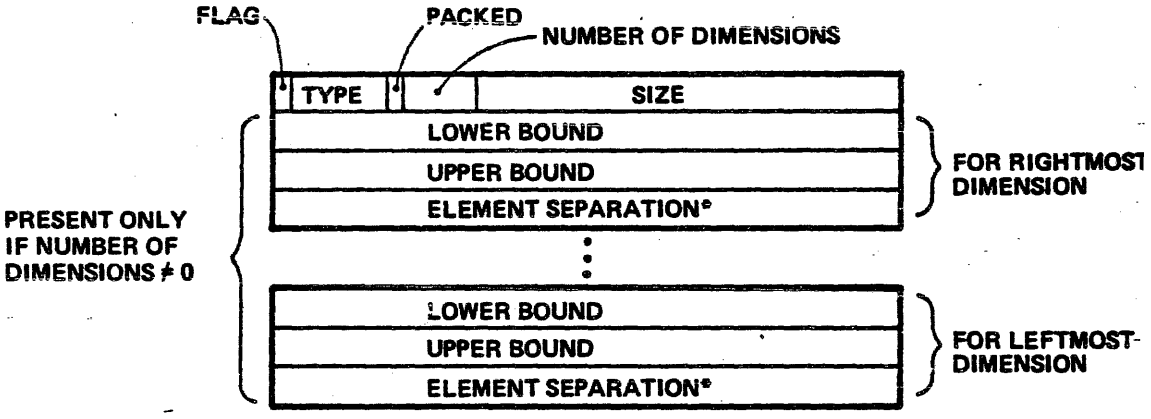
ARGUMENT LIST FORMAT

- AN OBJECT SEGMENT CREATES AN ARGUMENT LIST BEFORE INVOKING ANOTHER PROCEDURE
  - ARGUMENT LISTS CONFORM TO STANDARD FORMAT
  - ARGUMENT LISTS DO NOT CONTAIN THE ARGUMENTS THEMSELVES
  - SPECIFICALLY, THE ARGUMENT LIST CONTAINS
    - A TWO WORD HEADER
    - AN ARRAY OF ARG POINTERS
    - AN OPTIONAL POINTER TO STACK FRAME OF CONTAINING BLOCK
    - AN OPTIONAL ARRAY OF POINTERS TO ARGUMENT DESCRIPTORS

STANDARD ARGUMENT LIST



STANDARD DESCRIPTOR



* IN BITS IF PACKED; IN WORDS OTHERWISE

- IN THE PREVIOUS DIAGRAM:

  n IS THE NUMBER OF ARGUMENTS PASSED TO THE CALLED PROCEDURE

  code IS 4 FOR NORMAL INTERSEGMENT CALLS AND IS 10 (OCTAL) FOR CALLING SEQUENCES THAT CONTAIN AN EXTRA STACK FRAME POINTER - IT WILL BE PRESENT FOR CALLS TO PL/I INTERNAL PROCEDURES

  descriptor count = n OR 0

- A PROCEDURE WHICH MAY RECEIVE

  A VARYING NUMBER OF ARGS

  ARGS WITH VARYING EXTENTS

  MUST BE PASSED AN ARGUMENT LIST CONTAINING DESCRIPTORS OF THOSE ARGUMENTS, SO THAT THE CALLED PROCEDURE MAY KNOW HOW TO INTERPRET THE ARGUMENTS

- PL/1 ONLY PASSES DESCRIPTORS IF CALLED PROCEDURE IS DECLARED 'entry options (variable)' OR PARAMETERS OF CALLEE HAVE * EXTENTS

- IT IS THE RESPONSIBILITY OF A PROGRAM CALLING SUCH A PROCEDURE TO BUILD DESCRIPTORS AND INCLUDE THEM IN THE ARGUMENT LIST

- DESCRIPTORS HAVE A STANDARD FORMAT AS DEFINED BELOW:

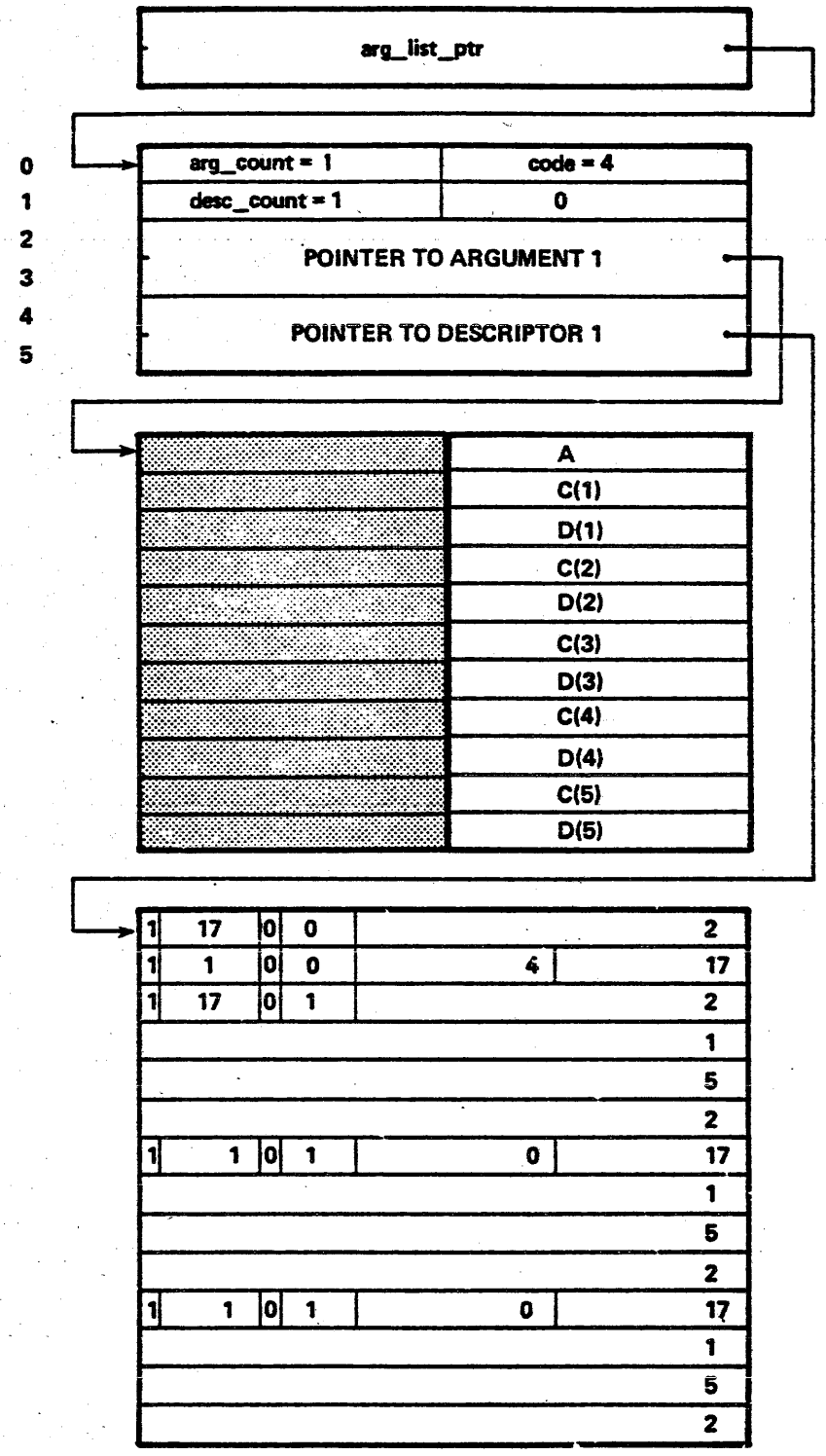
```
dcl 1 descriptor       aligned,
   (2 flag             bit(1),
    2 type             bit(6),
    2 packed           bit(1),
    2 number_dims      bit(4),  /* = 15 max */
    2 size             bit(24)) unaligned;  /* HAS VARIOUS
                                               MEANINGS */
```

  WHERE type IS ENCODED AS SHOWN ON THE NEXT PAGE.

```
/* BEGIN INCLUDE FILE ... std_descriptor_types.incl.pl1 */

/* This include file defines mnemonic names for the Multics
   standard descriptor types, using both pl1 and cobol terminology. */

dcl (real_fix_bin_1_dtype init (1),
     real_fix_bin_2_dtype init (2),
     real_flt_bin_1_dtype init (3),
     real_flt_bin_2_dtype init (4),
     cplx_fix_bin_1_dtype init (5),
     cplx_fix_bin_2_dtype init (6),
     cplx_flt_bin_1_dtype init (7),
     cplx_flt_bin_2_dtype init (8),
     real_fix_dec_9bit_ls_dtype init (9),
     real_flt_dec_9bit_dtype init (10),
     cplx_fix_dec_9bit_ls_dtype init (11),
     cplx_flt_dec_9bit_dtype init (12),
     pointer_dtype init (13),
     offset_dtype init (14),
     label_dtype init (15),
     entry_dtype init (16),
     structure_dtype init (17),
     area_dtype init (18),
     bit_dtype init (19),
     varying_bit_dtype init (20),
     char_dtype init (21),
     varying_char_dtype init (22),
     file_dtype init (23),
     real_fix_dec_9bit_ls_overp_dtype init (29),
     real_fix_dec_9bit_ts_overp_dtype init (30),
     real_fix_bin_1_uns_dtype init (33),
     real_fix_bin_2_uns_dtype init (34),
     real_fix_dec_9bit_uns_dtype init (35),
     real_fix_dec_9bit_ts_dtype init (36),
     real_fix_dec_4bit_uns_dtype init (38),  /* digit-aligned */
     real_fix_dec_4bit_ts_dtype init (39),  /* byte-aligned */
     real_fix_dec_4bit_bytealigned_uns_dtype init (40),  /* COBOL */
     real_fix_dec_4bit_ls_dtype init (41),  /* digit-aligned */
     real_flt_dec_4bit_dtype init (42),  /* digit-aligned */
     real_fix_dec_4bit_bytealigned_ls_dtype init (43),
     real_flt_dec_4bit_bytealigned_dtype init (44),
     cplx_fix_dec_4bit_bytealigned_ls_dtype init (45),
     cplx_flt_dec_4bit_bytealigned_dtype init (46),
```

```
      cobol_comp_6_dtype init (1),
      cobol_comp_7_dtype init (1),
      cobol_display_ls_dtype init (9),
      cobol_structure_dtype init (17),
      cobol_char_string_dtype init (21),
      cobol_display_ls_overp_dtype init (29),
      cobol_display_ts_overp_dtype init (30),
      cobol_display_uns_dtype init (35),
      cobol_display_ts_dtype init (36),
      cobol_comp_8_uns_dtype init (38),  /* digit aligned */
      cobol_comp_5_ts_dtype init (39),  /* byte aligned */
      cobol_comp_5_uns_dtype init (40),
      cobol_comp_8_ls_dtype init (41)   /* digit aligned */
      ) fixed bin internal static options (constant);

dcl (ft_integer_dtype init (1),
      ft_real_dtype init (3),
      ft_double_dtype init (4),
      ft_complex_dtype init (7),
      ft_external_dtype init (16),
      ft_logical_dtype init (19),
      ft_char_dtype init (21)
      ) fixed bin internal static options (constant);

dcl (label_constant_runtime_dtype init (24),
      int_entry_runtime_dtype init (25),
      ext_entry_runtime_dtype init (26),
      ext_procedure_runtime_dtype init (27),
      picture_runtime_dtype init (63)
      ) fixed bin internal static options (constant);


/* END INCLUDE FILE ... std_descriptor_types.incl.pl1 */
```

ARGUMENT DESCRIPTORS



TOPIC XII

Special Programming Techniques

	Page
Calling a Procedure on the Fly	12-1
Building Data Segments	12-4
Creating an Error Table	12-9

CALLING A PROCEDURE ON THE FLY

- MOTIVATION
  - CONSIDER THE COMMAND PROCESSOR, WHICH:
    - TAKES THE COMMAND LINE FROM THE USER
    - PARSES THE LINE INTO COMMAND NAME AND ARGUMENTS
    - CALLS THE COMMAND
  - HOW CAN IT CALL ALL THE VARIOUS COMMANDS (AND USER WRITTEN OBJECT SEGMENTS) WITHOUT HAVING DECLARED THEM AS AN EXTERNAL ENTRY?

CALLING A PROCEDURE ON THE FLY

- HOW CAN THE USER DO SUCH A "CALL ON THE FLY"?

  - hcs_$make_entry

    - call hcs_$make_entry (ref_ptr, entryname, entry_point_name, entry_point, code);

    - GIVEN A REFERENCE NAME AND AN ENTRY POINT NAME, RETURNS THE ENTRY VALUE OF THE SPECIFIED ENTRY POINT

      - IF THE REFERENCE NAME HAS NOT BEEN INITIATED, THE SEARCH RULES ARE USED TO FIND A SEGMENT WITH THAT NAME, THE SEGMENT IS MADE KNOWN AND THE REFERENCE NAME INITIATED

  - cu_$generate_call

    - call cu_$generate_call (proc_entry, arg_ptr);

    - USED TO INVOKE A PROCEDURE BY PASSING IT AN ENTRY VALUE AND AN ARGUMENT POINTER (THE ENTRY VALUE WAS OBTAINED BY A PREVIOUS CALL TO hcs_$make_entry)

      - THE USER MUST HAVE PROVIDED AN ARGUMENT LIST STRUCTURE (EVEN IF NO ARGUMENTS ARE PASSED TO THE PROCEDURE BEING INVOKED)

- SIMPLIFIED EXAMPLE:

```
generate_pwd: proc;
dcl 1 arg_list aligned,
      2 header,
        3 arg_count fixed bin (17) unsigned unal init (0),
        3 pad1 bit (1) unal,
        3 call_type fixed bin (18) unsigned unal,
        3 desc_count fixed bin (17) unsigned unal,
        3 pad2 bit (19) unal,
      2 arg_ptr ptr init (null()),
      2 desc_ptr ptr init (null());

dcl cu_$generate_call entry (entry, ptr);
dcl hcs_$make_entry entry (ptr, char (*), char (*), entry,
                           fixed bin (35));
dcl code fixed bin (35);
dcl com_err_ entry options (variable);
dcl entry_point entry variable;

    call hcs_$make_entry (null(), "pwd", "pwd", entry_point, code);
    if code ^= 0 then do;
        call com_err_ (code, "generate");
        return;
    end;
     call cu_$generate_call (entry_point, addr (arg_list));
end generate_pwd;
```

- create_data_segment_

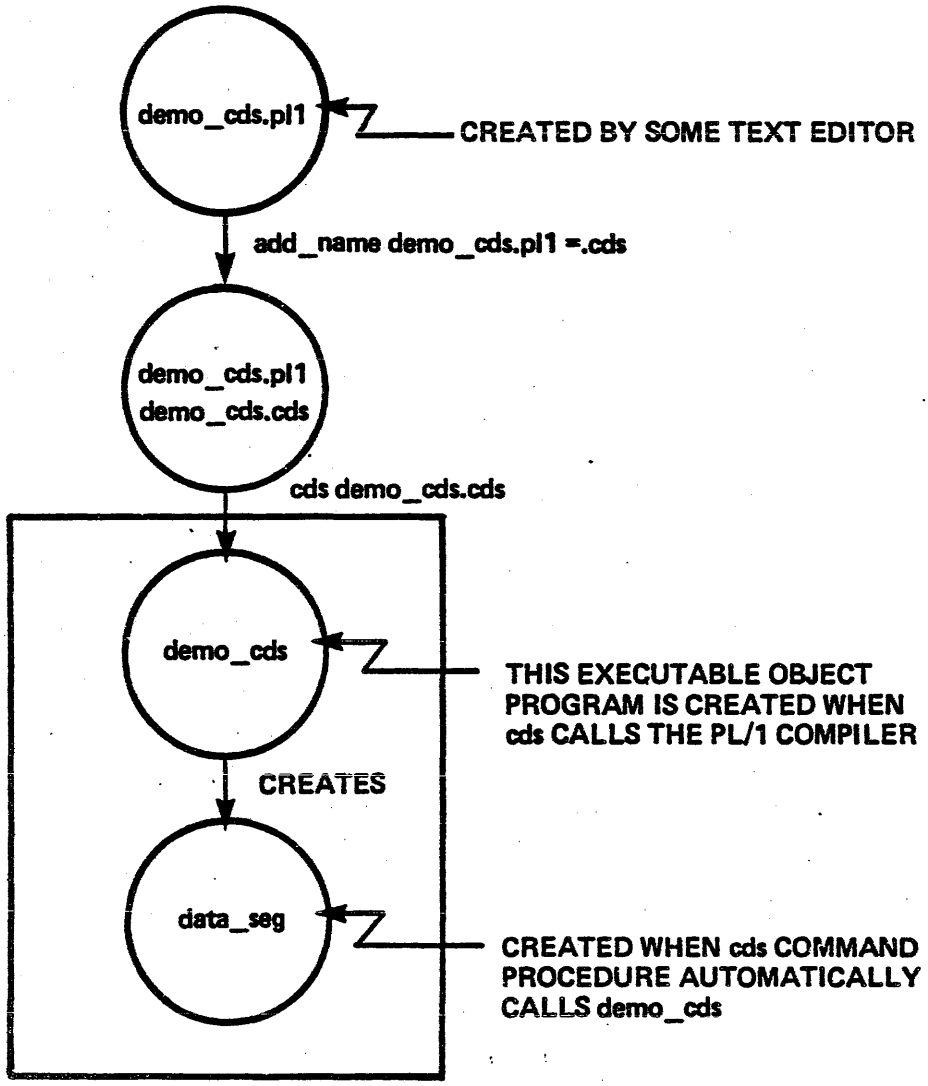
  - call create_data_segment_ (cds_arg_ptr, code);

  - USED IN CONJUNCTION WITH THE create_data_segment COMMAND TO CREATE A DATA SEGMENT IN STANDARD OBJECT FORMAT

    - REFERENCES A PL/1 DATA STRUCTURE (SUPPLIED BY THE USER) WHEN BUILDING THE DATA SEGMENT

    - PERMITS THE CALLER TO CREATE A DATA SEGMENT FOR WHICH THE LEVEL-2 MEMBERS OF THE PL/1 STRUCTURE ARE ACCESSIBLE AS ENTRY POINTS (NOTE THE pds AND sys_info SEGMENTS)

HOW cds WORKS



```
! pr >ldd>include>cds_args.incl.pl1


/* BEGIN INCLUDE FILE cds_args.incl.pl1 */

dcl 1 cds_args based aligned,
    2 sections (2),
      3 p ptr,      /* pointer to data for text/static section */
      3 len fixed bin (18),     /* size of text/static section */
      3 struct_name char (32),   /* name of declared structure */
    2 seg_name char (32),         /* name to create segment by */
    2 num_exclude_names fixed bin,  /* number in exclude array */
    2 exclude_array_ptr ptr,       /* pointer to exclude array */
    2 switches,                          /* control switches */
      3 defs_in_link bit (1) unal, /* says put defs in linkage */
      3 separate_static bit (1) unal,  /* separate static section
                                                      wanted */
      3 have_text bit (1) unal,    /* ON if text section given */
      3 have_static bit (1) unal,/* ON if static section given */
      3 pad bit (32) unal;                    /* must be zero */

dcl  exclude_names (1) char (32) based;
                  /* pointed to by cds_args.exclude_array_ptr */

/* END INCLUDE FILE cds_args.incl.pl1 */
```

```
! pr demo_cds.pl1


demo_cds: proc;
dcl  create_data_segment_ entry (ptr, fixed bin (35));
%include cds_args;
dcl  cds_arg_ptr ptr;
dcl (ioa_, com_err_) entry options (variable);
dcl  code fixed bin (35);
dcl (size, null) builtin;
dcl 1 entrypointnames based (cds_arg_ptr -> cds_args.sections (1).p),
    2 alpha fixed bin (35),
    2 beta char (4),
    2 gamma bit (36),
    2 delta ptr;

   allocate cds_args set (cds_arg_ptr);
   cds_arg_ptr -> cds_args.sections (1).len = size (entrypointnames);
   cds_arg_ptr -> cds_args.sections (1).struct_name = "entrypointnames";
   cds_arg_ptr -> cds_args.seg_name = "data_seg";
   cds_arg_ptr -> cds_args.sections (2).len = 0;
   cds_arg_ptr -> cds_args.sections (2).struct_name = "";
   cds_arg_ptr -> cds_args.num_exclude_names = 0;
   cds_arg_ptr -> cds_args.exclude_array_ptr = null ();
   cds_arg_ptr -> cds_args.switches.defs_in_link = "0"b;
   cds_arg_ptr -> cds_args.switches.separate_static = "0"b;
   cds_arg_ptr -> cds_args.switches.have_text = "1"b;
   cds_arg_ptr -> cds_args.switches.have_static = "0"b;
   cds_arg_ptr -> cds_args.switches.pad = "0"b;

   call create_data_segment_ (cds_arg_ptr, code);
   if code ^= 0 then call com_err_ (code, "demo_cds");
      else call ioa_ ("Segment creation complete.");
end demo_cds;
```

```
!an demo_cds.pl1 =.cds
 r 14:32 0.096 2

!cds demo_cds
 CDS -PL/I 26a
 Segment creation complete.
 r 14:32 1.699 53

!ls -first 3

 Segments = 43, Lengths = 155.

 r       1  data_seg
 re      1  demo_cds
 r w     1  demo_cds.pl1
            demo_cds.cds

 r 14:33 0.151 2

!pli data_seg

                      data_seg    10/10/80  1433.9 mst Fri


 Object Segment >udd>MEDmult>F15C>do>data_seg
 Created on 10/10/80  1432.9 mst Fri
 by NDibble.MEDmult.a
 using create_data_segment_, Version II of Friday, May 16, 1980

          Object    Text    Defs    Link    Symb  Static
 Start         0       0       6      52      62      62
 Length      224       6      44      10     126       0


 6 Definitions:


 segname:    data_seg

 text|0          alpha
 text|1          beta
 text|4          delta
 text|2          gamma
 symb|0          symbol_table


 No Links.
```

CREATING AN ERROR TABLE

- USERS MAY CREATE THEIR OWN STATUS CODE TABLE

  - THEY ARE CONSTRUCTED USING ALM MACROS DEFINED IN et_macros.incl.alm

  - THE SKELETON OF THE SOURCE CODE IS AS FOLLOWS:

```
include et_macros
et .....
ec .....
ec .....
ec .....
 .
 .
 .
end
```

  - THERE ARE THUS 2 MACROS USED

    - THE "et" MACRO INITIALIZED THE CODE TABLE AND MUST APPEAR FIRST

```
et <name_of_table>
```

    - THE "ec" MACRO ASSOCIATES A STATUS CODE NAME WITH A SHORT MESSAGE AND A LONG MESSAGE

```
ec <code_name>,<short_message>,(<long_message>)
```

  - EXAMPLE:

```
include et_macros
et   user_errors
ec   too_few_arguments, toofew,(There were too few arguments.)
ec   could_not_access_data,nopriv,
     (User is not sufficiently priviliged to access data.
ec   (fatal,disaster),disaster,,
     (There was a disastrous error in the data base.)
end
```

CREATING AN ERROR TABLE

- THE CODE NAME:
  - MUST BE 31 CHARACTERS OR LESS IN LENGTH
  - MULTIPLE NAMES MAY BE GIVEN (SEPARATED BY COMMAS AND ENCLOSED IN PARENTHESIS)
- THE SHORT MESSAGE:
  - MUST BE 8 OR LESS CHARACTERS IN LENGTH
  - IF OMITTED, IT IS SET TO THE CODE NAME
- THE LONG MESSAGE MUST BE:
  - 100 OR LESS CHARACTERS IN LENGTH
  - ENCLOSED IN PARENTHESIS

```
! pr weird_errors.alm

  include et_macros
  et weird_errors
  ec error_a,number2,(Warning: The number has reached two.)
  ec error_b,number3,(Second Warning: The number has reached three.)
  end

! alm weird_errors.alm
  ALM

! pr calling_program.pl1

   calling_program:  proc;
     dcl x fixed bin (17) external static init(0);
     dcl com_err_ entry options (variable);
     dcl my_subprogram entry (fixed bin(17), fixed bin (35));
     dcl code fixed bin (35);
     dcl sysprint file;
     x = x + 1;
     put data (x);
     put skip;
     call my_suprogram (x, code);
     if code ^= 0 then call com_err_ (code, "calling_program");
   end calling_program;

! pr my_subprogram.pl1

   my_subprogram:  proc (input, code);
     dcl input fixed bin (17);
     dcl code fixed bin (35);
     dcl weird_errors$error_a fixed bin (35) external static;
     dcl weird_errors$error_b fixed bin (35) external static;
     code = 0;
     if input = 2 then code = weird_errors$error_a;
     if input = 3 then code = weird_errors$error_b;
   end my_subprogram;

! calling_program
  x=        1;

! calling_program
  x=        2;
  calling_program: Warning: The number has reached two.

! calling_program
  x=        3;
  calling_program: Second Warning: The number has reached three.

! calling_program
  x=        4;
```

YOU ARE NOW READY FOR WORKSHOP #6

TOPIC XIII

The Process Environment

	Page
The Standard Process Environment	13-1
Process Creation in the Supervisor Ring	13-3
Process Initialization in the User Ring	13-5
Modifying the Process Environment	13-9
Project Administration	13-11
Closed Subsystems	13-13
Limited Subsystems	13-14
New Subsystems	13-18

THE STANDARD PROCESS ENVIRONMENT

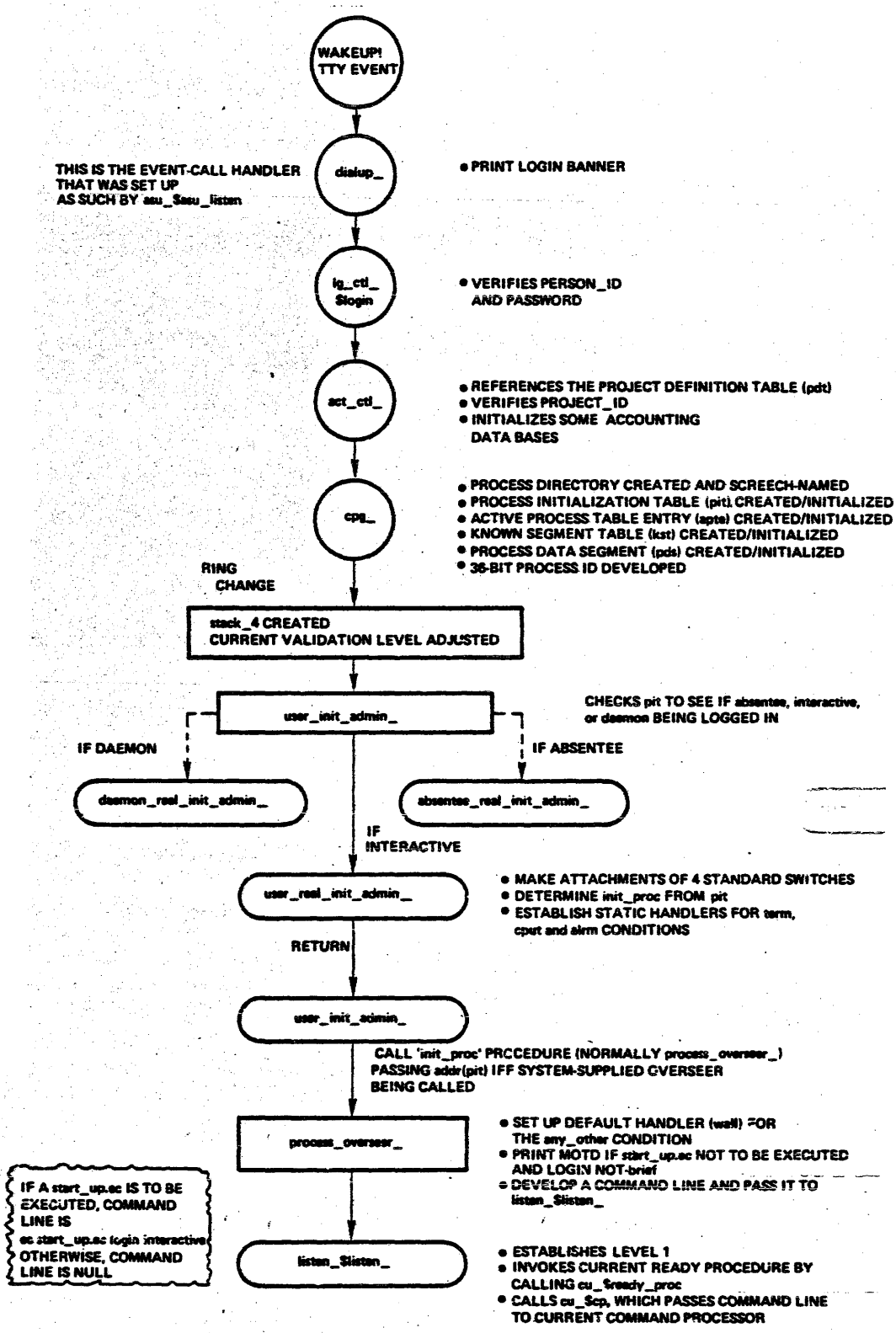
- THE STANDARD PROCESS ENVIRONMENT IS ESTABLISHED FOR A PROCESS BY THE SUPERVISOR WHEN THAT PROCESS IS CREATED

- THE SERIES OF PROGRAMS INVOKED BEGINNING WITH THE TIME THAT A USER CONNECTS WITH THE FRONT-END PROCESSOR AND TERMINATING WITH THE FIRST READY MESSAGE PRINTED ON THE USER'S TERMINAL IS CONSIDERED THE "PROCESS CREATION CYCLE"

- MUCH OF THE WORK DONE TO CREATE A PROCESS IS DONE BY THE SUPERVISOR IN THE SUPERVISOR RING (RING 0), AND SOME OF THE WORK IS DONE IN THE USER'S RING

- BEFORE EXAMINING THE OPTIONS AVAILABLE TO THE DESIGNER FOR MODIFYING OR REPLACING THE STANDARD PROCESS ENVIRONMENT, AN EXAMINATION OF WHAT STEPS THE SUPERVISOR NORMALLY TAKES WILL BE WORTHWHILE

THE STANDARD PROCESS ENVIRONMENT



PROCESS CREATION IN THE SUPERVISOR RING

- THE "SYSTEM CONTROL PROCESS" (Initializer.SysDaemon) IS RESPONSIBLE FOR RESPONDING TO ATTEMPTS TO CREATE A PROCESS, AND SOME OF THE MODULES EXECUTED BY THE SYSTEM CONTROL PROCESS IMPLEMENT PROCESS CREATION AS FOLLOWS:

  - ANSWERING SERVICE

    - IS RESPONSIBLE FOR

      - INTERACTIVE LOGINS
      - HANGUPS
      - THE LOGGING OF THESE ACTIVITIES
      - MANAGING THE COMMUNICATION LINES CURRENTLY ATTACHED TO SYSTEM (USES CHANNEL DEFINITION TABLE)

        - asu_$asu_listen ESTABLISHES AN EVENT-CALL CHANNEL FOR EACH COMLINE, MAKING 'dialup_' THE EVENT HANDLER
        - A "CONNECT" ON A COMLINE IS CONSIDERED A "TTY EVENT", WHICH RESULTS IN A WAKEUP ON ONE OF THESE CHANNELS

  - dialup_ (IN >tools>bound_user_control_)

    - IS AUTOMATICALLY CALLED WHEN A TERMINAL IS CONNECTED
    - PRINTS LOGIN BANNER ON THE TERMINAL
    - READS INITIAL TYPED LINE (login, enter, enterp, dial, ETC) AND PASSWORD
    - CALLS lg_ctl_$login TO VERIFY PERSON_ID AND PASSWORD

- dialup_, CONTINUED
  - CALLS act_ctl_$open_account AND act_ctl_$cp TO
    - VERIFY PROJECT_ID (VIA A CHECK OF PROJECT DEFINITION TABLE)
    - INITIALIZE SOME ACCOUNTING DATA BASES FOR THIS PROCESS
  - FINALLY CALLS cpg_ (CREATE PROCESS GROUP) TO ACTUALLY CREATE PROCESS
    - 36-BIT PROCESS ID DEVELOPED
    - PROCESS DIRECTORY CREATED AND SCREECH-NAMED
    - PROCESS INITIALIZATION TABLE CREATED/INITIALIZED
    - ACTIVE PROCESS TABLE ENTRY CREATED/INITIALIZED
    - KST AND DSEG CREATED/INITIALIZED
    - PROCESS DATA SEGMENT (RING 0 STACK) CREATED/INITIALIZED
- FINALLY, CONTROL IS PASSED IN A RATHER UNUSUAL FASHION TO THE OUTER RING IN WHICH THE PROCESS IS TO RESIDE
  - A STACK SEGMENT IS CREATED IN THE ULTIMATE RING (USUALLY 4)
  - A STACK FRAME IS LAID DOWN FOR A USER RING INITIALIZATION PROGRAM
  - THE CURRENT VALIDATION LEVEL IS ADJUSTED TO THAT RING LEVEL
  - A PSEUDO-RETURN IS MADE TO THAT USER-RING INITIALIZATION PROGRAM

PROCESS INITIALIZATION IN THE USER RING

- THE STANDARD PROCESS ENVIRONMENT IS ESTABLISHED IN THE USER RING BY THE INVOCATION OF THE USER-RING PROCESS CREATION PROGRAMS:

  - user_init_admin_ FUNCTIONS

    - AFTER BEING INVOKED THROUGH ONE OF THREE ENTRY POINTS:

      - user_init_admin_
      - absentee_init_admin_
      - daemon_init_admin_

    - IT CALLS EITHER

      - user_real_init_admin_$user_real_init_admin_
      - OR daemon_real_init_admin_$daemon_real_init_admin_
      - OR absentee_real_init_admin_$absentee_real_init_admin_

    - WHEN RETURNED TO BY ONE OF ABOVE, IT CALLS THE 'init_proc' PROCEDURE

      - THE VALUE OF 'init_proc' IS RETURNED BY ONE OF THE ABOVE THREE
      - A pit_ptr IS PASSED TO init_proc

PROCESS INITIALIZATION IN THE USER RING

- user_real_init_admin_ FUNCTIONS
  - DEVELOP PTR TO PROCESS INITIALIZATION TABLE (pit)
  - DETERMINE FROM pit WHICH IO_MODULE TO USE (USUALLY tty_)
  - ATTACH user_input, user_output, error_output AS SYNONYMS FOR user_i/o
  - ATTACH AND OPEN user_i/o TO CHANNEL_NAME FOUND IN pit
  - DETERMINE (FROM pit) WHAT 'init_proc' PROCEDURE IS TO BE USED (USUALLY process_overseer_$process_overseer_)
    - PROJECT ADMINISTRATOR MAY FORCE A PARTICULAR init_proc TO BE USED
    - USER MAY HAVE BEEN GIVEN PERMISSION TO USE '-po' CONTROL ARG OF THE login COMMAND, AND COULD HAVE USED IT TO SPECIFY THE DESIRED init_proc
  - SET UP STATIC HANDLERS FOR 'alrm', 'cput', AND 'term' CONDITIONS BY CALLING sct_manager_$set

- process_overseer_ FUNCTIONS
  - CALL condition_ TO SET UP 'default_error_handler_$wall' AS THE HANDLER FOR 'any_other' CONDITION
    - IMPORTANT: IN ORDER TO COMPLETELY CONTROL A USER, ONE MUST BE ABLE TO CHANGE THIS DEFAULT BEHAVIOR. THIS IS ONE OF THE MOST IMPORTANT REASONS FOR FURNISHING A SPECIAL init_proc.
  - CHECK pit TO SEE IF THIS IS new_proc OR login
  - CHECK pit TO SEE IF start_up.ec SHOULD BE INVOKED
    - RECALL: USE OF -ns LOGIN CONTROL ARG IS RESTRICTABLE
  - BUILD ONE OF TWO INITIAL COMMAND LINES:
    - exec_com startup_dir>start_up instance type
      - WHERE startup_dir IS HOME DIRECTORY, PROJECT DIRECTORY, OR >sc1
      - WHERE instance IS 'login' OR 'new_proc'
      - AND type IS 'interactive' OR 'absentee'
      - home_dir, instance, AND type ARE DETERMINED FROM pit
    - "" (NULL COMMAND LINE)
  - TERMINATE pit
  - CALL listen_$listen_ WITH INITIAL COMMAND LINE

- listen_
  - BASIC FUNCTIONS
    - 'LISTENS' FOR LINES TYPED BY THE USER
    - PASSES COMMAND LINE ON TO THE CURRENT COMMAND PROCESSOR
    - INVOKES CURRENT READY PROCEDURE WHEN RETURNED TO (AFTER COMMAND LINE EXECUTION OR PRINTING OF ERROR MESSAGES)
    - ALSO ENABLES QUITS
  - ENTRY POINTS IN listen_:
    - listen_
      - INVOKED ONLY BY init_proc
      - ESTABLISHES A FRAME CONSIDERED TO BE "FIRST LEVEL"
      - PASSES com_line_ptr AND com_line_length TO cu_$cp
      - CALLS cu_$ready_proc WHEN RETURNED TO
    - release_stack
      - CALLED WHEN AN ATTEMPT IS MADE TO REENTER COMMAND LEVEL (LEVEL ≠ 1)
      - ESTABLISHES ITS STACK FRAME AS CURRENT LEVEL OF LISTENER
      - 'REMEMBERS' PREVIOUS LEVEL AND VERY FIRST LEVEL OF LISTENER (FOR PURPOSES OF release)
      - PASSES com_line_ptr AND com_line_length TO cu_$cp
      - CALLS cu_$ready_proc WHEN RETURNED TO
    - OTHER ENTRY POINTS
      - USED BY OTHER PROCEDURES TO OBTAIN INFO ABOUT

        WHERE TO RELEASE TO
        WHERE TO 'start'

- ONE CAN MODIFY THE STANDARD PROCESS ENVIRONMENT USING SEVERAL DIFFERENT TECHNIQUES

- THE AMOUNT OF CONTROL DESIRED ON A PROCESS CAN BE CLASSIFIED AS FOLLOWS:

  - SIMPLE CONTROL

    - USING 'exec_com' SEGMENTS, THE USER CAN BE RESTRICTED TO LABEL ENTRY POINTS IN THE ec SEGMENT ITSELF

  - STANDARD PROCESS OVERSEERS

    - PROVIDED BY SYSTEM TO CONTROL THE ENVIRONMENT OF THE USER IN VARYING LEVELS OF RESTRICTION

      - accounts_overseer IS USED BY 'REGISTRATION AND ACCOUNTING ADMINISTRATORS' TO LIMIT THE NUMBER OF THINGS THEY CAN DO (SEE APPENDIX B)

    - SEE APPENDIX E FOR SOME OF THE STANDARD OVERSEERS

  - CLOSED SUBSYSTEM OVERSEERS

    - THE 'fst_process_overseer_' IS AN EXAMPLE OF AN OVERSEER WHICH PLACES THE USER IN A COMPLETELY CLOSED ENVIRONMENT FROM WHICH ESCAPE IS IMPOSSIBLE

- LIMITED SUBSYSTEMS
  - THE SYSTEM PROVIDES THREE WAYS OF FORCING USERS INTO A LIMITED SUBSYSTEM
- USER-CREATED SUBSYSTEMS
  - BY WRITING ONE'S OWN PROCESS OVERSEER, ONE CAN ATTAIN COMPLETE, CUSTOMIZED CONTROL OVER A PROCESS

- MOST OF THESE FUNCTIONS REQUIRE INVOLVEMENT OF A PROJECT ADMINISTRATOR (BECAUSE OF NEED TO MODIFY pmf AND INSTALL NEW pdt)

- EXAMPLE OF A pmf

```
Projectid:          F15D;
Initproc:           process_overseer_;
Grace:              60;
Attributes:         vinitproc,vhomedir,multip,nostartup,dialok,
                    disconnect_ok,save_on_disconnect;
Limit:              75.00;

  personid:         Student_01;
  personid:         Student_02;
  personid:         Student_03;
  personid:         Student_04;
  personid:         Student_05;
  personid:         Student_06;
  personid:         Student_07;
end;
```

PROJECT ADMINISTRATION

- THE LIST OF PERSONS WHO MAY LOG IN ON A PROJECT IS CONTAINED IN A BINARY TABLE, THE PROJECT DEFINITION TABLE (pdt), WHICH RESIDES IN THE DIRECTORY >sc1>pdt
  - ONE pdt SEGMENT EXISTS FOR EACH PROJECT
  - ONE pdt ENTRY EXISTS FOR EACH USER, SPECIFYING THE USER'S ATTRIBUTES AND RESOURCE LIMITS ON THE PARTICULAR PROJECT
  - USING THE cv_pmf COMMAND, A PROJECT ADMINISTRATOR CREATES A TEMPORARY pdt FROM A SEGMENT KNOWN AS THE PROJECT MASTER FILE (pmf), WHICH IS USUALLY UNDER THE PROJECT DIRECTORY
  - THE TEMPORARY pdt IS INSTALLED IN THE SYSTEM DIRECTORY BY THE PROJECT ADMINISTRATOR USING THE install COMMAND
  - AT LOGIN TIME, act_ctl USES THE APPROPRIATE pdt TO DETERMINE WHICH OPTIONS AND RESOURCES ARE AVAILABLE TO A USER
- SEE MAM PROJECT ADMINISTRATOR (Order No. AK51) FOR COMPLETE DETAILS

- SOME OF THE ATTRIBUTE INFORMATION MAINTAINED FOR EACH USER IN THE PDT IS GIVEN BELOW:

  - homedir - ABSOLUTE PATHNAME OF USER'S HOME DIRECTORY

  - initproc - NAME OF THE USER'S PROCESS OVERSEER PROCEDURE

  - attributes:

    - nobump - USER NOT SUBJECT TO PREEMPTION

    - dialok - USER MAY USE THE DIAL FACILITY

    - multip - USER MAY LOG IN MORE THAN ONE INTERACTIVE PROCESS

    - vinitproc - USER MAY SPECIFY PROCESS OVERSEER AT LOGIN

    - vhomedir - USER MAY SPECIFY HOME DIRECTORY AT LOGIN

    - nostartup - USER MAY ESCAPE FROM USING HIS start_up.ec

    - AND SO ON...

MODIFYING THE PROCESS ENVIRONMENT

CLOSED SUBSYSTEMS

- A CLOSED SUBSYSTEM IS A SUBSYSTEM IN WHICH THE Multics SYSTEM IS NOT DIRECTLY AVAILABLE - RATHER, THOSE PROGRAMS IMPLEMENTING THE CLOSED ENVIRONMENT TAKE FULL CONTROL OVER A PROCESS

- THE SYSTEM-SUPPLIED OVERSEER 'accounts_overseer_' IMPLEMENTS A CLOSED SUBSYSTEM

  - accounts_overseer_ HANDLES ALL INPUT FROM THE TERMINAL (I.E., IT IS IT'S OWN LISTENER), AND IT SEVERELY RESTRICTS THE USER TO A SMALL SET OF COMMANDS

  - IT MAKES USE OF AN exec_com SEGMENT, '>tools>master.ec' TO IMPLEMENT SPECIAL FUNCTIONS FOR THE "REGISTRATION AND ACCOUNTING ADMINISTRATOR" WHO WILL BE OPERATING UNDER THIS ENVIRONMENT

- THE SYSTEM-SUPPLIED 'fst_process_overseer_' ALSO IMPLEMENTS A CLOSED SUBSYSTEM, A "TIME-SHARING FORTRAN" SYSTEM

- THE SYSTEM_SUPPLIED OVERSEER 'project_start_up_' MAY BE USED TO IMPLEMENT A CLOSED SUBSYSTEM

  - project_directory>project_start_up.ec ALWAYS EXECUTED BEFORE start_up.ec (WITH QUITS DISABLED)

MODIFYING THE PROCESS ENVIRONMENT

LIMITED SUBSYSTEMS

- IN A LIMITED SUBSYSTEM, THE USER IS LIMITED TO A SET OF COMMANDS CONTAINED IN A "COMMAND LIST" SEGMENT

- THREE MEANS OF UTILIZING THE "LIMITED SERVICE SUBSYSTEM":

  - IF init_proc = lss_login_responder_$lss_login_responder_

    - LIST OF COMMANDS ARE CONTAINED IN >sss>lss_command_list_

    - command_processor_ WILL CHECK EVERY COMMAND ENTERED BY THE USER AGAINST THIS LIST - ONLY IF IT IS ON LIST WILL IT BE EXECUTED

    - IN ADDITION, A CPU USAGE GOVERNOR WILL BE ENABLED, LIMITING THE PROCESS TO 'ratio' CPU SECONDS PER 'interval' REAL SECONDS

      - NOTE: IF >sss>lss_command_list_ NOT FOUND AT LOGIN TIME, MESSAGE RELAYED IS:

        "The system is currently unavailable".

        (A logout -hold IS DONE)

  - lss_login_responder_$limited_command_system_

    - SIMILAR TO ABOVE, BUT lss_command_list_ IS SEARCHED FOR IN USER'S PROJECT DIRECTORY

- THE COMMAND, enter_lss
  - USER SPECIFIES SEGMENT CONTAINING THE "COMMAND LIST"
  - CONSIDER IF enter_lss COMMAND APPEARS IN project_start_up.ec

- COMMAND TABLES ARE CREATED USING THE 'make_commands' COMMAND
  - make_commands ACCEPTS THE NAME OF AN ASCII SEGMENT WITH THE SUFFIX OF 'ct', AND PRODUCES A COMMAND TABLE SEGMENT (THE ENTRYNAME WITHOUT THE 'ct' SUFFIX)
  - ASCII COMMAND LIST CONTAINS THREE TYPES OF STATEMENTS:
    - ratio: R;
      - SPECIFIES THE NUMBER OF CPU SECONDS MAXIMUM ALLOWED FOR THE PROCESS DURING THE SPECIFIED INTERVAL
    - interval: N;
      - SPECIFIES THE NUMBER OF REAL-TIME SECONDS WITHIN WHICH THE PROCESS IS LIMITED TO 'R' CPU SECONDS
    - (command_list): pathname;
command_name: pathname;
      - SPECIFIES THAT THE COMMANDS IN THE (BLANK DELIMITED) command_list OR THE COMMAND SPECIFIED BY command_name ARE ALLOWED, AND THAT THEY SHOULD CAUSE THE PROCEDURE SPECIFIED BY PATHNAME TO BE INVOKED WHEN THEY ARE ENTERED AS COMMANDS

- A COMMAND LIST EXAMPLE

```
/* set ratio and interval length */

ratio:                          45;
interval:                       120;

/* define commands */

(addname an):                   ;
calc:                           ;
(delete dl):                    ;
(deletename dn):                ;
(list ls):                      >udd>MED>nd>list;
logout:                         ;
(print pr):                     ;
(program_interrupt pi):         ;
(rename rn):                    ;
(start sr):                     ;
edit:                           >sss>qedx;
```

```
r 19:39 0.116 2

!pr xxx.ct 1

 logout:   >sss>logout;
 pwd:      >sss>pwd;
 ls:       >sss>ls;
 new_proc: >sss>new_proc;
 probe:    >sss>probe;

 r 19:39 0.104 3

!make_commands xxx
 r 19:43 0.248 14

!ls -first 1

 Segments = 72, Lengths = 80.

 re    1  xxx

 r 19:43 0.216 13

!enter_lss xxx
 r 19:44 0.065 0

!who
 who is not a legal command
 r 19:44 0.058 0

 QUIT
 r 19:44 0.118 4 level 2

!who
 who is not a legal command
 r 19:44 0.035 2 level 2

!pr xxx.ct
 pr is not a legal command
 r 19:44 0.033 0 level 2

!logout
 NDibble MED logged ;out 02/26/81  1459.8 mst Thu
 CPU usage 7 sec, memory usage 23.0 units, cost $0.48.
 hangup
```

MODIFYING THE PROCESS ENVIRONMENT

NEW SUBSYSTEMS

- IN THE MOST EXTREME CASE, A DESIGNER MAY IMPLEMENT HIS/HER OWN PROCESS ENVIRONMENT BY REPLACING THE STANDARD PROCESS OVERSEER WITH HIS/HER OWN PROCESS OVERSEER

  - THE DESIGNER IS WARNED TO BE SURE TO PERFORM THE CRITICAL FUNCTIONS WHICH ARE NORMALLY PERFORMED BY STANDARD OVERSEERS, BUT BEYOND THESE RESTRICTIONS, THE DESIGNER IS FREE TO IMPLEMENT ANY ENVIRONMENT DESIRED

  - 'process_overseer_.pl1' IS A GOOD REFERENCE FOR THE DESIGNER ATTEMPTING TO CREATE A NEW ONE (SEE APPENDIX B)

YOU ARE NOW READY FOR WORKSHOP #7

TOPIC XIV

Dialing Terminals to a Process

	Page
Overview	14-1
Implementation of the Dial Facility	14-2
dial_manager_	14-3
Dialing Terminals to a Process	14-4
Subroutines	14-5
The 'dial' Command	14-7
An Example	14-8
Dialing Out to a Terminal	14-13
dial_manager_ Entry Points	14-14

OVERVIEW

- NORMALLY THERE IS A ONE-TO-ONE CORRESPONDENCE BETWEEN AN INTERACTIVE PROCESS AND A TERMINAL DEVICE

- HOWEVER, Multics PROVIDES THE 'dial' FACILITY WHICH ENABLES A PROCESS TO CONTROL MORE THAN ONE TERMINAL DEVICE

- THE DIAL FACILITY IS PART OF THE ANSWERING SERVICE

- THE ANSWERING SERVICE IS RESPONSIBLE FOR LISTENING TO THE COMMUNICATION LINES ATTACHED TO THE FRONT-END PROCESSOR (FNP)

- A PROCESS HAVING THE 'dialok' ATTRIBUTE (ASSIGNABLE BY THE SYSTEM AND PROJECT ADMINISTRATORS) MAY USE THE DIAL FACILITY

  - ACCEPT DIALED TERMINALS (REQUIRES NO SPECIAL HARDWARE)

  - DIAL OUT TO TERMINALS (REQUIRES ACCESS TO AUTO-CALL CHANNEL)

IMPLEMENTATION OF THE DIAL FACILITY

- ipc_ ESTABLISHES THE EVENT CHANNEL REQUIRED FOR COMMUNICATION BETWEEN THE USER PROCESS AND THE ANSWERING SERVICE

- dial_manager_ INTERFACES TO THE ANSWERING SERVICE'S DIAL FACILITY

- convert_dial_message_ INTERPRETS THE SPECIAL IPC MESSAGE SENT BY THE ANSWERING SERVICE TO A USER PROCESS

- THE 'dial' PRE-ACCESS COMMAND IS USED BY A TERMINAL OPERATOR ATTEMPTING TO DIAL IN TO AN EXISTING PROCESS (AG92)

- A POINTER TO AN ARGUMENT STRUCTURE IS PASSED IN ALL CALLS TO dial_manager_, AND THE MEANING OF THE STRUCTURE MEMBERS VARIES WITH EACH ENTRY POINT

```
dcl 1 dial_manager_arg based aligned,
    2 version          fixed bin /* MUST BE SET TO 1 */
    2 dial_qualifier   char(22),
    2 dial_channel     fixed bin(71), /* EVENT-WAIT CHANNEL */
    2 channel_name     char(32);
```

  - dial_qualifier
    - WILL BE A 'dial_id' TO BE SUPPLIED WHEN THE dial COMMAND IS TYPED (IF WE'RE ACCEPTING DIALS)
    - OR WILL BE A PHONE NUMBER (IF WE'RE DIALING OUT)
  - dial_channel
    - AN EVENT-WAIT CHANNEL_ID RETURNED BY ipc_$create_ev_chn
    - MUST BE THE SAME FOR ALL CALLS TO dial_manager_ IN THIS PROCESS
  - channel_name
    - IDENTIFIES A LINE ADAPTER AND PORT NUMBER ON THE FRONT END PROCESSOR
    - SEE APPENDIX C

DIALING TERMINALS TO A PROCESS

- STEPS INVOLVED IN DIALING TERMINALS TO A PROCESS

  - PROCESS DECLARES AN EVENT-WAIT CHANNEL TO BE USED BY THE ANSWERING SERVICE TO NOTIFY THE PROCESS OF CRITICAL EVENTS (SUCH AS SOMEONE DIALING IN TO THE PROCESS, SOMEONE HANGING UP, AND SO ON)

  - PROCESS REQUESTS THAT THE ANSWERING SERVICE NOW ALLOW DIALS FOR THE PROCESS, PASSING THE EVENT-WAIT CHANNEL_ID AND A "DIAL QUALIFIER"

  - PROCESS MAY NOW CONVERT WAIT-CHANNEL TO CALL CHANNEL, IF DESIRED

  - TERMINALS ARE DIALED INTO THE PROCESS USING THE 'dial' COMMAND

  - UPON NOTIFICATION FROM THE ANSWERING SERVICE THAT A TERMINAL HAS DIALED-IN, PROCESS MUST INTERPRET THE IPC MESSAGE PASSED, WHICH CONTAINS

    - CHANNEL-NAME (DEVICE ID) OF THE TERMINAL DIALED-IN

    - FLAGS INDICATING WHAT TOOK PLACE ON THE COMLINE (DIALUP, HANGUP)

  - PROCESS NOW ATTACHES THAT DEVICE AND COMMENCES TO DO LOGICAL I/O TO THAT TERMINAL

DIALING TERMINALS TO A PROCESS

SUBROUTINES

- dial_manager_$allow_dials

  - REQUESTS THAT THE ANSWERING SERVICE ALLOW TERMINALS TO DIAL TO THE CALLING PROCESS

    - THE CALLER SETS 'dial_qualifier' IN THE dial_manager_arg STRUCTURE TO AN ALPHANUMERIC STRING FROM 1 TO 22 CHARACTERS

    - THE CALLER SETS dial_manager_arg.dial_channel TO THE EVENT-WAIT CHANNEL_ID ESTABLISHED FOR COMMUNICATING WITH THE ANSWERING SERVICE (NOTE THAT FOLLOWING A CALL TO dial_manager_$allow_dials, THE CALLER MAY CHANGE THE EVENT-WAIT CHANNEL INTO AN EVENT-CALL CHANNEL IF DESIRED)

- dial_manager_$registered_server

  - SIMILAR TO dial_manager_$allow_dials

  - PERMITS TERMINALS TO DIAL IN WITHOUT FURNISHING Personid.Projectid AS dial COMMAND ARGUMENT

  - dial_qualifier MUST BE REGISTERED BY SYSTEM ADMINISTRATOR

  - CALLER MUST HAVE rw ON >sc1>rcp>dial.<dial_qualifier>.acs

DIALING TERMINALS TO A PROCESS

SUBROUTINES

- dial_manager_$shutoff_dials
  - INFORMS THE ANSWERING SERVICE THAT THE PROCESS WISHES TO PREVENT FURTHER DIAL CONNECTIONS, AND THAT EXISTING CONNECTIONS SHOULD BE TERMINATED
  - ACCEPTS SAME INFORMATION AS dial_manager_$allow_dials
  - IMPORTANT RESTRICTION: dial_channel MUST BE AN EVENT-WAIT; CALLER MAY THEREFORE HAVE TO CALL ipc_$decl_ev_wait_chn FIRST

- convert_dial_message_$return_io_module
  - SHOULD BE INVOKED BY A PROCESS WHEN IT HAS RECEIVED A WAKEUP FROM THE ANSWERING SERVICE
  - REQUIRES THE IPC event_info.message AS INPUT AND RETURNS:
    - THE DEVICE-ID (CHANNEL NAME) OF THE COMMUNICATIONS LINE THAT HAS DIALED-UP OR HUNG-UP
    - A STRUCTURE INDICATING WHETHER THE TERMINAL IN QUESTION HAS DIALED-UP, OR HUNG-UP

DIALING TERMINALS TO A PROCESS

THE 'DIAL' COMMAND

- THE 'dial' COMMAND:
  - IS TYPED IN LIEU OF THE login COMMAND
  - IS A REQUEST TO THE ANSWERING SERVICE TO CONNECT THE TERMINAL TO AN EXISTING PROCESS AND TO NOTIFY THAT PROCESS OF THE CONNECTION
  - USAGE:

    dial dial_id {Person_id.Project_id}

  - THE USER MUST SPECIFY THE 'dial_id' THAT WAS PASSED TO THE ANSWERING SERVICE BY THE PROCESS ACCEPTING DIALS
  - THE USER MUST ALSO SPECIFY THE 'Person_id.Project_id' OF THE EXISTING PROCESS, UNLESS dial_manager_$registered_server WAS ORIGINALLY USED TO ALLOW DIALS

DIALING TERMINALS TO A PROCESS

AN EXAMPLE

```
SET_UP_DIAL:        proc;
dcl  ipc_$create_ev_chn entry (fixed bin (71), fixed bin (35)),
     ipc_$delete_ev_chn entry (fixed bin (71), fixed bin (35)),
     ipc_$decl_ev_call_chn entry (fixed bin (71), entry,
     ptr, fixed bin, fixed bin (35)),
     dial_manager_$allow_dials entry (ptr, fixed bin (35)),
    (ioa_, com_err_, ioa_$ioa_switch) entry options (variable);
dcl  code fixed bin (35),
     ME char (12) varying init ("SET_UP_DIAL") ;
dcl 1 dial_manager_arg aligned static,
    2 version fixed bin init (1),
    2 dial_qualifier char (22) init ("astra"),
    2 dial_channel fixed bin (71),
    2 channel_name char (32) ;

          call ioa_ ("Begin ^a", ME);
          call ipc_$create_ev_chn (dial_manager_arg.dial_channel,
               code);
          if code ^= 0 then call ERROR (1);
          call dial_manager_$allow_dials (addr (dial_manager_arg),
               code);
          if code ^= 0 then call ERROR (2);
          call ipc_$decl_ev_call_chn (dial_manager_arg.dial_channel,
              DIAL_HANDLER, null(), 0, code);
          if code ^= 0 then call ERROR (3);

          call ioa_ ("Now listening for dials: ^a", ME);
          return;
```

DIALING TERMINALS TO A PROCESS

AN EXAMPLE

```
DIAL_HANDLER: entry (info_ptr);
dcl  info_ptr ptr parameter;
dcl 1 event_info based (info_ptr),
    2 channel_id fixed bin (71),
    2 message fixed bin (71),
    2 sender bit (36),
    2 origin,
      3 dev_signal bit (18) unal,
      3 ring bit (18) unal,
    2 data_ptr ptr;

dcl convert_dial_message_$return_io_module entry (fixed bin(71),
    char(*), char(*), fixed bin, 1 aligned,       2 bit(1) unal,
    2 bit(1) unal, 2 bit(1) unal, 2 bit(33) unal, fixed bin(35));
dcl which_channel char (32);
dcl iocb_ptr ptr;

dcl ipc_$cutoff entry (fixed bin(71), fixed bin(35));
dcl ipc_$reconnect entry (fixed bin(71), fixed bin(35));

dcl iox_$attach_name entry (char(*), ptr, char(*), ptr, fixed bin(35));
dcl iox_$open entry (ptr, fixed bin, bit(1) aligned, fixed bin(35));
dcl iox_$close entry (ptr, fixed bin(35));
dcl iox_$detach_iocb entry (ptr, fixed bin(35));
dcl iox_$control entry (ptr, char(*), ptr, fixed bin(35));
dcl iox_$get_line entry (ptr, ptr, fixed bin(21), fixed bin(21),
                         fixed bin(35));

dcl buffer char(80);
dcl actually_read char(n_read) based (addr(buffer));
dcl n_read fixed bin (21);

          ME = "DIAL_HANDLER";

          call ipc_$cutoff (dial_manager_arg.dial_channel, code);
          if code ^= 0 then call ERROR (4);

          call convert_dial_message_$return_io_module (
               event_info.message, which_channel, "", 0, "0"b, code);
          if code ^= 0 then call ERROR (5);
```

DIALING TERMINALS TO A PROCESS

AN EXAMPLE

```
        call iox_$attach_name ("switch", iocb_ptr,
                               "tty_ "||which_channel, null(),code);
        if code ^= 0 then call ERROR (6);

        call iox_$open (iocb_ptr, 3, "0"b, code);
        if code ^= 0 then call ERROR (7);

        call ioa_$ioa_switch (iocb_ptr, "Welcome to my world.
Please type a line and I will echo it back.");
        call iox_$get_line (iocb_ptr, addr(buffer), 80, n_read, code);
        call ioa_$ioa_switch (iocb_ptr, "^a", actually_read);
        call ioa_$ioa_switch (iocb_ptr, "Good bye");

        call iox_$control (iocb_ptr, "hangup", null(), code);
        if code ^= 0 then call ERROR (8);

        call iox_$close (iocb_ptr, code);
        if code ^= 0 then call ERROR (9);

        call iox_$detach_iocb (iocb_ptr, code);
        if code ^= 0 then call ERROR (10);
        return;
```

```
ERROR:    proc (error_number);
          /* Internal proc to report errors */
dcl  error_number;
          call com_err_ (code, ME, "Check call ^i of ERROR",
                         error_number);
          goto FINISH;
     end ERROR;

SHUTOFF:  entry;

dcl  dial_manager_$shutoff_dials entry (ptr, fixed bin (35));
dcl  ipc_$decl_ev_wait_chn entry (fixed bin (71), fixed bin (35));

          ME = "SHUTOFF";

          call ipc_$decl_ev_wait_chn (dial_manager_arg.dial_channel, code);
          if code ^= 0 then call ERROR (11);
          call dial_manager_$shutoff_dials (addr (dial_manager_arg), code);
          if code ^= 0 then call ERROR (12);
          call ipc_$delete_ev_chn (dial_manager_arg.dial_channel, code);
          if code ^= 0 then call ERROR (13);
          return;

FINISH:
     end SET_UP_DIAL;
```

DIALING TERMINALS TO A PROCESS

AN EXAMPLE

- THE PRECEDING EXAMPLE IS VERY SIMPLE AND THEREFORE HAS LITTLE PRACTICAL APPLICATION

  - OBVIOUS PROBLEMS:

    1. IT ONLY HANDLES ONE DIALED IN TERMINAL AT A TIME
    2. IT CANNOT HANDLE THE SITUATION IN WHICH THE DIALED IN TERMINAL SIMPLY "HANGS UP"
    3. IT CANNOT HANDLE THE SITUATION IN WHICH THE DIALED IN TERMINAL SIGNALS QUIT (USER HITS BREAK KEY)
    4. THE MASTER PROCESS GOES BLOCKED WHILE WAITING FOR INPUT FROM THE "SLAVE" TERMINAL

- A 10 PAGE EXAMPLE APPEARS IN APPENDIX G

  - THIS EXAMPLE SOLVES ALL OF THE ABOVE PROBLEMS

  - IT PROVIDES A STARTING POINT FOR A REALISTIC DIAL IN APPLICATION

DIALING OUT TO A TERMINAL

- STEPS INVOLVED IN DIALING OUT
  - ESTABLISH EVENT-WAIT CHANNEL
  - REQUEST ANSWERING SERVICE TO DIAL A SPECIFIED PHONE NUMBER
  - ONE CAN CHANGE EVENT-WAIT TO EVENT-CALL CHANNEL AT THIS TIME
  - AFTER NOTIFICATION OF SUCCESSFUL DIAL-OUT, USER ATTACHES DEVICE AND DOES LOGICAL I/O
- A PRIVILEGED PROCESS (HAVING 'rw' ON THE APPROPRIATE ACCESS CONTROL SEGMENT) MAY USE THE DIAL FACILITY TO DIAL OUT TO A TERMINAL

DIALING OUT TO A TERMINAL

DIAL MANAGER ENTRY POINTS

- ENTRY POINTS USED TO DIAL OUT:

  - dial_manager_$dial_out

    - REQUESTS THAT AN AUTO-CALL CHANNEL BE DIALED TO A GIVEN TELEPHONE NUMBER, AND, IF THE CHANNEL IS SUCCESSFULLY DIALED, THAT THE CHANNEL BE ASSIGNED TO THE REQUESTING PROCESS

      - THE CALLER SETS dial_manager_arg.dial_qualifier TO THE TELEPHONE NUMBER TO BE DIALED (NONNUMERIC CHARACTERS IN THE NUMBER ARE IGNORED, SO THE NUMBER MAY BE SPECIFIED AS, FOR INSTANCE, "301/977-4292")

      - dial_manager_arg.dial_channel IS SET TO THE EVENT-WAIT CHANNEL CREATED TO ALLOW THE ANSWERING SERVICE TO COMMUNICATE WITH THE PROCESS

      - THE CALLER MAY SET dial_manager_arg.channel_name TO A SPECIFIC CHANNEL-NAME OF AN AUTO-CALL CHANNEL - IF THE CALLER ASSIGNS THE NULL STRING TO THIS ARGUMENT, THE ANSWERING SERVICE WILL ATTEMPT TO ASSIGN ANY AVAILABLE AUTO-CALL CHANNEL AND CALLER MUST USE convert_dial_message_$return_io_module TO DETERMINE WHAT IT IS

  - dial_manager_$terminate_dial_out

    - REQUESTS THAT THE ANSWERING SERVICE HANG UP AN AUTO-CALL LINE AND UNASSIGN IT FROM THE REQUESTING PROCESS

      - ACCEPTS THE SAME INFO AS dial_manager_$dial_out

      - HOWEVER, THE 'channel_name' ARGUMENT MUST BE SUPPLIED WITH THE NAME OF THE AUTO-CALL CHANNEL WHICH WAS USED FOR THE DIAL OUT

TOPIC XV

Message Segment Facility

	Page
What Is It?	15-1
Applications	15-2
The Message Segment	15-3
Layered Design	15-6
Primitive Message Segment Facility	15-8
Extended Access	15-9
message_segment_ Subroutine Summary	15-10
Message_Segment_Facility Illustrative Example	15-12

WHAT IS IT?

- A SERIES OF PRIMITIVES, SUBROUTINES AND COMMANDS

- DESIGNED TO

  - MANIPULATE RING 1 MESSAGE SEGMENTS

  - FACILITATE PROTECTED AND ORDERED MESSAGE EXCHANGE BETWEEN AND WITHIN PROCESSES

  - SALVAGE MESSAGE SEGMENTS CONTAINING "DAMAGED" MESSAGES

  - MINIMIZE WRITE/UPDATE WINDOW TIME

  - SUPPORT CHANGE-ABLE MESSAGE SIZE

APPLICATIONS

- I/O AND ABSENTEE DAEMON QUEUES

- SUPPORT FOR MAIL AND SEND_MESSAGE FACILITIES

- USER-DESIGNED APPLICATIONS REQUIRING THE SPECIAL CAPABILITIES OF MESSAGE SEGMENTS

THE MESSAGE SEGMENT

- PROPERTIES
  - ACCESSIBLE ONLY IN RING 1 (AND RING 0)
  - HAS A SUFFIX
    - 'ms' FOR QUEUE MESSAGE SEGMENTS
    - 'mbx' FOR MAILBOXES
  - MUST BE A SINGLE-SEGMENT FILE
  - HAS AN EXTENDED ACCESS CONTROL LIST

THE MESSAGE SEGMENT

- STRUCTURE
  - HEADER
    - LOCK WORD
    - 36-BIT MESSAGE SEGMENT ID BIT PATTERN
    - OFFSET TO FIRST MESSAGE
    - OFFSET TO LAST MESSAGE
    - MESSAGE COUNT
    - SWITCHES
      - MSEG INCONSISTENT
      - MSEG HAS BEEN SALVAGED
    - ALLOCATION BIT STRING SAYS WHICH BLOCKS ARE USED
    - LENGTH (ALLOCATION BIT STRING)
    - MESSAGE BLOCK SIZE
    - UNUSED BLOCK COUNT
    - ...AND OTHER INFO

THE MESSAGE SEGMENT

- DOUBLY-THREADED LIST OF MESSAGES
  - EACH MESSAGE IS COMPRISED OF 1 OR MORE FIXED-LENGTH BLOCKS
    - EACH BLOCK HAS A HEADER CONTAINING
      - OFFSET TO NEXT BLOCK IN MESSAGE (OR ZERO)
      - A "FIRST-BLOCK" SWITCH
      - NUMBER OF MESSAGE BITS IN BLOCK
    - FIRST BLOCK IN MESSAGE ALSO HAS A TRAILER
  - EACH MESSAGE TRAILER CONTAINS
    - MESSAGE BIT SIZE
    - TIME MESSAGE WAS SENT
    - VALIDATION LEVEL OF SENDER
    - Personid.Projectid OF SENDER

LAYERED DESIGN

- TWO HIGH-LEVEL SUBROUTINE INTERFACES ALREADY EXIST FOR MANIPULATION OF MESSAGE SEGMENTS
  - message_segment_ FOR QUEUE MESSAGE SEGMENTS
  - mailbox_ FOR MAILBOXES
- TWO CORRESPONDING COMMAND-SETS EXIST AS WELL
  - FOR QUEUE MSEGS WE HAVE:

    ms_add_name, msan

    ms_create, mscr

    ms_delete, msdl

    ms_delete_acl, msda

    ms_delete_name, msdn

    ms_list_acl, msla

    ms_rename, msrn

    ms_set_acl, mssa

- FOR MAILBOXES WE HAVE:

  mbx_add_name, mban

  mbx_create, mbcr

  mbx_delete, mbdl

  mbx_delete_acl, mbda

  mbx_delete_name, mbdn

  mbx_list_acl, mbla

  mbx_rename, mbrn

  mbx_set_acl, mbsa

- message_segment_ AND mailbox_ ARE GATES INTO THE ADMINISTRATIVE RING

  - WHICH TRANSFER CONTROL TO THE PROCEDURES queue_mseg_ AND mbx_mseg_, RESPECTIVELY

  - queue_msg_ AND mbx_mseg_ IN TURN CALL MODULES IN THE PRIMITIVE MESSAGE SEGMENT FACILITY

LAYERED DESIGN

PRIMITIVE MESSAGE SEGMENT FACILITY

- THE PRIMITIVE MESSAGE FACILITY IS COMPRISED OF MODULES WHICH
  - CREATE AND DELETE MSEGS
  - MANIPULATE EXTENDED ACCESS
  - LOCK AND UNLOCK MSEGS
  - MANIPULATE MESSAGES
  - MANIPULATE 'OWN' MESSAGES
  - SALVAGE MSEGS
  - CONVERT MSEGS FROM A PREVIOUS FORMAT

- BOTH QUEUE AND MAILBOX MSEGS EMPLOY THESE ATTRIBUTES:

  - a

    ALLOWS USER TO ADD A MESSAGE

  - d

    ALLOWS USER TO DELETE ANY MESSAGE

  - r

    ALLOWS USER TO READ ANY MESSAGE

  - o

    ALLOWS USER TO READ/DELETE 'OWN' MESSAGES

  - s

    ALLOWS USER TO DETERMINE WHETHER MSEG HAS BEEN SALVAGED AND MESSAGE COUNT

- IN ADDITION, MAILBOX MESSAGE SEGMENTS EMPLOY:

  - w

    ALLOWS USER TO SEND NORMAL WAKEUP WHEN ADDING MESSAGE

MESSAGE SEGMENT SUBROUTINE SUMMARY

- CREATING AND DELETING QUEUE MESSAGE SEGMENTS

  message_segment_$create

  message_segment_$delete

- MANIPULATING EXTENDED ACCESS

  message_segment_$ms_acl_add

  message_segment_$ms_acl_delete

  message_segment_$ms_acl_list

  message_segment_$ms_acl_replace

- RENAMING

  message_segment_$chname_file

- OPENING AND CLOSING

  message_segment_$open

  message_segment_$close

MESSAGE SEGMENT SUBROUTINE SUMMARY

- OBTAINING HEADER STATUS INFO

```
message_segment_$check_salv_bit_index
message_segment_$check_salv_bit_file

message_segment_$get_message_count_index
message_segment_$get_message_count_file
```

- OBTAINING EFFECTIVE ACCESS

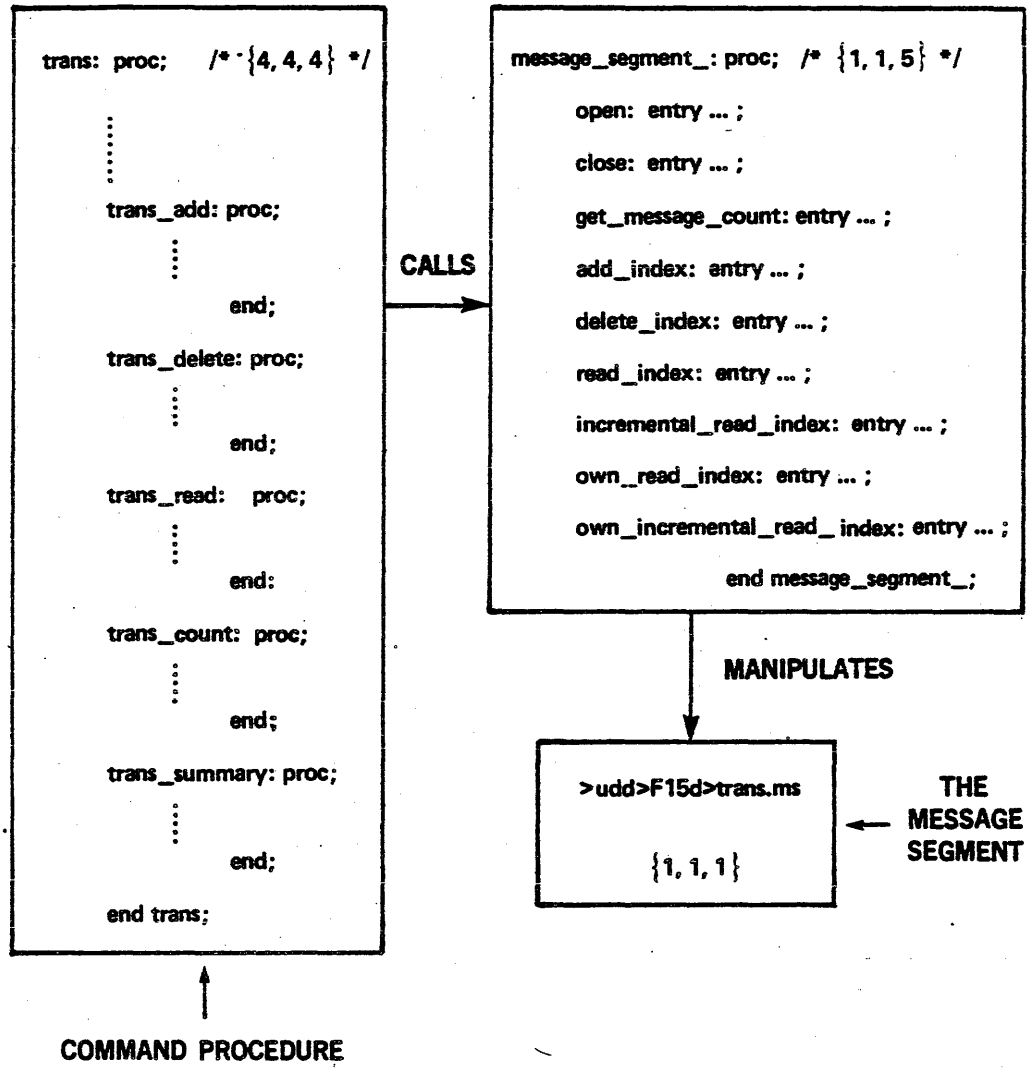
```
message_segment_$get_mode_index
message_segment_$get_mode_file
```

- MANIPULATING MESSAGES

```
message_segment_$add_index
message_segment_$add_file

message_segment_$delete_index
message_segment_$delete_file

message_segment_$read_index
message_segment_$read_file

message_segment_$incremental_read_index
message_segment_$incremental_read_file

message_segment_$update_message_index
message_segment_$update_message_file
```

- MANIPULATING 'OWN' MESSAGES

```
message_segment_$own_read_index
message_segment_$own_read_file

message_segment_$own_incremental_read_index
message_segment_$own_incremental_read_file
```



MESSAGE SEGMENT FACILITY ILLUSTRATIVE EXAMPLE

TOPIC XVI

Program Library Management

	Page
Introduction	16-1
Organization of Program Libraries	16-2
Naming Conventions	16-4
A Typical Program Library	16-6
Program Library Management Tools	16-9
Installation Tools	16-10
Library Descriptor Tools	16-20

INTRODUCTION

- LARGE AND COMPLEX SUBSYSTEMS REQUIRE GOOD PROGRAM LIBRARY MANAGEMENT TECHNIQUES

  - THE DESIGNER MUST BE CONCERNED WITH PROPERLY ORGANIZING THE SOURCE PROGRAMS, OBJECT PROGRAMS, BOUND SEGMENTS, LISTINGS, AND SO ON

  - THE DESIGNER COULD DEVELOP HIS OWN LIBRARY CONVENTIONS AND TOOLS, BUT:

- CONVENTIONS AND SYSTEM-PROVIDED TOOLS EXIST FOR

  - ORGANIZING SOURCE, OBJECT, EXECUTABLE, AND DOCUMENTATION LIBRARIES IN A CONVENIENT MANNER

  - MANIPULATING THE COMPONENTS OF A USER MAINTAINED LIBRARY IN A CONTROLLED MANNER

  - CONTROLLING THE INSTALLATION AND DE-INSTALLATION OF SUBSYSTEM MODULES IN AN ORDERLY MANNER

ORGANIZATION OF PROGRAM LIBRARIES

- A USER'S PROGRAM LIBRARY FOR A GIVEN SUBSYSTEM IS GENERALLY ORGANIZED AS A DIRECTORY SUBTREE

  - A DIRECTORY SEGMENT, NAMED FOR THE LIBRARY ITSELF, SERVES AS THE ROOT OF THE SUBTREE

  - EXECUTABLE PROGRAMS, WHETHER STAND-ALONE OR BOUND, RESIDE UNDER THE LIBRARY ROOT DIRECTORY

  - SUBDIRECTORIES UNDER THIS ROOT CONTAIN:

    - SOURCE PROGRAMS, EITHER INDIVIDUALLY OR IN ARCHIVE SEGMENTS

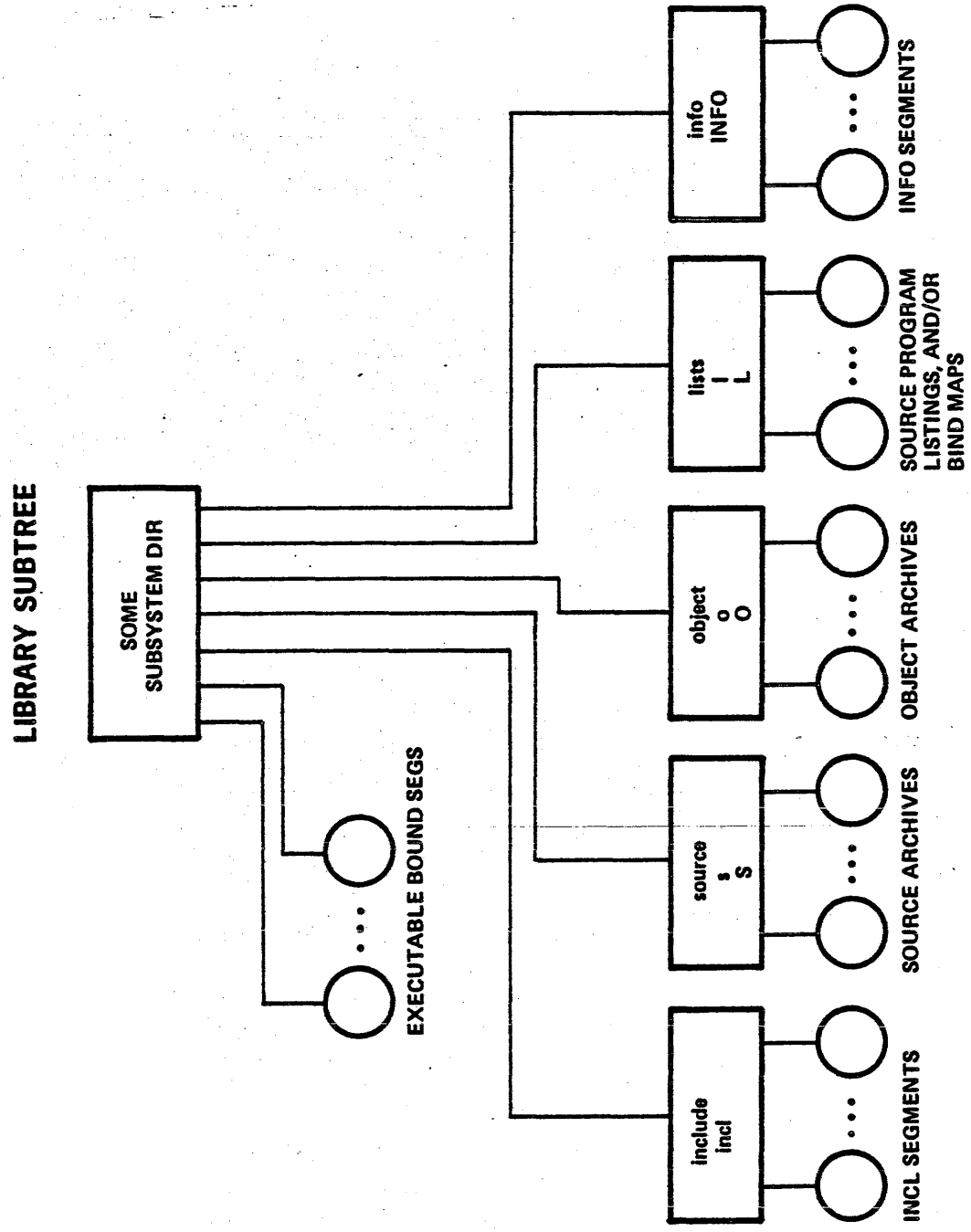
    - OBJECT PROGRAMS, EITHER INDIVIDUALLY, OR (MORE GENERALLY) IN ARCHIVE SEGMENTS

    - LISTINGS AND/OR BIND MAPS

    - INCLUDE FILES

    - HELP FILES

ORGANIZATION OF PROGRAM LIBRARIES



NAMING CONVENTIONS

- THE LIBRARY COMPONENTS MENTIONED ABOVE ARE GENERALLY NAMED ACCORDING TO THE FOLLOWING STANDARD NAMING CONVENTIONS:

  - THE SOURCE SUBDIRECTORY IS GENERALLY GIVEN THE NAMES

    source
    s
    S

  - THE OBJECT PROGRAM SUBDIRECTORY IS GENERALLY GIVEN THE NAMES

    object
    o
    O

  - THE LISTINGS SUBDIRECTORY IS GENERALLY GIVEN THE NAMES

    lists
    l
    L

  - THE INCLUDE FILE SUBDIRECTORY IS GENERALLY GIVEN THE NAMES

    include
    incl

  - THE HELP FILES SUBDIRECTORY IS GENERALLY GIVEN THE NAMES

    info
    INFO

- NAMING CONVENTIONS FOR BOUND SEGMENTS AND CORRESPONDING ARCHIVES

  - THE BOUND SEGMENT ITSELF IS GIVEN AN ENTRYNAME "bound_???????", WHERE ??????? IS A NAME CHOSEN BY THE DESIGNER (E.G., bound_command_loop_)

  - THE ARCHIVE WHICH CONTAINS THE SOURCE PROGRAMS USED TO GENERATE THE INDIVIDUAL COMPONENTS OF THE BOUND SEGMENT IS NAMED "bound_???????.s.archive" (E.G., bound_command_loop_.s.archive)

  - THE ARCHIVE WHICH CONTAINS THE OBJECT PROGRAMS AND WHICH WAS INPUT TO THE BINDER IS NAMED "bound_???????.archive" (E.G., bound_command_loop_.archive)

A TYPICAL PROGRAM LIBRARY

```
!list >udd>F15dw>Auerbach>user_library_1 -all

Directories = 1.

sma  user_library_1
       ul1

!list -pn >udd>F15dw>Auerbach>ul1 -all

Segments = 3, Lengths = 4.

re     2  bound_cde_
          c
          d
          e
re     1  b
re     1  a

Directories = 5.

sma  include
       incl
sma  info
       INFO
sma  lists
       l
       L
sma  object
       o
       O
sma  source
       s
       S

!cwd >udd>F15dw>Auerbach>ul1
```

A TYPICAL PROGRAM LIBRARY

```
!ls -pn source -all

Segments = 3, Lengths = 3.

r w     1  bound_cde_.s.archive
           c.pl1
           d.pl1
           e.pl1
r w     1  b.fortran
r w     1  a.pl1


!ls -pn object -all

Segments = 3, Lengths = 4.

r w     2  bound_cde_.archive
           c
           d
           e
           bound_cde_.bind
r w     1  b
r w     1  a


!ls -pn L -all

Segments = 6, Lengths = 6.

r w     1  bound_cde_.list
r w     1  a.list
r w     1  b.list
r w     1  c.list
r w     1  d.list
r w     1  e.list
```

```
!ls -pn INFO -all

Segments = 6, Lengths = 6.

r w     1  user_library_1.gi.info
r w     1  e.info
r w     1  d.info
r w     1  c.info
r w     1  b.info
r w     1  a.info


!ls -pn include -all

Segments = 3, Lengths = 3.

r w     1  DATABASE_STRUCTURE.incl.pl1
r w     1  REC2.incl.pl1
r w     1  REC1.incl.pl1
```

PROGRAM LIBRARY MANAGEMENT TOOLS

- LIBRARY ADMINISTRATOR TOOLS EXIST TO PROPERLY UPDATE AND MANIPULATE LIBRARIES IN A STRICTLY CONTROLLED AND CONSISTENT MANNER

- THE MAJOR TOOLS CAN BE CLASSIFIED AS FOLLOWS:

  - INSTALLATION TOOLS

  - PROGRAM LIBRARY MANIPULATION TOOLS

INSTALLATION TOOLS

- THE "INSTALLATION PROBLEM"
  - ARISES FROM ATTEMPTS TO DYNAMICALLY INSTALL A NEW OR REPLACEMENT VERSION OF A HEAVILY USED SUBSYSTEM MODULE (OR MODULES)
    - THOSE USERS CURRENTLY EXECUTING THE (NOW) OBSOLETE MODULES MUST CONTINUE TO EXECUTE THEM UNTIL THEY HAVE COMPLETED THEIR SESSION - IN ADDITION, ANY USERS WHO SUBSEQUENTLY ATTEMPT TO EXECUTE THE MODULE SHOULD RECEIVE THE NEW, UPDATED VERSION
    - ANY MODIFICATIONS TO THE PROGRAM LIBRARY SHOULD BE CAREFULLY DOCUMENTED OR LOGGED

INSTALLATION TOOLS

- SOLUTION TO PROBLEM: Multics INSTALLATION SYSTEM (MIS)
  - MIS SUBROUTINES ARE
    - RESTARTABLE ACROSS A SYSTEM OR PROCESS FAILURE (AS LONG AS STORAGE SYSTEM IS INTACT)
    - REVERSIBLE, ALLOWING FOR "DE-INSTALLATION" IF TROUBLE ARISES MIDSTREAM
  - MIS FEATURES
    - PLANNED AUTOMATIC RECOVERY (VIA DE-INSTALL ENTRY POINTS) FOR ERRORS LIKE record_quota_overflow, namedup, entry_not_found
    - AUTOMATIC DOCUMENTATION OF AN INSTALLATION
    - A COMMAND INTERFACE: update_seg

- update_seg IS USED TO DEFINE THE CONTENTS OF A MODIFICATION, AND TO INSTALL OR DE-INSTALL THAT MODIFICATION IN A LIBRARY

  - A MODIFICATION IS A GROUP OF PHYSICALLY OR LOGICALLY RELATED SEGMENTS WHICH MUST BE INSTALLED IN A LIBRARY AT THE SAME TIME IN ORDER TO MAINTAIN LIBRARY CONSISTENCY AND INTEGRITY

    - SOURCE AND OBJECT ARE PHYSICALLY RELATED

    - OBJECT AND OBJECT ARE LOGICALLY RELATED

  - A MODIFICATION IS INSTALLED THUSLY:

    - THE INSTALLATION OF EACH SEGMENT IS DIVIDED INTO A SERIES OF STEPS (GETTING A UNIQUE ID, NAMES, AND ACL OF THE NEW AND OLD SEGMENTS, COPYING THE TARGET SEGMENT, ADDING TO AND DELETING FROM THE TARGET SEGMENT'S NAMES, FREEING NAMES ON THE OLD SEGMENT, ETC.)

    - ONE STEP AT A TIME IS PERFORMED FOR ALL SEGMENTS OF THE MODIFICATION BEFORE MOVING ON TO THE NEXT STEP

    - THE EXECUTABLE SEGMENTS ARE INSTALLED LAST, AS A GROUP, AFTER INSTALLING THE OTHER SEGMENTS IN THE MODIFICATION (SOURCE SEGMENTS, ARCHIVES, ETC.)

  - THE INSTALLATION WINDOW CAN BE REDUCED TO LESS THAN ONE MINUTE PER MODIFICATION, AND IS USUALLY ABOUT FIVE SECONDS

- OPERATIONS PERFORMED BY update_seg:

  - CREATING MODIFICATIONS

    initiate
    set_defaults
    print_defaults

  - DEFINING OPERATIONS TO BE PERFORMED DURING THE MODIFICATION

    add
    delete
    move
    replace

  - LISTING THE DEFINED MODIFICATION

    print
    list

  - INSTALLING/DE-INSTALLING THE MODIFICATION

    install
    de_install

  - CLEARING THE CURRENT MODIFICATION

    clear

- NONPRIVILEGED USERS OF update_seg SHOULD FIRST TYPE:

  initiate [wh hcs_] installation_tools_

  OTHERWISE ENTRY POINTS IN installation_tools_ WILL BE CALLED BY update_seg AND MOST USERS HAVE NULL ACCESS TO THIS SEGMENT

```
!list -first 3

 Segments = 74, Lengths = 62.

 r w     0  04/06/81.audit
 rew     1  test
 rew     1  test.pl1

!cwd junk

!list
 Directory empty: >user_dir_dir>MED>NDibble>junk

!cwd <

!us print_defaults

 Global defaults
      ring brackets:
          1,5,5
      ACL:
          re          *.*.*

!us initiate example -rb 4 4 4

!us print_defaults

 Defaults for  >user_dir_dir>MED>NDibble>example.io
      ring brackets:
          4,4,4
      ACL:
          re          *.*.*

 Global defaults
      ring brackets:
          1,5,5
      ACL:
          re          *.*.*

!list -first 4

 Segments = 75, Lengths = 126.

 r w    64  example.io
 r w     0  04/06/81.audit
 rew     1  test
 rew     1  test.pl1
```

```
!us add test.pl1 junk>==

!us move test junk>==

!us print

 Add       >user_dir_dir>MED>NDibble>test.pl1
 as        >user_dir_dir>MED>NDibble>junk>test.pl1
      Set ring brackets:
          4,4,4
      Access control list:
          re          *.*.*
      Names:
          test.pl1

 Move      >user_dir_dir>MED>NDibble>test
 to        >user_dir_dir>MED>NDibble>junk>test
      Access control list:
          rew         NDibble.MED.*
          rew         NDibble.*.*
          rw          *.SysDaemon.*
      Names:
          test

!list -first 2

 Segments = 75, Lengths = 126.

 r w   64  example.io
 r w    0  04/06/81.audit

!us list

!list -first 2

 Segments = 76, Lengths = 127.

 r w    1  example.il
 r w   64  example.io
```

```
!print example.il 1

 INSTALLATION OBJECT SEGMENT  >user_dir_dir>MED>NDibble>example.io

 Listed on:           04/06/81  0728.3 mst Mon
 Created by:          NDibble.MED.*
 Created with:        update_seg (MIS Version 1.5)
 Created on:          04/06/81  0727.1 mst Mon


 SUMMARY OF THE INSTALLATION:


 Add        >user_dir_dir>MED>NDibble>test.pl1
 as         >user_dir_dir>MED>NDibble>junk>test.pl1


 Move       >user_dir_dir>MED>NDibble>test
 to         >user_dir_dir>MED>NDibble>junk>test

 INSTALLATION OBJECT SEGMENT HAS NOT BEEN INSTALLED.


 A DESCRIPTION OF THE INSTALLATION FOLLOWS.


 INSTALLATION DESCRIPTION:


 Add        >user_dir_dir>MED>NDibble>test.pl1
 as         >user_dir_dir>MED>NDibble>junk>test.pl1
      Set ring brackets:
          4,4,4
      Access control list:
          re          *.*.*
      Names:
          test.pl1


 Move       >user_dir_dir>MED>NDibble>test
 to         >user_dir_dir>MED>NDibble>junk>test
      Access control list:
          rew         NDibble.MED.*
          rew         NDibble.*.*
          rw          *.SysDaemon.*
      Names:
          test
```

```
!us install
 Beginning installation of example.io
 Error:  Linkage error by upd_ring_task_$set|1000
 (>system_library_tools>bound_mis_)
 referencing installation_tools_|set_ring_brackets
 Incorrect access on entry.

!list -first 5

 Segments = 76, Lengths = 127.

 r w    1  example.il
 r w   64  example.io
 r w    0  04/06/81.audit
 rew    1  test
 rew    1  test.pl1

!cwd junk

!list

 Segments = 2, Lengths = 2.

 r w    1  !BBBJKzgHHgZMDK
 r w    1  !BBBJKzgHHFjDLd

!in [wh hcs_] installation_tools_

!us de_install
 update_seg: The lock was already locked by this process.
 Non-fatal error encountered while locking
     >user_dir_dir>MED>NDibble>example.io.
 update_seg will continue performing the de_install function.
 Non-special target segments deleted.
 De-installation complete.

!list
 Directory empty: >user_dir_dir>MED>NDibble>junk

!us install
 Beginning installation of example.io
 Installation complete.
```

```
!list

 Segments = 2, Lengths = 2.

 rew     1  test
 re      1  test.pl1

!cwd <

!list -first 5

 Segments = 76, Lengths = 127.

 r w     1  example.il
 r w    64  example.io
 r w     0  04/06/81.audit
 rew     1  test.1
            !BBBJKzgHHgZM11
 rew     1  test.pl1

!cob test.1 junk>test

 >user_dir_dir>MED>NDibble>test.1:  (segment 1)
 03/14/81  1121.1 mst Sat    PL/I

 >user_dir_dir>MED>NDibble>junk>test:  (segment 2)
 03/14/81  1121.1 mst Sat    PL/I

 The 2 segments match.

!us list

!dp example.il
```

LIBRARY DESCRIPTOR TOOLS

- THE 'library_descriptor' COMMANDS ARE A COLLECTION OF TOOLS ALLOWING THE SUBSYSTEM DESIGNER OR LIBRARY ADMINISTRATOR TO MANIPULATE LIBRARY STRUCTURES

  - ALL REFERENCE 'library_descriptors', WHICH ARE

    - SPECIAL SEGMENTS THAT

      - DESCRIBE THE STRUCTURE OF LIBRARIES IN THE HIERARCHY
      - LIST THOSE LIBRARY DESCRIPTOR COMMANDS WHICH MAY BE USED ON THE DESCRIBED LIBRARIES
      - NAME THE PROCEDURES WHICH "KNOW" HOW TO OPERATE ON THE DESCRIBED LIBRARIES

    - CREATED IN A TWO STEP OPERATION

      - ASCII DESCRIPTOR SOURCE SEGMENT IS TRANSLATED INTO AN alm SEGMENT BY library_descriptor_compiler COMMAND PROCEDURE
      - alm ASSEMBLER GENERATES BINARY LIBRARY DESCRIPTOR

  - THE COMMANDS ARE

    - library_fetch, lf

      - COPIES SPECIFIED ENTRIES FROM A LIBRARY DEFINED BY THE "CURRENT LIBRARY DESCRIPTOR" INTO THE USER'S WORKING DIRECTORY
      - ACCEPTS THE STAR CONVENTION
      - HAS SOME USEFUL OPTIONS

        - CAN TELL YOU WHERE MATCHING ENTRY WAS FOUND (-long)
        - CAN BE TOLD WHERE TO PUT FETCHED ENTRIES AND WHAT TO CALL THEM (-into)

- CAN COPY THE ENTIRE ARCHIVE CONTAINING THE MATCHING ENTRIES, AS OPPOSED TO JUST SOME OF THE ARCHIVE COMPONENTS (-container)
- CAN INDIVIDUALLY COPY ALL COMPONENTS OF ARCHIVES CONTAINING THE MATCHING ENTRIES (-components)

- library_print
  - SELECTS PRINTABLE ENTRIES FROM A LIBRARY DEFINED BY THE CURRENT LIBRARY DESCRIPTOR AND WRITES THEM TO A FILE SUITABLE FOR DPRINTING
  - DPRINT CONTAINS AN INDEX
  - ACCEPTS THE STAR CONVENTION
  - USEFUL OPTIONS
    - -container
    - -components
    - CAN PRINT CUSTOMIZED PAGE FOOTINGS (-footer) AND FIRST PAGE HEADING (-header)

- library_info, li
  - RETURNS TO THE TERMINAL STATUS INFORMATION ABOUT SPECIFIED ENTRIES IN LIBRARY DEFINED BY CURRENT LIBRARY DESCRIPTOR
  - ACCEPTS THE STAR CONVENTION
  - USEFUL OPTIONS
    - -container
    - -components

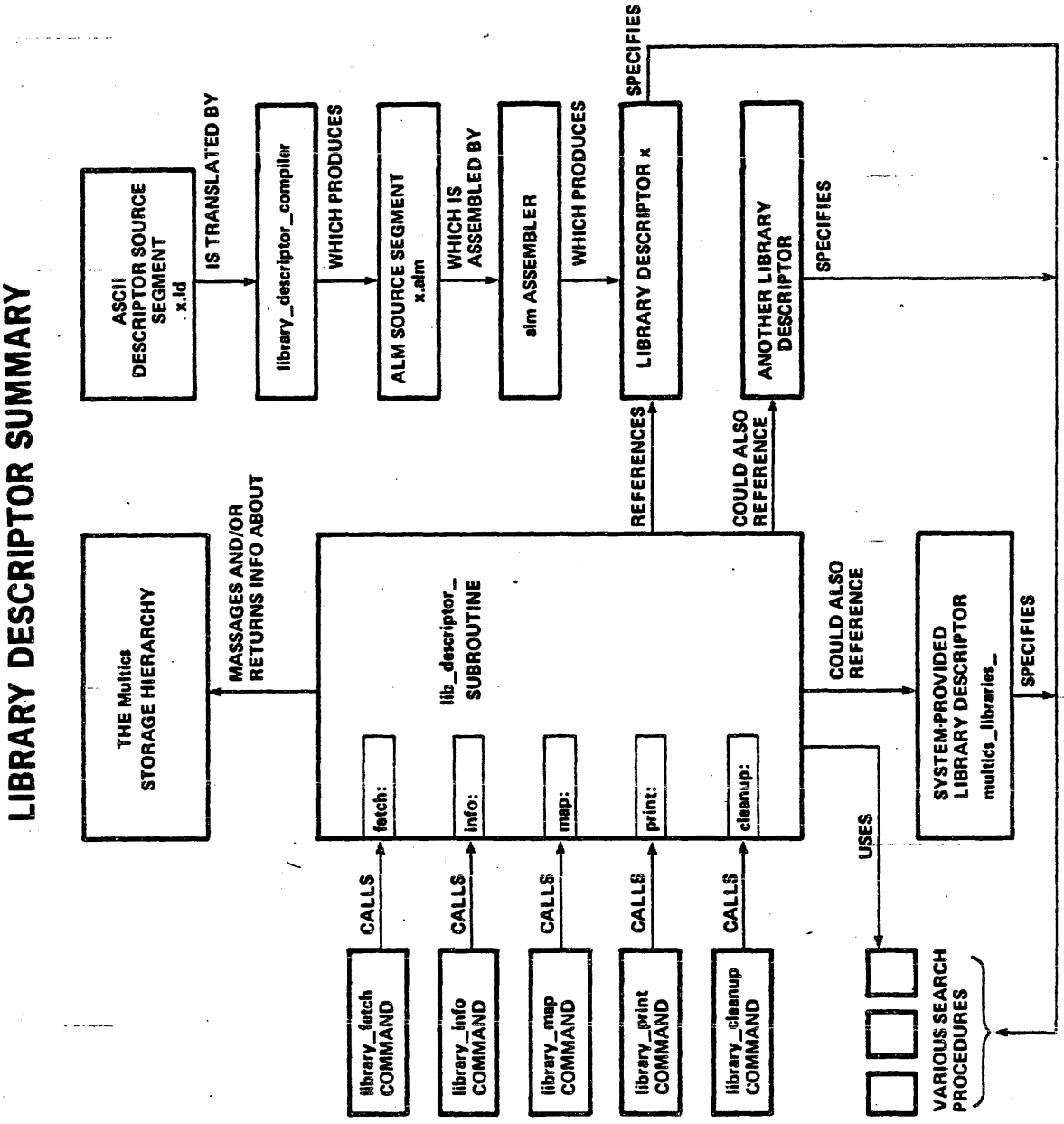
LIBRARY DESCRIPTOR TOOLS

- library_map
  - LIKE library_info, BUT GENERATES A MAP FILE SUITABLE FOR DPRINTING
  - USEFUL OPTIONS
    - -header
    - -footer
- library_cleanup, lcln
  - LISTS LIBRARY ENTRIES THAT HAVEN'T BEEN MODIFIED WITHIN THE SPECIFIED "GRACE" PERIOD
  - OPTIONALLY DELETES SUCH "OLD" SEGMENTS, LINKS, AND MULTISEGMENT FILES
  - ACCEPTS THE STAR CONVENTION
- library_descriptor, lds
  - PRINTS INFORMATION ABOUT LIBRARY DESCRIPTORS, AND CONTROLS USE OF LIBRARY DESCRIPTORS BY THE OTHER LIBRARY DESCRIPTOR COMMANDS
    - RETURNS NAME OF CURRENT LIBRARY DESCRIPTOR BEING USED
    - CHANGES CURRENT LIBRARY DESCRIPTOR

LIBRARY DESCRIPTOR TOOLS

- lib_descriptor_ SUBROUTINE
  - CONTAINS ENTRY POINTS THAT ARE CALLED BY ABOVE COMMANDS TO ACHIEVE THEIR GOALS
  - REFERENCES THE LIBRARY DESCRIPTORS

- WHY ALL THIS INDIRECTION?
  - AVOIDS REPLICATION OF COMMON CODE IN LIBRARY DESCRIPTOR COMMANDS
  - AVOIDS MODIFICATION OF MANY, SEPARATE COMMANDS WHEN
    - A NEW LIBRARY IS ADDED
    - A NEW LIBRARY ORGANIZATION IS INSTITUTED
  - PERMITS LIBRARY DESCRIPTOR COMMANDS TO WORK ON NON-SYSTEM LIBRARIES WITHOUT REWRITING THEM


LIBRARY DESCRIPTOR SUMMARY
ASCII DESCRIPTOR SOURCE SEGMENT x.ld
IS TRANSLATED BY
library_descriptor_compiler
WHICH PRODUCES
ALM SOURCE SEGMENT x.alm
WHICH IS ASSEMBLED BY
alm ASSEMBLER
WHICH PRODUCES
LIBRARY DESCRIPTOR x
SPECIFIES
ANOTHER LIBRARY DESCRIPTOR
SPECIFIES
REFERENCES
COULD ALSO REFERENCE
THE Multics STORAGE HIERARCHY
MASSAGES AND/OR RETURNS INFO ABOUT
lib_descriptor_ SUBROUTINE
fetch:
info:
map:
print:
cleanup:
COULD ALSO REFERENCE
SYSTEM-PROVIDED LIBRARY DESCRIPTOR
multics_libraries_
SPECIFIES
USES
CALLS
CALLS
CALLS
CALLS
CALLS
library_fetch COMMAND
library_info COMMAND
library_map COMMAND
library_print COMMAND
library_cleanup COMMAND
VARIOUS SEARCH PROCEDURES

```
!list -first 1

 Segments = 3, Lengths = 80.

 r w    1  handout_desc.ld

!pr handout_desc.ld 1

 Descriptor: handout_desc;

 Define: commands;
   command: library_print;
     library name:  handout;
     search names:  **;
   command: li;
     library name: handout;
     search names: **;

 Root: handout;
   path: >udd>F15D>s1>handout;
   search procedure:  multics_library_search_$hardcore_bc_dir;

 End:  handout_desc;

!ldc handout_desc

!list -first 2

 Segments = 4, Lengths = 82.

 r      2  handout_desc.alm
 r w    1  handout_desc.ld

!alm handout_desc.alm
 ALM
!list -first 3

 Segments = 5, Lengths = 83.

 re     1  handout_desc
 r      2  handout_desc.alm
 r w    1  handout_desc.ld
```

```
!li gw.archive
 library_info: Use of star convention resulted in no match.
   While searching for entries in the library.
   Descriptor:           multics_libraries_
   library name:          online_libraries_
   search name:           gw.archive

!lds set handout_desc

!li gw.archive

 1 gw.archive                                        type:  archive
                path:  >udd>F15D>s1>handout
   contents modified:  02/18/81  1648.4              system id:  34-32

!lpr gw.archive -components

!list -first 4

 Segments = 6, Lengths = 119.

 r w   36  library.print
 re     1  handout_desc
 r      2  handout_desc.alm
 r w    1  handout_desc.ld
```

APPENDIX A

AIM

Page

Concepts A-1
Commands and Subroutines A-3

- SOME BASIC TERMINOLOGY AND PROPERTIES SHOULD BE UNDERSTOOD:

  - AIM IS A NONDISCRETIONARY ACCESS CONTROL MECHANISM

  - SENSITIVITY (AS MANY AS 8)

  - CATEGORY SET (AS MANY AS 18)

  - ACCESS CLASS OF A SEGMENT IS COMPRISED OF A SENSITIVITY AND A CATEGORY SET

  - ACCESS AUTHORIZATION OF A PROCESS IS LIKEWISE COMPRISED OF A SENSITIVITY AND A CATEGORY SET

  - RELATIONSHIPS BETWEEN AUTHORIZATIONS AND ACCESS CLASSES

		SEGS	DIRS
AUTH	$\geq$ ACCESS CLASS	re	s
AUTH	= ACCESS CLASS	rew	sma
OTHERWISE		null	null

  - SYSTEMS "NOT RUNNING AIM" USE A SENSITIVITY OF "SYSTEM_LOW" WITH NO CATEGORIES

- DETERMINING THE "PROCESS MAXIMUM AUTHORIZATION"
  - TAKE THE MINIMUM OF THE FOLLOWING 3:
    - PERSON MAXIMUM AUTHORIZATION ON ANY PROJECT
    - PERSON MAXIMUM AUTHORIZATION ON THE GIVEN PROJECT
    - PROJECT MAXIMUM AUTHORIZATION

- A SEGMENT RECEIVES ITS ACCESS CLASS FROM THE CONTAINING DIR, NOT FROM THE ACCESS AUTHORIZATION OF THE CREATING PROCESS

- A DIRECTORY'S ACCESS CLASS DEFAULTS TO ITS CONTAINING DIRECTORY, BUT CAN BE "UPGRADED" UP TO THE PROCESS MAXIMUM AUTHORIZATION

- THE FOLLOWING COMMANDS AND SUBROUTINES DEAL WITH THE ACCESS ISOLATION MECHANISM:

  - print_auth_names (AG92)
    - THIS COMMAND PRINTS THE SHORT AND LONG NAMES OF THE AIM SENSITIVITIES AND CATEGORIES SET FOR THIS SITE

  - get_authorization_ (AG93)
    - RETURNS THE AUTHORIZATION VALUE FOR THE CALLING PROCESS AS 'bit(72)'

  - print_proc_auth (AG92)
    - THIS COMMAND RETURNS CHAR-STRING REPRESENTATION OF THE PROCESS' AUTHORIZATION

  - get_max_authorization_ (AG93)
    - RETURNS THE MAXIMUM AUTHORIZATION VALUE OF THE CALLING PROCESS AS THE 'bit(72)' VALUE

  - convert_authorization_ (AG93)
    - PROVIDES SEVERAL ENTRY POINTS FOR CONVERTING AN AUTHORIZATION BACK AND FORTH BETWEEN ITS BINARY AND ITS CHARACTER-STRING REPRESENTATION

- hcs_$get_access_class AG93)
  hcs_$get_access_class_seg (AG93)

  - THESE RETURN THE ACCESS CLASS OF A SEGMENT OR DIRECTORY GIVEN EITHER A DIRECTORY PATHNAME AND ENTRYNAME, OR GIVEN A POINTER TO THE SEGMENT ITSELF

- aim_check_ (AK92)

  - PROVIDES SEVERAL ENTRY POINTS WHICH ALLOW THE CALLER TO DETERMINE THE AIM RELATIONSHIP ("EQUAL", "GREATER", "GREATER-OR-EQUAL") BETWEEN TWO ACCESS ATTRIBUTES (AUTHORIZATION OR ACCESS CLASS)

- read_allowed_ (AK92)
  write_allowed_ (AK92)
  read_write_allowed_ (AK92)

  - DETERMINE WHETHER THE SUBJECT OF A SPECIFIED AUTHORIZATION HAS ACCESS TO READ, WRITE, OR READ-AND-WRITE AN OBJECT OF SPECIFIED ACCESS CLASS

- get_privileges_ (AK92)

  - THIS FUNCTION RETURNS THE ACCESS PRIVILEGES OF THE CALLING PROCESS (E.G., ipc ALLOWED, ETC.)

FOR A DISCUSSION OF AIM, SEE CHAPTER 6 OF THE MPM REFERENCE GUIDE.

APPENDIX B

Program Listings

	Page
user_init_admin_	B-1
user_real_init_admin_	B-3
process_overseer_	B-6
project_start_up_	B-10
listen_	B-15

```
"  ************************************************************
"  *                                                          *
"  *                                                          *
"  * Copyright (c) 1972 by Massachusetts Institute of         *
"  * Technology and Honeywell Information Systems, Inc.       *
"  *                                                          *
"  *                                                          *
"  ************************************************************

          name      user_init_admin_
          entry     user_init_admin_
          entry     daemon_init_admin_
          entry     absentee_init_admin_

          tempd     pit_ptr,po_ptr,arg(3)
          include   stack_header

user_init_admin_:
          eppab     <user_real_init_admin_>|[user_real_init_admin_]
          tra       join

daemon_init_admin_:
          eppab     <daemon_real_init_admin_>|[daemon_real_init_admin_]
          tra       join

absentee_init_admin_:
          eppab     <absentee_real_init_admin_>|[absentee_real_init_admin_]
          tra       join


join:     push
          eppbp     po_ptr              prepare argument list
          spribp    arg+2
          eppbp     pit_ptr
          spribp    arg+4
          fld       =2b24,dl
          staq      arg
"
"         call real_init_admin_ (po_ptr, pit_ptr)
"
          call      ab|0(arg)
"
"         call the process overseer ... always pass it the pit pointer
"
```

```
use_arglist:
          fld       =1b24,dl            set up argument list (1 arg)
          staq      sp|0
          eppbp     pit_ptr,*           retrieve the pit pointer
          spribp    sp|4
          eppbp     sp|4
          spribp    sp|2
          eppbp     po_ptr,*            save pointer to process overseer
          eppap     sp|0                get pointer to argument list
          short_call          bp|0
          eppbp     process_killing_pointer
"         get a pointer to -2|0 which will blow us away
          arg       0
          even
process_killing_pointer:
          its       -2,0

          end
```

```
/* ********************************************************
   *                                                      *
   *                                                      *
   * Copyright (c) 1972 by Massachusetts Institute of     *
   * Technology and Honeywell Information Systems, Inc.   *
   *                                                      *
   *                                                      *
   ******************************************************** */

/* This procedure is called by user_init_admin_ and is the second
   user ring program called in a newly created process.  It
   initializes the I/O system and returns a pointer to the
   process overseer to be called by user_init_admin_ after it
   returns. */

user_real_init_admin_: proc (process_overseer_ptr, pit_ptr);

dcl  process_overseer_ptr ptr; /* pointer returned to user_init_admin_ */
dcl  pit_ptr ptr; /* pointer to pit: returned non-null only for
                        standard case of process_overseer_ */


dcl  po_ptr ptr, /* points to process overseer for process */
    (type, string) char (32),
     status bit (72) aligned,
     code fixed bin (35),
     based_code fixed bin (35) based (addr (status)),
     pp ptr; /* points to PIT */


dcl (null, length, addr, substr, pointer) builtin;

dcl  terminate_process_ ext entry (char (*), ptr),
     sct_manager_$set entry (fixed bin, entry, fixed bin (35)),
     timer_manager_$alarm_interrupt entry,
     timer_manager_$cpu_time_interrupt entry,
     term_signal_handler_ entry,
     sus_signal_handler_ entry,
     change_wdir_ entry (char (168), fixed bin (35)),
     wkp_signal_handler_ entry,
     hcs_$terminate_noname entry (ptr, fixed bin (35)),
     hcs_$make_seg entry (char (*), char (*), char (*),
                    fixed bin (5), ptr, fixed bin (35)),
     iox_$attach_iocb entry (ptr, char (*), fixed bin (35)),
     iox_$open entry (ptr, fixed bin, bit (1) aligned, fixed bin (35)),
     iox_$user_io ptr ext,
     ios_$attach entry (char (*), char (*), char (*), char (*),
                    bit (72) aligned),
     find_command_$fc_no_message entry (ptr, fixed bin, ptr,
                    fixed bin (35)),
     ios_$ios_quick_init entry (),
     ioa_ entry options (variable);
```

```
%include pitmsg;
%include static_handlers;
%include iox_modes;


          call hcs_$make_seg ("", "pit", "", 01000b, pp, code);
              /* get pointer to PIT */
          type = pp -> pit.outer_module; /* Get DIM name */
          call ios_$ios_quick_init; /* initialize syn attachments */

  /* now do things that need doing before working dir exists */

          if ^pp -> pit.at.vinitproc then call find_po_and_dim;

          call change_wdir_ ((pp -> pit.homedir), code);
          /* put home dir into search path */
          /* ignore code-- if no wdir we do the best we can */

          if pp -> pit.at.vinitproc then call find_po_and_dim;


/* Now set up static handlers for "alrm", "cput", and "term" */

          call sct_manager_$set (cput_sct_index,
               timer_manager_$cpu_time_interrupt, code);
          call sct_manager_$set (alrm_sct_index,
               timer_manager_$alarm_interrupt, code);
          call sct_manager_$set (term_sct_index,
               term_signal_handler_, (0));
          call sct_manager_$set (wkp_sct_index,
               wkp_signal_handler_, code);
          call sct_manager_$set (susp_sct_index,
               sus_signal_handler_, (0));

          return;

find_po_and_dim:
          procedure ();

               if type = "tty_" then do;
                    call iox_$attach_iocb (iox_$user_io,
                         "tty_ -login_channel", code);
                    if code = 0 then go to open;
               end;
               call iox_$attach_iocb (iox_$user_io,
                    type || " " || pp -> pit.tty, code);
               /* attach primary input/output stream */
               if code = 0 then do;
open:               call iox_$open (iox_$user_io,
                         Stream_input_output, "0"b, code);
                    if code ^= 0 then call login_abort ("io_attach",
                                                        code);
               end;
```

```
               else do;
                    string = pp -> pit.tty;
                    call ios_$attach ("user_i/o", type,
                         string, "", status);
                    if based_code ^= 0 then call login_abort
                         ("io_attach", based_code);
               end;

               pit_ptr = pp;

               call find_command_$fc_no_message
                    (addr (pp -> pit.login_responder),
                    length (pp -> pit.login_responder), po_ptr, code);
               if code ^= 0 then do;
                    call ioa_ ("Could not find specified initial
\cprocedure: ^a", pp -> pit.login_responder);
                    call login_abort ("no_initproc", code);
               end;
               process_overseer_ptr = po_ptr;
               return;
          end /* find_po_and_dim */;

login_abort: proc (why, fatal_code);
 /* this procedure logs out the process with a special
 message indicating that an initialization error occurred */

dcl  why char (*); /* reason we can't go */
dcl  fatal_code fixed bin (35); /* code indicating fatal error */

dcl 1 term_structure aligned static,
    2 version fixed bin init (0), /* version of structure */
    2 status_code fixed bin (35); /* fatal error code */

               status_code = fatal_code;
               /* transmit code to terminate routine */
               call terminate_process_ (why, addr (term_structure));
               /* terminate the process */
          end login_abort;

     end user_real_init_admin_;
```

```
/* ********************************************************
   *                                                      *
   *                                                      *
   * Copyright (c) 1972 by Massachusetts Institute of     *
   * Technology and Honeywell Information Systems, Inc.   *
   *                                                      *
   *                                                      *
   ******************************************************** */

process_overseer_: proc (pit_ptr);

/* process_overseer_ is the standard process overseer on the system.
It has four responsibilities:

setting up an unclaimed signal handler, otherwise known as an any_other
handler.  This handler catched otherwise uncaught conditions.  The
supplied handler, default_error_handler_$wall prints any message provided
for the error condition, establishes a condition wall, and calls the
listener to get a new listener level.  A condition wall is just another
any_other handler; this intercepts conditions that might otherwise be
caught by other handlers present on the stack.

setting up a static handler for the mme2 condition.  The mme2 condition
is raised when the mme2 instruction is executed.  It is used by debug
to establish breakdots.  The handler transfers control to debug when
the condition is signalled.

allowing the ".." excape to command query.  This is enabled by calling
command_query_$set_cp_excape with the appropriate bits.

finding the start_up.ec.  It looks in the homedir, projectdir, and
finally >sc1 to try to find it.  It ends by calling listen_ with "ec
start_upName" as the initial command line.

The code is written for time rather than space efficiency, so that
operations that might look prettier in a do loop are done with inline
code.  */
```

```
/* Automatic */

dcl  initial_command_line char (104) var init ("");
dcl  pit_ptr ptr;
dcl  code fixed bin (35);
dcl  unaligned_homedir char (64) unaligned based
     (addr (pit_ptr -> pit.homedir));
dcl  bc fixed bin (24);
dcl  entry_type fixed bin (2);
dcl  first_process bit (1);

/* Constants */

dcl  process_type (0 : 3) character (12) varying internal
     static options (constant) initial ("initializer", "interactive",
     "absentee", "daemon");
dcl down_sc1 char (4) internal static options (constant) init (">sc1");
dcl start_up_dot_ec char (11) internal static options (constant)
    init ("start_up.ec");
/* Entries */

dcl  hcs_$terminate_noname entry (ptr, fixed bin (35));
dcl  condition_ entry (char (*), entry);
dcl  command_query_$set_cp_escape_enable entry (bit (1) aligned,
     bit (1) aligned);
dcl  listen_ ext entry (char (*) var);
dcl  default_error_handler_$wall entry;
dcl  hcs_$status_minf entry (char (*), char (*), fixed bin (1),
     fixed bin (2), fixed bin (24), fixed bin (35));
dcl  sct_manager_$set entry (fixed bin, ptr, fixed bin (35));
dcl  process_overseer_$mme2_fault_handler_ entry (ptr, char (*),
     ptr, ptr, bit (1));

/* External variables */

dcl  iox_$user_output ptr ext;

/* Builtins */

dcl (codeptr, divide, null, rtrim) builtin;

%include pitmsg;
%include static_handlers;

 /*set up the unclaimed signal handler */
          call condition_ ("any_other", default_error_handler_$wall);

 /* turn on ".." */
          call command_query_$set_cp_escape_enable ("1"b, (""b));

          first_process = (pit_ptr -> pit.n_processes = 1);
          /* see if new_proc or login */
```

```
          if ^ pit_ptr -> pit.at.nostartup then do;
                  /* start_up is allowed */
               initial_command_line = "exec_com ";

/* First try homedir */

               call hcs_$status_minf (unaligned_homedir,
               start_up_dot_ec, 1, entry_type, bc, code);

/* note that we assume any error is cause to look elsewhere to
   give best chance of success */

               if code = 0 & entry_type = 1 then
                    initial_command_line = initial_command_line ||
                    rtrim (pit_ptr -> pit.homedir);

/* now try projectdir */

               else do;
                    call hcs_$status_minf (">udd>" ||
                      rtrim (pit_ptr -> pit.project),
                      start_up_dot_ec, 1, entry_type, bc, code);

                    if code = 0 & entry_type = 1 then
                         initial_command_line = initial_command_line ||
                         ">udd>" || rtrim (pit_ptr -> pit.project);
                    else do;
                         call hcs_$status_minf (down_sc1,
                         start_up_dot_ec, 1, entry_type, bc, code);
                        if code = 0 & entry_type = 1 then
                             initial_command_line =
                             initial_command_line || down_sc1;
                         else do;
                             initial_command_line = "";
                             goto no_start_up;
                         end;
                    end;
               end;

               initial_command_line = initial_command_line || ">";
               initial_command_line = initial_command_line ||
                                      start_up_dot_ec;
/* the piecemeal assemble makes faster code */

               if first_process
                    then initial_command_line =
                         initial_command_line || " login ";
               else initial_command_line =
                    initial_command_line || " new_proc ";

               initial_command_line = initial_command_line
                 || process_type (pit_ptr -> pit.process_type);
          end; /* the block that checked pit.nostart*/
```

```
no_start_up:
          call hcs_$terminate_noname (pit_ptr, code);

/* set up the mme2 handler */
/* this is done here rather than in xxx_real_init_admin_ so that
   process overseers for limited subsystems can leave it out */

          call sct_manager_$set (mme2_sct_index,
          codeptr (process_overseer_$mme2_fault_handler_), code);


          do while ("1"b);
               call listen_ (initial_command_line);
          end;

          return;

mme2_fault_handler_: entry (mcp, cname, cop, infop, cont);
dcl (mcp ptr, /* to machine conditions */
     cname char (*), /* name of condition being signalled */
     cop ptr,
     infop ptr,
     cont bit (1)) parameter;

dcl  debug$mme2_fault entry (ptr);

          call debug$mme2_fault (mcp);
          cont = "0"b; /* do not continue searching for handlers */
          return;
     end process_overseer_;
```

```
/* ************************************************************
   *                                                          *
   *                                                          *
   * Copyright (c) 1972 by Massachusetts Institute of         *
   * Technology and Honeywell Information Systems, Inc.       *
   *                                                          *
   *                                                          *
   ************************************************************ */

project_start_up_:
     procedure (pit_ptr);

          dcl pit_ptr ptr;
          dcl initial_command_line char (256) varying;
          dcl listen_ entry (char (*) var);
          dcl terminate_process_ entry (char (*), ptr);
          dcl com_err_ entry () options (variable);
          dcl ioa_$ioa_switch entry options (variable);
          dcl any_other_handler entry variable;
          dcl any_other condition;
          dcl (length, null, unspec) builtin;

%include iox_dcls;

          any_other_handler = error_handler;
          on any_other call any_other_handler;
          /* Set up any_other handler outside the begin block */

          begin options (non_quick);

               dcl saved_cl_intermediary entry variable;
               dcl home_dir char (168);
               dcl project_dir char (168);
               dcl mme2_handler ptr;
               dcl saved_mme2_handler ptr;
               dcl ps_ec_cl character (256) aligned;
               dcl code fixed bin (35);
               dcl bc fixed bin (24);
               dcl entry_type fixed bin (2);
               dcl first_process bit (1);
               dcl (first_ec_arg, second_ec_arg) char (12);
               dcl wall_entry entry variable;

               dcl process_type (0:3) character (12)
                       internal static options (constant) initial
                 ("initializer", "interactive", "absentee", "daemon");
               dcl down_sc1 char (19) internal static
                   options (constant) init (">system_control_dir");
               dcl start_up_dot_ec char (11) internal static
                   options (constant) init ("start_up.ec");
```

```
          dcl cu_$cp entry (ptr, fixed bin (21), fixed bin (35));
          dcl convert_status_code_ entry (fixed bin (35),
              char (8) aligned, char (100) aligned);
          dcl change_wdir_ entry (char (168), fixed bin (35));
          dcl hcs_$make_entry entry (ptr, char (*), char (*),
              entry, fixed bin (35));
          dcl hcs_$terminate_noname  entry (ptr, fixed bin (35));
          dcl command_query_$set_cp_escape_enable entry
              (bit (1) aligned, bit (1) aligned);
          dcl default_error_handler_$wall entry;
          dcl hcs_$status_minf entry (char (*), char (*),
              fixed bin (1), fixed bin (2), fixed bin (24),
              fixed bin (35));
          dcl sct_manager_$set entry (fixed bin, ptr,
              fixed bin (35));
          dcl sct_manager_$get entry (fixed bin, ptr,
              fixed bin (35));
          dcl process_overseer_$mme2_fault_handler
              entry (ptr, char (*), ptr, ptr, bit (1));
          dcl cu_$set_cl_intermediary entry (entry);
          dcl cu_$get_cl_intermediary entry (entry);

          dcl command_error condition;

          dcl (addr, codeptr, length, null, rtrim, unspec)
                                              builtin;

%include pitmsg;
%include static_handlers;

          home_dir = pit_ptr -> pit.homedir;
          project_dir = ">user_dir_dir>" ||
               rtrim (pit_ptr -> pit.project);

          call sct_manager_$get (mme2_sct_index,
               saved_mme2_handler, (0));

          call hcs_$status_minf (project_dir,
               "project_start_up.ec", 1, entry_type, bc, code);
          if ^(entry_type = 1 & code = 0)
          then call abort_handler (rtrim (project_dir) ||
     ">project_start_up.ec was not found or is not a segment.",
                    code);

          call change_wdir_ (project_dir, code);
          if code ^= 0
          then call abort_handler
("Could not set working directory to project directory.", code);
```

```
          first_process = (pit_ptr -> pit.n_processes = 1);
          if first_process
          then first_ec_arg = "login";
          else first_ec_arg = "new_proc";

          second_ec_arg = process_type (pit_ptr -> pit.process_type);

          call hcs_$make_entry (null (), "default_error_handler_",
               "wall", wall_entry, code);
          if code ^= 0
          then wall_entry = default_error_handler_$wall;

          any_other_handler = wall_entry;

          call cu_$get_cl_intermediary (saved_cl_intermediary);
          call cu_$set_cl_intermediary (error_handler);

          on command_error call com_err_handler;
          /* die on com_err */

          ps_ec_cl = "exec_com " || rtrim (project_dir) ||
               ">project_start_up " || rtrim (first_ec_arg) ||
               " " || rtrim (second_ec_arg);
          call cu_$cp (addr (ps_ec_cl),
               length (rtrim (ps_ec_cl)), (0));

          revert command_error;

          call cu_$set_cl_intermediary (saved_cl_intermediary);

          call change_wdir_ (home_dir, code);
          if code ^= 0
          then call com_err_ (code, "project_start_up_",
"Could not set working directory to ^a.", home_dir);

          call command_query_$set_cp_escape_enable ("1"b, (""b));

          if ^pit_ptr -> pit.at.nostartup
          then do;
               initial_command_line = "exec_com ";
               call hcs_$status_minf (home_dir,
                    start_up_dot_ec, 1, entry_type, bc, code);
               if code = 0 & entry_type = 1
                    then initial_command_line =
                    initial_command_line || rtrim (home_dir);
```

```
                              else do;
                                   call hcs_$status_minf (project_dir,
                                   start_up_dot_ec, 1, entry_type, bc, code);

                                   if code = 0 & entry_type = 1
                                        then initial_command_line =
                                        initial_command_line || project_dir;
                                    else do;
                                         call hcs_$status_minf (down_sc1,
                                           start_up_dot_ec, 1, entry_type,
                                           bc, code);
                                         if code = 0 & entry_type = 1
                                            then initial_command_line =
                                            initial_command_line || down_sc1;
                                         else do;
                                              initial_command_line = "";
                                              goto no_start_up;
                                              end;
                                     end;
                              end;

          initial_command_line = initial_command_line || ">";
          initial_command_line = initial_command_line ||
               start_up_dot_ec;
          initial_command_line = initial_command_line || " " ||
               first_ec_arg;
          initial_command_line = initial_command_line || " " ||
               second_ec_arg;
          end;

no_start_up:
     call hcs_$terminate_noname (pit_ptr, code);
     call sct_manager_$get (mme2_sct_index, mme2_handler, (0));
     if mme2_handler = saved_mme2_handler
     then call sct_manager_$set (mme2_sct_index, codeptr
          (process_overseer_$mme2_fault_handler_), code);
     end;
call listen_ (initial_command_line);
do while ("1"b);
     call listen_ ("");
end;
return;

com_err_handler:
     procedure;
%include condition_info_header;
%include condition_info;
%include com_af_error_info;
          declare 1 CI aligned like condition_info;
          declare find_condition_info_ entry (pointer, pointer,
                  fixed binary (35));
          declare code fixed bin (35);
```

```
          unspec (CI) = ""b;

          call find_condition_info_ (null (), addr (CI), code);
          if code ^= 0
          then call abort_handler ("Can't get error message.", code);
          call ioa_$ioa_switch (iox_$error_output, "^a",
               CI.info_ptr -> com_af_error_info.info_string);
          call abort_handler ("Error in project start up.", 0);
     end;
error_handler:
     entry;

          call abort_handler ("Error in project start up.", 0);

abort_handler:
     proc (reason, code);

          dcl code fixed bin (35);
          dcl reason char (*);
          dcl 1 term_structure aligned,
                    2 version fixed bin init (0),
                    2 status_code fixed bin (35);

          status_code = code;
          call ioa_$ioa_switch (iox_$error_output, reason);
          call terminate_process_ ("init_error", addr (term_structure));
     end;

     end project_start_up_;
```

```
/* ********************************************************
   *                                                      *
   *                                                      *
   * Copyright (c) 1972 by Massachusetts Institute of     *
   * Technology and Honeywell Information Systems, Inc.   *
   *                                                      *
   *                                                      *
   ******************************************************** */

listen_: procedure (initial_command_line);

/* Multics Listener */
dcl  iox_$user_input ptr ext static;
dcl  iox_$user_io ptr ext static;
dcl (buffer_ptr,
     read_ptr ptr,
     dummy_ptr ptr,
     pct internal static initial (null),
     old_sp
     ) pointer aligned;

dcl (input_length, buffer_length) fixed bin (21);
dcl  total_input_length fixed bin (21);
dcl  entry, /* 0->$listen_, 1->$release_stack */
     i fixed bin aligned;
dcl  code fixed bin (35) aligned;

dcl  initial_command_line char (*) var,
      /* first command line to be executed */
     command_line char (input_length) aligned based (buffer_ptr);

dcl  spno bit (18) aligned, /* used to store stack segno */
    (should_restore_attachments,
    (first, /* "1"b means control structure not initialized */
     quits_not_enabled) int static init ("1"b)
     ) bit (1) aligned;

dcl 1 x based (buffer_ptr) aligned,
    2 ch (0:65536) char (1) unaligned;

dcl 1 label_var aligned based, /* overlay for a label */
    2 target ptr, /* target of entry/label variable */
    2 stackp ptr; /* stack offset of entry/label variable */

dcl 1 ct aligned, /* automatic structure containing control info */
    2 prev_ptr ptr, /* ptr to last listener stack frame (if any) */
    2 release_all label, /* label in "top" level to release to */
    2 release_label, /* label to release to */
    2 new_release label, /* label next invocation is to release to */
    2 start label, /* label for start command */
    2 flags aligned,
      3 dont_restore bit (1) unal, /* "1"b causes io attachments
                                      not to be restored on start */
```

```
      3 pad bit (35) unal,
    2 frame fixed bin, /* stack frame of current invocation */
    2 level fixed bin; /* level of current invocation (from 1) */

dcl 1 bct aligned based (pct) like ct;

dcl  ios_signal_ entry (char (32) aligned, fixed bin (35)),
     iox_$get_line entry (ptr, ptr, fixed bin (21), fixed bin (21))
      returns (fixed bin (35)),
     iox_$control entry (ptr,char (*), ptr) returns (fixed bin (35)),
     com_err_ entry options (variable),
     cu_$cp ext entry (ptr, fixed bin (21), fixed bin (35)),
     cu_$ready_proc ext entry (),
     cu_$grow_stack_frame entry (fixed bin (21), ptr,fixed bin (35)),
     get_system_free_area_ entry returns (ptr),
     cu_$stack_frame_ptr ext entry () returns (ptr);

dcl (addr,
     baseno,
     divide,
     fixed,
     length,
     min,
     null,
     ptr,
     rel
     ) builtin;
dcl  cleanup condition;
dcl  error_table_$long_record ext static fixed bin (35);
%include stack_frame;
```

```
/* Establish this frame as the "top" of the listener frame thread,
   so that this frame cannot be "released" around. */
          entry = 0;
          go to re_enter;

/* Entry called after processing quit or unclaimed signal */
release_stack: entry (should_restore_attachments);

          entry = 1;

/* Save pointer to previous listener control information, save return
\c
   point for subsequent invocations of the listener,
   and initialize switches */
re_enter:
          if first then do; /* no previous invocation to work from */
               ct.prev_ptr = null;
               ct.level = 1; /* this is first invocation */
               sp = cu_$stack_frame_ptr (); /* find stack frame */
               spno = baseno (sp); /* get segno for comparing */
               i = 0;
               do while (baseno (sp -> stack_frame.prev_sp) = spno);
                    i = i + 1;
                    sp = sp -> stack_frame.prev_sp;
               end;
               ct.frame = i;
          end;
          else do; /* can use info from previous invocation */
               ct.prev_ptr = pct;
               ct.level = bct.level + 1;
               old_sp = addr (bct.start) -> label_var.stackp;
                   /* find previous frame */
               sp = cu_$stack_frame_ptr ();
               i = bct.frame;
               do while (sp ^= old_sp);
                 /* find number of intervening frames */
                    i = i + 1;
                    sp = sp -> stack_frame.prev_sp;
               end;
               ct.frame = i;
          end;

/* fill in labels for release and start */
          if (entry = 0) | first then do;
               ct.release_all,
                    ct.release,
                    ct.new_release = readyt;
               first = "0"b;
          end;
          else do;
            /* will want to release to invocation before this one */
               ct.release_all = bct.release_all; /* dont change it */
               ct.release = bct.new_release;
```

```
          ct.new_release = readyt;
     end;
     ct.start = start_return_point;

     pct = addr (ct);
       /* have finished getting info from old frame */

     ct.flags.dont_restore = "0"b;

/* set ptrs to current control info and to buffer
   in which to read in command line */
     buffer_length = 32; /* start with 128 char input buffer */
     call cu_$grow_stack_frame (buffer_length,buffer_ptr, code);
       /* get storage for initial buffer */

/* Establish cleanup procedure to restore control structure thread */
     on condition (cleanup) begin;
          pct = bct.prev_ptr; /* pop structure of interest */
          if pct = null then first = "1"b;
     end;

/* Check for entering via "release_stack" entry without having first
   entered via "listen_". This can happen, e.g., if user takes fault
   before standard_process_overseer_ calls listen_. If this happens,
   enable quits. */
     if quits_not_enabled then do;
          quits_not_enabled = "0"b;
          code = iox_$control (iox_$user_io,"quit_enable",null);
     end;
    /* If called at the listen_ entry, set up initial command
       line and enable quits */
     if entry = 0 then do;
          if initial_command_line ^= "" then do;
               if length (initial_command_line) >
                buffer_length * 4 then do;
                    call com_err_ (0, "listen_",
                     "Initial command line is too long."||
                     "Max=^d chars.",buffer_length*4);
                    go to readyt;
               end;
               input_length = length (initial_command_line);
               command_line = initial_command_line;
               total_input_length = 0;
               go to CALL_CP;
          end;
     end;
```

```
/* ********************START OF BASIC LISTENER LOOP************ */

/* Call the "ready procedure". */
readyt:   call cu_$ready_proc ();

/* Read the next command line */
readnew:  read_ptr = buffer_ptr;
          total_input_length = 0;
           /* extra input line character count */
read:
          code = iox_$get_line (iox_$user_input, read_ptr,
           buffer_length*4-total_input_length,input_length);
          if code ^= 0 then do;
               if code ^= error_table_$long_record then
                call ios_signal ("user_input", code);
               else do;
                    if input_length <
                     buffer_length * 4 - total_input_length
                     then goto CALL_CP;
                    call cu_$grow_stack_frame (buffer_length,
                     dummy_ptr, code); /* double size of buffer */
                    buffer_length = buffer_length + buffer_length;
                    read_ptr = addr (read_ptr -> ch (input_length));
                    total_input_length = total_input_length +
                     input_length;
               end;
               goto read;
          end;

CALL_CP:  call cu_$cp (buffer_ptr, total_input_length + input_length,
                       code);
          if code = 100 then go to readnew;
            /* ignore null command line */
          go to readyt;

/* ****************END OF BASIC LISTENER LOOP****************** */
```

```
start_return_point: /* start command goes here */
          if ct.flags.dont_restore then
           should_restore_attachments = "0"b;
          pct = bct.prev_ptr;
          return;

get_pct:  entry (ct_ptr);
dcl  ct_ptr ptr;

/* Return pointer to control structure */
          ct_ptr = pct;
          return;

get_level: entry (level_no, frame_no);
/* return command level number and stack frame number of caller's
   caller */

dcl (level_no, frame_no) fixed bin;

          if pct = null then do; /* no previous invocation */
               level_no = 0;
               old_sp = ptr (addr (old_sp), 0) ->
                stack_header.stack_begin_ptr;
         /* in case we're not in highest ring */
               frame_no = 0;
          end;
          else do; /* count only up to previous listener */
               level_no = pct -> bct.level;
               old_sp = addr (bct.start) -> label_var.stackp;
               frame_no = bct.frame;
          end;

          sp = cu_$stack_frame_ptr () -> stack_frame.prev_sp ->
           stack_frame.prev_sp;
         /* want frame no of caller's caller */
          do while (sp ^= old_sp);
               frame_no = frame_no + 1;
               sp = sp -> stack_frame.prev_sp;
          end;
          return;

get_area: entry returns (ptr);
          return (get_system_free_area_ ());

%include stack_header;

     end;
```

APPENDIX C

Encoding of Channel Names

Page

The name used to designate an MCS communications channel is a character string of up to 32 characters. The name is composed of components separated by periods, where each component represents a level of multiplexing. the first two components identify the physical channel on an FNP; further components (if present) identify the subchannels of a concentrator (such as a VIP 7700 controller).

Format of physical channel name: The physical channel name (which corresponds to the old-style name of the from ttyXXX) has the following format:

F.ANSS

where:

F	is an FNP identifier (a, b, c, or d)
A	is an adapter type (h for an HSLA channel, l for an LSLA channel)
N	is the number of the particular adapter (0-2 for an HSLA, 0-5 for an LSLA)
SS	is the decimal number of the subchannel on the specified adapter.

Examples:

Name	Description	Old form
a.l000	FNP a, LSLA 0, subchannel 0	tty000
a.h108	FNP a, HSLA 1, subchannel 8	tty708
b.h016	FNP b, HSLA 0, subchannel 16	ttyG16

Multiplexed channels: The format of the additional components of the names of subchannels of a concentrator or "multiplexer" depends on the particular multiplexer; it may be a station id, or a sequential number, etc. For example:

Name	Description
b.h016.01	FNP b, HSLA 0, subchannel 16, concentrator subchannel 1
b.h016.09	same physical channel, concentrator subchannel 9

ARPANET channels: The names of ARPANET channels are of the form netXXX for user telnet channels or ftpXXX for file-transfer channels, where XXX is an arbitrary 3-digit number.

APPENDIX D

Instructor Code for IPC Workshop

Page

The following segments set up the environment such that students may complete the interprocess communication workshop. Note in ipc_driver.pl1 the call to get_userid_. For this call to successfully return, it is required that your instructor obtain read access to >sc1>answer_table.

```
ipc_report: proc;

dcl  i fixed bin,
     get_wdir_ entry returns (char (168)),
     stud_ptr ptr,
     ioa_ entry options (variable),
     hcs_$initiate entry (char (*), char (*), char (*),
     fixed bin (1), fixed bin (2), ptr, fixed bin (35)),
     code fixed bin (35),
     clock_ entry returns (fixed bin (71)),
     date_time_ entry (fixed bin (71), char (*)),
     my_time char (24);

dcl 1 stud_ipc based (stud_ptr) aligned,
    2 index fixed bin,
    2 studs (0 refer (index)),
     (3 codes char (8),
      3 name char (22),
      3 proj char (9),
      3 time char (16)) unal;


          call date_time_ (clock_ (), my_time);
          call hcs_$initiate (get_wdir_ (), "ipc_status", "",
               0, 1, stud_ptr, code);

          call ioa_ ("REPORT FOR F15D WORKSHOP #3 ^a", my_time);
          call ioa_ ("^3/      user_id              time^/");
          do i = 1 to index;
               if name (i) ^= "" then
                    call ioa_ ("^a.^a      ^a",
                               name (i), proj (i), time (i));
          end;

     end ipc_report;
```

```
ipc_driver: proc;

dcl  send_mail_ entry (char (*), char (*), ptr, fixed bin (35)),
     get_wdir_ entry returns (char (168)),
     hcs_$make_seg entry (char (*), char (*), char (*), fixed bin (5),
     ptr, fixed bin (35)),
     hcs_$wakeup entry (bit (36), fixed bin (71), fixed bin (71),
     fixed bin (35)),
     ipc_$create_ev_chn entry (fixed bin (71), fixed bin (35)),
     ipc_$decl_ev_call_chn entry (fixed bin (71), entry, ptr, fixed bin,
     fixed bin (35)),
     get_process_id_ entry returns (bit (36)),
    (ioa_, com_err_) entry options (variable),
     get_userid_ entry (bit (36), char (*), char (*), fixed bin, fixed bin,
     fixed bin (35)),
     iox_$control entry (ptr, char (*), ptr, fixed bin (35)),
     date_time_ entry (fixed bin (71), char (*)),
     clock_ entry returns (fixed bin (71)),
     unique_bits_ entry returns (bit (70)),
     unique_chars_ entry (bit (*)) returns (char (15));

dcl 1 send_mail_info aligned,
    2 version fixed bin init (1),
    2 sent_from char (32) aligned init ("Mr. Wonderful"),
    2 switches,
     (3 wakeup bit (1) init ("1"b),
      3 mbz1 bit (1),
      3 always_add bit (1) init ("1"b),
      3 never_add bit (1) init ("0"b),
      3 mbz2 bit (1),
      3 acknowledge bit (1) init ("0"b),
      3 mbz bit (30)) unal;

dcl  congrats char (40) internal static options (constant)
          init ("Congratulations - mission accomplished!!");
dcl  ipc_status_full_msg char (66) internal static options (constant) init
    ("Instructor's table has overflowed.  Please notify him immediately.");
dcl  destination char (32);
dcl  code fixed bin (35);

dcl 1 set_up based (su_ptr),
    2 my_pid bit (36),
    2 my_chid fixed bin (71);

dcl  me char (10) init ("ipc_driver") static options (constant);

dcl 1 event_info based (ei_ptr),
    2 channel_id fixed bin (71),
    2 message fixed bin (71),
    2 sender bit (36),
    2 origin,
      3 dev_signal bit (18) unal,
      3 ring bit (18) unal,
    2 data_ptr ptr;
```

```
dcl (su_ptr, ei_ptr, sptr) ptr;
dcl  iox_$user_io ext ptr;
dcl  error_table_$invalid_channel ext fixed bin (35);
dcl  string0 static fixed bin (71);
dcl  string1 char (8) based (sptr),
     string2 char (8) based (mptr);
dcl  mptr ptr;
dcl  person char (22),
     project char (9),
    (type, anon) fixed init (0) bin,
     stud_ptr static ptr;

dcl 1 stud_ipc based (stud_ptr) aligned,
    2 index fixed bin,
    2 studs (0 refer (index)),
     (3 codes char (8),
      3 name char (22),
      3 proj char (9),
      3 time char (16)) unal;

dcl  i fixed bin;


/* Set it up */
          call hcs_$make_seg (get_wdir_ (), "channel_info", "", 10,
               su_ptr, code);
          call ipc_$create_ev_chn (my_chid, code);
          if code ^= 0 then do;
               call com_err_ (code, "ipc_driver");
               return;
          end;
          my_pid = get_process_id_ ();
          call ipc_$decl_ev_call_chn (my_chid, wakeme, null (), 0, code);
          if code ^= 0 then do;
               call com_err_ (code, "ipc_driver");
               return;
          end;
          call ioa_ ("End ^a$^a", me, me);
/* ************************************************************** */


dcl  timer_manager_$sleep entry (fixed bin (71), bit (2));
          do while ("1"b);
               call timer_manager_$sleep (900, "11"b);      /* 15 min. */
          end;
          return;


/* Come here when wakeup received */
wakeme:   entry (ei_ptr);

          call hcs_$make_seg (get_wdir_ (), "ipc_status", "", 10,
               stud_ptr, code);
```

```
          if stud_ptr = null () then do;
               call com_err_ (code, "ipc_driver");
               return;
          end;
          call get_userid_ (sender, person, project, type, anon, code);
          if code ^= 0 then do;
               call com_err_ (code, "ipc_driver",
                    "Need 'r' access on >sc1>answer_table");
               return;
          end;

          call ioa_ ("A wakeup from ^a.^a was just received.", person,
               project); /* But that does not imply he'll wake me
                            up again with the proper reversed msg */

          mptr = addr (message);
          do i = 1 to 250 while (codes (i) ^= "");
               if string2 = codes (i) then goto got_one;
          end;

          if i = 251 then do;
               call com_err_ (0, "ipc_driver",
                    "Table in the segment 'ipc_status' is full.
System will not function properly.");
               destination = rtrim (person) || "." || rtrim (project);
               call send_mail_ (rtrim (destination), ipc_status_full_msg,
                    addr (send_mail_info), code);
               if code ^= 0 then do;
                    call com_err_ (code, "ipc_driver", "Bad call to send_ma
occurred while trying to complain about full table in 'ipc_status'.");
                    return;
               end;
               return;
          end;

          sptr = addr (string0);  /* Overlay string1 onto string0 */
          string1 = substr (unique_chars_ (unique_bits_ ()), 8, 15);
          index = index + 1  /* index reflects the true size of
                                stud_ipc.studs array */
          codes (index) = reverse (string1);
          call hcs_$wakeup (sender, message, string0, code);
          if code ^= 0 then do;
               call com_err_ (code, "ipc_driver");
               return;
          end;
          goto finis;


got_one:
          name (i) = person;
          proj (i) = project;
          call date_time_ (clock_ (), time (i));
          call ioa_ ("^a.^a completes assignment", person, project);
          destination = rtrim (person)||"."||rtrim (project);
```

```
          call send_mail_ (rtrim (destination), congrats,
                    addr (send_mail_info), code);
          if code ^= 0 then do;
               call com_err_ (code, "ipc_driver", "Bad call to send_mail_");
               return;
          end;

finis:

     end ipc_driver;
```

```
init_ipc: proc;

dcl  get_wdir_ entry returns (char (168));
dcl 1 stud_ipc based (stud_ptr) aligned,
    2 index fixed bin,
    2 studs (250),
     (3 codes char (8),
      3 name char (22),
      3 proj char (9),
      3 time char (16)) unal;
dcl  hcs_$make_seg entry (char (*), char (*), char (*),
     fixed bin (5), ptr, fixed bin (35));
dcl  hcs_$add_acl_entries entry (char (*), char (*), ptr,
     fixed bin, fixed bin (35));
dcl 1 seg_acl aligned,
    2 access_name char (32) init ("*.F15d.*"),
    2 modes bit (36) init ("1"b),
    2 zero_pad bit (36) init ("0"b),
    2 status_code fixed bin (35);
dcl  code fixed bin (35),
     ioa_ entry options (variable),
     stud_ptr ptr,
     i fixed bin;

          call ioa_ ("Begin init_ipc");
          call hcs_$make_seg (get_wdir_ (), "ipc_status", "",
               10, stud_ptr, code);
          call hcs_$add_acl_entries (get_wdir_ (), "ipc_status",
                                     addr (seg_acl), 1, code);
          index = 0;
          do i = 1 to 250;
               time (i), name (i), proj (i), codes (i) = "";
          end;
          call ioa_ ("End init_ipc");
     end init_ipc;


          ABSENTEE SCRIPT ipc.absin

&ready off
cwd >udd>F15d>s1
ipc_driver
& To prevent the absout segment from growing inordinately large,
& the instructor has inserted the following command, which will guarantee
& an absout segment of 10k or less.
if [greater [st ipc.absout -bc] 1474560] -then "tc ipc.absout"
logout
```

APPENDIX E

Standard Process Overseers

Page

On April 23, 1981, the following command was typed:

```
li *overseer_.* -library source
```

The terminal output that resulted appears below.

```
1 accounts_overseer_.pl1                              type:  arch comp
                path:  >ldd>tools>source>bound_admin_rtnes_.s.archive
  component updated:  01/29/75  1711.6

1 cards_overseer_.pl1                                 type:  arch comp
                path:  >ldd>tools>source>bound_card_input_.s.archive
  component updated:  09/04/79  1718.4

1 dfast_process_overseer_.pl1                         type:  arch comp
                path:  >ldd>unb>source>bound_dfast_.s.archive
  component updated:  09/01/76  1342.7

1 fst_process_overseer_.pl1                           type:  arch comp
                path:  >ldd>unb>source>bound_fast_.s.archive
  component updated:  06/07/77  1504.7

1 ftp_server_overseer_.pl1                            type:  arch comp
                path:  >ldd>net>source>bound_ftp_server_.s.archive
  component updated:  09/23/77  1031.5

1 iod_overseer_.pl1                                   type:  arch comp
                path:  >ldd>tools>source>bound_iodc_.s.archive
  component updated:  03/13/81  1038.7

1 terminals_overseer_.pl1                             type:  arch comp
                path:  >ldd>tools>source>bound_admin_rtnes_.s.archive
  component updated:  03/23/81  1014.8

1 tolts_overseer_.pl1                                 type:  segment
                path:  >ldd>tools>source
  contents modified:  12/01/80  1136.8
```

Another important overseer: project_start_up_

APPENDIX F

Gate and Message Segment Examples

Page

APPENDIX G

Advanced Dial Facility Example

Page

```
set_up_dial: proc;

/* The set_up_dial entry point initializes the dialing environment

   1) An event-call channel is established so that
   the answering service can notify this process
   of dialins, hangups, etc.

   2) Dials are enabled
   */
/*  'dialok' attribute essential if this procedure is to succeed */

dcl  ipc_$create_ev_chn entry (fixed bin (71), fixed bin (35)),
     ipc_$delete_ev_chn entry (fixed bin (71), fixed bin (35)),
     ipc_$decl_ev_call_chn entry (fixed bin (71), entry,
     ptr, fixed bin, fixed bin (35)),
     hcs_$assign_channel entry (fixed bin (71), fixed bin (35)),
     dial_manager_$allow_dials entry (ptr, fixed bin (35)),
    (ioa_, com_err_, ioa_$ioa_switch) entry options (variable);

dcl  wasted_channel fixed bin (71);
dcl  time char (24);
dcl  code fixed bin (35),
     iox_$user_output external static ptr,
     ME char (12) varying init ("set_up_dial") ;


dcl 1 dial_manager_arg aligned static,
    2 version fixed bin init (1),
    2 dial_qualifier char (22) init ("astra"),
    2 dial_channel fixed bin (71),
    2 channel_name char (32) ;

/* ********************************************** */

          call ioa_ ("Begin ^a", ME);

/* The following code is inserted to fake out tty_, which will
attempt to give us fast channels we can not deal with (see the
'read_status' control order). By consuming all the fast channels now
(and there aren't many available to us), we'll force tty_ to use
garden_variety channels, which we can easily handle.  */

          code = 0;
          do while (code = 0);
               call hcs_$assign_channel (wasted_channel, code);
          end;
```

```
/* Channel must be obtained for notifying this process of */
/* all hangups and dialups.                               */
          call ipc_$create_ev_chn (dial_manager_arg.dial_channel,
               code);
          if code ^= 0 then call ERROR (1);

/* Next let the answering service know we will accept
   dials - we must do this before changing the event-wait
   channel to an event-call channel  */
          call dial_manager_$allow_dials (addr (dial_manager_arg),
               code);
          if code ^= 0 then call ERROR (2);

/* Make the event-wait channel an event-call channel and
   specify that my 'dial_handler' will be invoked whenever
   the answering service wakes my process on this channel */
          call ipc_$decl_ev_call_chn (dial_manager_arg.dial_channel,
               dial_handler, null (), 0, code);




          if code ^= 0 then call ERROR (3);

/* Okay...now return and wait for something to happen */
          call ioa_ ("Now listening for dials: ^a", ME);
          return;


/* ********************************************** */
dial_handler: entry (info_ptr);

/* Handler for dial messages - this entry point will
   be invoked whenever something happens that the answering
   service notifies me about                          */

dcl  info_ptr ptr parameter;
dcl 1 event_info based (info_ptr),
    2 channel_id fixed bin (71),
    2 message fixed bin (71),
    2 sender bit (36),
    2 origin,
      3 dev_signal bit (18) unal,
      3 ring bit (18) unal,
    2 data_ptr ptr;

dcl  listen_to_dial entry(ptr); /* proc to dialog with terminals */
```

```
dcl  convert_dial_message_$return_io_module entry
    (fixed bin (71), char (*), char (*), fixed bin,
     1 aligned like status_flags, fixed bin (35));

dcl 1 status_flags aligned,
   (2 dialed_up bit (1),
    2 hung_up bit (1),
    2 control bit (1),
    2 pad bit (33)) unal;

dcl 1 dialed static,          /* This structure works for
                                 max_num_allowed <= 10 */
    2 no_dialed fixed bin init (0),
    2 sw (10),
      3 swname char (6) init (
     "dial01", "dial02",
     "dial03", "dial04",
     "dial05", "dial06",
     "dial07", "dial08",
     "dial09", "dial10"),
      3 iocb_ptr ptr init ((10)null ()),/* An available switch is
                                           characterized by null ptr */
      3 devname char (32);

/* We will only talk to 2 terminals at a time, hanging up the third */
dcl  max_num_allowed internal static options (constant) init (2);

dcl  nomore_ptr internal static ptr init (null ()); /* This iocbptr is
          used for talking to a doomed tty when system full */

dcl  iox_$find_iocb entry (char (*), ptr, fixed bin (35)),
     iox_$attach_ptr entry (ptr, char (*), ptr, fixed bin (35)),
     iox_$destroy_iocb entry (ptr, fixed bin (35)),
     iox_$control entry (ptr, char (*), ptr, fixed bin (35)),
     error_table_$io_no_permission external static fixed bin (35),
     iox_$open entry (ptr, fixed bin, bit (1) aligned, fixed bin (35)),
     iox_$detach_iocb entry (ptr, fixed bin (35)),
     iox_$close entry (ptr, fixed bin (35)),
     clock_ entry returns (fixed bin (71)),
     date_time_ entry (fixed bin (71), char (*)),
     ipc_$cutoff entry (fixed bin (71), fixed bin (35)),
     ipc_$reconnect entry (fixed bin (71), fixed bin (35));

dcl  i fixed bin;   /* an index */

dcl (which_channel automatic, nomore_channel static) char (32),
     io_module char (32),
     n_dialed fixed bin;
```

```
          ME = "dial_handler";          /* For com_err_ */
          /* First of all, interpret the event message sent from
             the answering service - it should either be that
             someone has dialed in or hung up */
          call convert_dial_message_$return_io_module (
               event_info.message, which_channel,
               io_module, n_dialed, status_flags, code);
          /* n_dialed = -1 if this is an informative
                              message (which it is)*/
          if code ^= 0 &
           code ^= error_table_$io_no_permission then call ERROR (5);

/* Log in event */

          call date_time_ (clock_ (), time);
          call ioa_$ioa_switch (iox_$user_output,
               "^a: ^[DIALED UP^;HUNG UP^] AT ^a.",
               which_channel, status_flags.dialed_up, time);

/* Restart any interrupted io to the master terminal        */
          call iox_$control (iox_$user_output, "start", null (), code);

          if ^status_flags.dialed_up then do;   /* Then it must
                    be a hang up and we have to find out which
                    device, mark it available, & detach the switch. */

                    do i = 1 to max_num_allowed
                              while (dialed.sw (i).devname ^=
                              which_channel);
                    end;

             if ^(i > max_num_allowed) then do;
                    dialed.no_dialed = dialed.no_dialed -1;
                    call ioa_$ioa_switch (iox_$user_output,
"^9xAt this instant, ^i ^[terminal^;terminals^] ^[is^;are^] logged on.",
                         dialed.no_dialed, (dialed.no_dialed = 1),
                         (dialed.no_dialed = 1));

          /* Close and detach the switch */
                    call iox_$close (dialed.sw (i).iocb_ptr, code);
                    if code ^= 0 then call ERROR (6);
                    call iox_$detach_iocb (dialed.sw (i).iocb_ptr,
                         code);
                    if code ^= error_table_$io_no_permission
                    & code ^= 0 then call ERROR (7);
                     dialed.sw (i).iocb_ptr = null ();   /* free
                              this switch */
               end;
               return;
          end;
```

```
      else if status_flags.dialed_up then do;
    /* Somebody dialed in - get to work on attaching and
     listening to him.  As we must update a critical database
     (the structure 'dialed'), let us prevent interruption
     during the update period by 'masking' wakeups on the
     event-call channel 'dial_manager_arg.dial_channel'.  */
               call ipc_$cutoff (dial_manager_arg.dial_channel, code);
               if code ^= 0 then call ERROR (4);

                    /* Loop until we find a free iocb
                    (indicated by a null iocb_ptr), OR
                    until we exceed max_num_allowed.  */
               do i = 1 to max_num_allowed
                         while (dialed.sw (i).iocb_ptr
                         ^= null ());
               end;

               if ^(i > max_num_allowed) then do;
                    /* if there's a switch available */
                    dialed.devname (i) = which_channel;
                    dialed.no_dialed = dialed.no_dialed + 1;
                    call ioa_$ioa_switch (iox_$user_output,
"^9xAt this instant, ^i ^[terminal^;terminals^] ^[is^;are^] logged on.",
                         dialed.no_dialed, (dialed.no_dialed = 1),
                         (dialed.no_dialed = 1));

          /* Find an iocb for the user and attach user's device
             via tty_ I/O module */
                    call iox_$find_iocb (dialed.sw (i).swname,
                    dialed.sw (i).iocb_ptr, code);
                    call ipc_$reconnect
                    (dial_manager_arg.dial_channel, code);
                              /* Safe to unmask */

                    call iox_$attach_ptr (dialed.sw (i).iocb_ptr,
                         "tty_ "||dialed.sw (i).devname, null (), code);
                    if code ^= 0 then call ERROR (9);


                    call iox_$open (dialed.sw (i).iocb_ptr,
                         3 /* stream_io */,
                         "0"b, /* unused must be zero */ code);
                    if code ^= 0 then call ERROR (10);
```

```
/* Now that we have attached the user's terminal, call the main
   program that handles these users; it will handle all
   processing for these terminals, and when the user is
   done, it will simply return to me */
                    call listen_to_dial (dialed.sw (i).iocb_ptr);
                    return;

               end;
               else do;       /* we've run out of switches */
                    if nomore_ptr = null () then do;
                         call iox_$find_iocb ("nomore", nomore_ptr, code);
                         /* Nope, we can not avoid this call. */
                         if code ^= 0 then call ERROR (14);
                    end;
                    nomore_channel = which_channel;
                    call iox_$attach_ptr (nomore_ptr,
                              "tty_ "||which_channel,
                              null (), code);
                    if code ^= 0 then call ERROR (15);
                    call iox_$open (nomore_ptr, 2, "0"b, code);
                         /* Stream output suffices for doomed tty */
                    if code ^= 0 then call ERROR (16);
                    call ioa_$ioa_switch (nomore_ptr,
                    "DIAL SYSTEM astra FULL WITH ^i USERS.
TRY AGAIN LATER.", max_num_allowed);    /* SORRY FELLA */
                    call iox_$control (nomore_ptr, "hangup", null (), code);
                    if code ^= 0 then call ERROR (17);
                    call iox_$close (nomore_ptr, code);
                    if code ^= 0 then call ERROR (24);
                    call iox_$detach_iocb (nomore_ptr, code);
                    if code ^= 0 &
                     code ^= error_table_$io_no_permission
                         then call ERROR (25);
                    call ipc_$reconnect (dial_manager_arg.dial_channel,
                                         code); /* Safe to unmask now */
                    return;
               end;
          end;

ERROR:    proc (error_number);
          /* Internal proc to report errors */
dcl  error_number;
               call com_err_ (code, ME, "Check call ^i of ERROR", error_number);
/* Restart any interrupted io to the master terminal        */
               call iox_$control (iox_$user_output, "start", null (), code);
               goto FINISH;
          end;
```

```
shutoff:  entry;

          /* This entry point resets the environment:
                    1) shuts off dials
                    2) zeroes dialed.no_dialed
                    3) wipes out iocbs and nulls ptrs. */

dcl  dial_manager_$shutoff_dials entry (ptr, fixed bin (35));
dcl  ipc_$decl_ev_wait_chn entry (fixed bin (71), fixed bin (35));

          ME = "shutoff";

/*        IT IS FIRST NECESSARY TO CHANGE THE EVENT-CALL BACK INTO AN
   EVENT-WAIT CHANNEL, BECAUSE THE shutoff_dials ENTRY POINT WILL
   NOT WORK ON ANYTHING BUT.                                        */

          call ipc_$decl_ev_wait_chn (dial_manager_arg.dial_channel, code);
          if code ^= 0 then call ERROR (21);
          call dial_manager_$shutoff_dials (addr (dial_manager_arg), code);
          if code ^= 0 then call ERROR (22);
          call ipc_$delete_ev_chn (dial_manager_arg.dial_channel, code);
          if code ^= 0 then call ERROR (23);
          dialed.no_dialed = 0;   /* In case we 'set_up_dial' again */
          do i = 1 to max_num_allowed;
              if dialed.sw(i).iocb_ptr ^=null() then do;
                    call iox_$close (dialed.sw(i).iocb_ptr, code);
                    call iox_$detach_iocb (dialed.sw(i).iocb_ptr, code);
                    call iox_$destroy_iocb (dialed.sw(i).iocb_ptr, code);
                      /* The above call automatically NULLS the
                         iocb_ptr.                          */
               end;
          end;
          return;
FINISH:
     end set_up_dial;
```

```
listen_to_dial: proc (iocb_ptr);

/*        Procedure to dialog with a dial-up terminal       */

%include dcl_iox_entries;

dcl
     iocb_ptr ptr parameter,
     code fixed bin (35),
     hcs_$initiate_count entry (char (*), char (*), char (*),
     fixed bin (24), fixed bin (2), ptr, fixed bin (35)),
    (seg_ptr_1, seg_ptr_2) internal static ptr init (null ()),
    (bit_count_1, bit_count_2) fixed bin (24) internal static,
     buff_ptr ptr,
     buffer char (256),
     n_read fixed bin (21),
     com_err_ entry options (variable),
     get_wdir_ entry returns (char (168));

dcl 1 info_structure aligned,
    2 ev_chan fixed bin (71),
    2 input_available bit (1);
dcl (addr, after) builtin;
dcl  ipc_$decl_ev_call_chn entry (fixed bin (71), entry,
     ptr, fixed bin, fixed bin (35));
dcl  ipc_$delete_ev_chn entry (fixed bin (71), fixed bin (35));

dcl  ioa_$ioa_switch entry options (variable);

%include iocb; /* We need this structure to get the
          attach_descrip_ptr */

dcl 1 attach_description unaligned based
          (iocb_ptr -> iocb.attach_descrip_ptr),
    2 descrip_length fixed bin (35),
    2 desc char (0
          refer (attach_description.descrip_length)) unal;

          call ioa_$ioa_switch (iocb_ptr,
          "Dialed to 'astra' on channel ^a.",
           after (attach_description.desc, " "));
          if seg_ptr_1 = null () then  /* Efficiency technique */
               call hcs_$initiate_count (get_wdir_ (), "msg1",
                    "", bit_count_1, 1, seg_ptr_1, code);
          call iox_$put_chars (iocb_ptr, seg_ptr_1,
                    bit_count_1/9, code);
          if code ^= 0 then call ERROR;
```

```
          call iox_$control (iocb_ptr, "read_status",
                    addr (info_structure), code);
          if code ^= 0 then call ERROR;
          if ^info_structure.input_available then do;
          /* If no line is available, use 'garden_variety'
          event channel provided to us by 'read_status' order */
               call ipc_$decl_ev_call_chn (info_structure.ev_chan,
                    respond_to_line_later, iocb_ptr, 0, code);
          /* iocb_ptr in above call is really our 'data_ptr' */
               if code ^= 0 then call ERROR;
          end;
          else call respond_to_line_now (iocb_ptr, 0);  /* Else
                    there is something out there to read NOW */
          return;   /*  so much for listen_to_dial  */

respond_to_line_later: entry (information_ptr);

dcl  information_ptr ptr parameter;

dcl 1 event_info based (information_ptr),
    2 channel_id fixed bin (71),
    2 message fixed bin (71),
    2 sender bit (36),
    2 origin,
      3 dev_signal bit (18) unal,
      3 ring bit (18) unal,
    2 data_ptr ptr;

          call respond_to_line_now (event_info.data_ptr,
               event_info.channel_id);
          /* But we are not sure whether there is a quit or
             null line out there. */
          return;

respond_to_line_now: entry (iocb_ptr, read_status_channel);

dcl  read_status_channel fixed bin (71) parameter;
          /* Channel ultimately to be deleted */

          call iox_$control (iocb_ptr, "read_status",
                    addr (info_structure), code);
          if code ^= 0 then call ERROR;
```

```
          if info_structure.input_available then do;
               n_read = 0;
               buff_ptr = addr (buffer);
               call iox_$get_chars (iocb_ptr, buff_ptr,
                    256, n_read, code);
               if code ^= 0 then call ERROR;
               if n_read > 1 then do; /* If there's something more
                    than just a newline, then do...    */
                    call iox_$put_chars (iocb_ptr, buff_ptr,
                         n_read, code);
                    if code ^= 0 then call ERROR;
                    if seg_ptr_2 = null () then
                         call hcs_$initiate_count (get_wdir_(),
                              "msg2", "", bit_count_2, 1,
                              seg_ptr_2, code);
          /* Ignore code,  it should be fine */

                    call iox_$put_chars (iocb_ptr, seg_ptr_2,
                         bit_count_2/9, code);


                    if read_status_channel ^= 0 then call
                         ipc_$delete_ev_chn (read_status_channel,
                              code); /* We no longer need it */
                    call iox_$control (iocb_ptr, "hangup",
                              null (), code);
                    if code ^= 0 then call ERROR;
               end;
               return;     /* There really is nothing else to do */
          end;
          else return;        /* If no input_available */

ERROR:    proc;

               call com_err_ (code, "listen_to_dial",
                    "Truly unexpected.");
               goto return_point;       /* Intentionally non-local */
          end ERROR;

return_point: return;

     end listen_to_dial;
```

Quiz 1

1. Subsystem requests that accept pathnames as input may allow the final entryname in the pathname to be a star name. To first determine whether the starname is valid (ie. does not begin or end with a period etc.), the following routine should be called:

a. hcs_$star

b. hcs_$star_list

c. check_star_name_

d. match_star_name_

e. none of the above

2. A free pool of temporary segments available to each user makes it possible to use the same temporary segment more than once via the "get_temp_segments_" and "release_temp_segments_" routines, without having to create one when needed.

a. temp segs belong to a single procedure when in use

b. the maximum size of a temp seg is 16k

c. the above mentioned free pool is system wide

d. if all temp segs have been used, an appropriate error is returned

e. all of the above

3. A multisegment file is composed of one or more components each the size of a segment.

a. like single segment files, any word in a MSF can be specified by a pathname and a word offset

b. the first component of a MSF is named component 0

c. components are stored in a relational data base with the pathname of the MSF

d. components are identified by consecutive upper case letters

e. all of the above

4. A call to the routine hcs_$status_minf is commonly used to distinguish between a segment, a directory and a MSF. Specifically the return arguments indicate a MSF when:

a. type is 0

b. type is 1

c. type is 2 and bit count is 0

d. type is 2 and bit count is nonzero

e. none of the above

5. The working directory is the directory in which the user's activity is centered and which identifies the user's location within the storage system. The directory that becomes the working dir when the "cwd" command is given without arguments is known as the:

a. home directory

b. referencing directory

c. process directory

d. default working directory

e. none of the above

6. The subroutine "hcs_$set_max_length" is often used to define the size limit of a segment so that an "out_of_bounds" condition will be obtained with any too-large offset.

a. the normal maximum length of a segment is 255k

b. the maximum length of a directory is 64k

c. the maximum length of a directory cannot be changed

d. system wide maximum lengths can be imposed by the system administrator

e. all of the above

7. The "msf_manager" Multics subroutine creates a File Control Block (fcb) in order to keep track of manipulations on the MSF. Which of the following is true?

a. the MSF must exist for the fcb to be allocated

b. the fcb is allocated on the stack frame of the caller

c. the fcb is good throughout the life of the the process

d. the address of the fcb must be given for any operation on the MSF

e. all of the above

8. Many Multics subroutines require an "area_ptr" to an area in which data and information is returned to the user. This is so for example with "hcs_$star" which returns star name matches. Areas can be formatted by:

a. a PL1 declaration specifying the "area" attribute

b. the "define_area_" subroutine

c. the Multics command "create_area"

d. use of the function "get_system_free_area_ ()"

e. all of the above

9. The subroutine call to "hcs_$star" returns 2 pointers within a user provided area, star_entry_ptr and star_names_ptr. The star_names_ptr addresses a name array of matching names:

a. for links

b. for segments

c. for directories

d. for entries depending upon the value of "star_select_sw"

e. none of the above

10. The "define_area_" subroutine must be provided with the address of some place to be formatted into an area. This address in the form of a pointer:

a. is provided as argument 1 of the call to "define_area_"

b. must not be null

c. is provided within the information structure that "define_area_" must know about

d. must point to a permanent segment in the storage hierarchy

e. none of the above

Quiz 2

1. If the ring brackets for a particular object segment are (x y z) the segment is regarded as a gate when:

a. x = y = z

b. x < z

c. y < z

d. x > z

e. none of the above

2. Inner ring procedures are often called by outer ring procedures to perform some service. It is necessary therefore for the inner ring procedure to know which ring it is working for. This ring information known as the "VALIDATION LEVEL" can be obtained via:

a. hcs_$set_ring_brackets

b. cu_$level_set

c. set_ring_ ()

d. cross_ring_

e. none of the above

3. Many subsystems honor an "e"- request (qedx calc probe etc.) What entrypoint is used in the implementation of this feature?

a. cu_$generate_call

b. cu_$cp

c. cu_$set_command_procesdsor

d. cu_$grow_stack_frame

e. none of the above

4. Since the "hcs_" subroutine is the means whereby the user can manipulate ring 0 directory segments, likely ring brackets are:

a. 0 0 0

b. 0 1 1

c. 0 0 5

d. 4 4 4

e. none of the above

5. The command level intermediary is usually not an active procedure unless an abnormal event (eg. condition) occurs and it becomes invoked. The procedure responsible for invoking it is:

a. default error handler

b. cu_$cl

c. cu_$set_cl_intermediary

d. set_to_cl_$unclaimed_signal

e. none of the above

6. Saving the attachments of the standard I/O switches, restoring these attachments to their default state and entering a new loop of reading and executing command lines is part of establishing a new command level. This is accomplished by:

a. the current command level intermediary

b. the current command processor

c. the current process overseer

d. the listener (listen_)

e. none of the above

7. The routine responsible for actually printing the ready message which indicates command level to the user is:

a. listen_

b. cu_$ready_proc

c. cu_$set_ready_procedure

d. cu_$set_ready_procedure

e. none of the above

8. Consider the case of a user (Ring 4) attempting to execute a segment with ring brackets of (3, 3, 4)

a. a segment can be in one ring only

b. the user can only read the segment subject of course to ACL and AIM

c. the user's process cannot execute in ring 3

d. the user can execute the segment with a ring change

e. none of the above

9. The "hcs_$set_entry_bound" routine provides the user with a method of limiting which locations of a segment may be targets of a call. If the entry-bound is to be set for a given object segment:

a. the user must have "modify" permission on the containing directory

b. all calls to the segment must be made to an entrypoint with offset less than the entry-bound

c. the segment itself remains unchanged

d. the default entry bound is 0

e. all of the above

10. The "cross_ring_" io_module which allows cross ring attachments of switches:

a. must be given the outer ring switchname as an argument

b. is used to attach the inner ring switch to a previously existing outer ring switch

c. enables use of the "cross_ring_io_$allow_cross" subroutine to do cross ring io

d. the inner ring switch must be open

e. all of the above

Quiz 3

1. There is actually a delay between the time a wakeup is received and the time the process is notified. It is possible therefore for several wakeups to be queued by the time a process is awakened. The priority disposition is as follows:

a. event wait channels have priority by default

b. event call channels have priority by default

c. no priority exists and priority cannot be assigned

d. no priority exists but priority can be assigned

e. none of the above

2. When a process establishes an event call channel:

a. the process must go blocked on the channel and wait for a wakeup to be received

b. the channel should be polled via "ipc_$read_ev_chn" to determine whether a wakeup has been received

c. the process should call "timer_manager_$sleep" for at least 900 seconds

d. the process may continue executing until interrupted by a wakeup on that channel

e. none of the above

3. On occasion, io to the user's terminal may be interrupted by an invoked "ipc_" or "timer_manager_" routine. In order to get things going again, the user should call the "iox_$control" subroutine with the following order:

a. printer_on

b. quit_disable

c. start

d. set_delay

e. none of the above

4. The IOCB which is the supporting structure for a switch, is created and partially initialized by either "iox_$attach_name" or "iox_$attach_ptr". Thereafter the IOCB is updated and maintained by:

a. the io_module

b. the user program

c. iox_$look_iocb

d. continue_to_signal

e. none of the above

5. When "pointer or "entry" variables are to be changed in a chain of synonymously attached IOCB's, the change must take place in the actual IOCB and then reflected in the other IOCB's via:

a. <module_name>$<module_name>attach

b. iox_$find_iocb

c. iox_$move_attach

d. iox_$propagate

e. none of the above

6. If a switch has been opened for "stream_input", obviously record_io is not supported. This implies that the "entry" value for "read_record" in the IOCB would be:

a. a null pointer

b. a null character string

c. iox_$err_no_operation

d. error_table_$no_operation

e. none of the above

7. In order that a process communicate with another, it must know the event channel identifier of a channel created by the other process. The former process sets this info from:

a. user_info_$terminal_data

b. ipc_$read_ev_chn

c. ipc_$decl_ev_call_chn

d. ipc_$decl_ev_wait_chn

e. none of the above

When a process is awakened on an event call channel, control is immediately passed to the procedure specified by the "ipc_$decl_ev_call_chn" with one argument. This argument is a pointer to a structure that specifies:

a. the channels on which events are being awaited

b. information about the event that caused it to return

c. data passed from the procedure that set up the event call channel

d. the machine conditions at the time of the interrupt

e. none of the above

8. The one way control path over which notification of the occurance of events is transmitted is called an event channel. The user obtains an event channel for the purposes of inter-process communication by:

a. looking up the channel master file (CMF)

b. looking up the channel definition table (CDT)

c. making one up

d. calling ipc_$create_ev_chn

e. none of the above

10. If a user wishes to inhibit reading of events on a particular channel but would like to have them queued for later handling, the following procedure should be invoked:

a. ipc_$cutoff

b. create_ips_mask_

c. ipc_$mask_ev_calls

d. ipc_$block

e. none of the above

Quiz 4

1. Often to ensure that only one process at a time can execute critical section of code, that section of code is associated with a so-called "lockword" which must be zero (unlocked) and into which a process places an identifier (thus locking it) via "set_lock_" routine. If many critical sections of code share the same lockword:

a. an execution time error will result

b. each section can be executed by a different process at a given time, thus multiplexing the lockword

c. only one process can execute in any section at a given time

d. the lockword will be reset by any process encountering a locked lock from another section

e. none of the above

2. The "lock identifier" placed in the lockword by a process to indicate a locked status is a:

a. bit string of 36 binary 1's

b. bit string of 36 binary zeroes

c. unique value generated on the fly

d. a special lock identifier kept in the active process table entry for this process

e. none of the above

3. If a procedure that sets up timers by calling entrypoints in "timer_manager_" terminates abnormally, those timers which have been set will:

a. go off at some undesired time

b. be discarded as the procedure is popped off the stack

c. cause the process to hang

d. be reset by the system default condition handler

e. none of the above

4. When an "alrm" or "cput" condition is signalled:

a. the stack is searched for a user defined on-unit

b. a static handler is invoked that determines which user-specified procedure should be called

c. some handler executes and the user's process is returned to command level

d. either internal static storage for the "timer_manager_" subroutine has been destroyed or the system is about to crash

e. none of the above

5. In order that a process so blocked for a certain period of time the following subroutine should be called.

a. timer_manager_$sleep

b. ipc-$block

c. create_ips_mask_

d. sus_signal_handler_

e. none of the above

6. The user_free_ptr stored in the process stack header points to the place where based and controlled variables etc. are stored. This is by default the:

a. process data segment

b. descriptor segment

c. [unique].area.linker

d. user ring stack

e. none of the above

7. The process stack segment is a:

a. circular linked list

b. single forward linked list

c. double threaded list with forward and backward pointers

d. stack of controlled variables

e. none of the above

8. A null pointer has a segment number of:

a. negative 2

b. negative 1

c. zero

d. "garbage value" that causes a fault tag 2

e. none of the above

9. The standard descriptor that complements the standard argument list indicates the argument data type to be an "offset" if the TYPE field has a value of

a. 16

b. 17

c. 18

d. 19

e. none of the above

10. The "cds" software used to build data segments for the user:

a. creates a file in standard object format

b. enables users to reference data using the format <data_seg>$<data_item>

c. requires as input a pointer addressing among other things the name of the data segment

d. invokes the PL1 compiler when the user issues the "cds" command

e. all of the above

APPENDIX W

Workshops

	Page
Workshop One	W-1
Workshop Two	W-3
Workshop Three	W-6
Workshop Four	W-7
Workshop Five	W-9
Workshop Six	W-12
Workshop Seven	W-13

WORKSHOP ONE

1. Copy the segment >udd>F15D>s1>STATUS.pl1. This is the source code for the program that was discussed in Topic 2. Alter this program so that it will:

   -- accept the star convention

   -- print out the link target if it is a link

2. When calling hcs_$star_ you must pass it a pointer to an area. Rather than using get_system_free_area_, create an area using the define_area_ subroutine. Have define_area obtain a segment for the area from the temporary segment pool. Make the area freeing, zero_on_free, extensible with a size of 50 words. Pass hcs_$star_ a pointer to that area.

   Part of this workshop is to look at the contents of the area you have created. Therefore, contrary to good programming practice, do not free the structures allocated in the area by hcs_$star_.

3. Test out your program. Be sure to test its ability to accept star names. As a final test give your program the argument, **. Use the list_temp_segments command. The last temp segment will be the area you just created. Use the area_status command with the -long argument to examine the contents of the area. (The area_status command will accept a segment number for an argument, therefore, you can avoid typing the screech name).

   You will probably want to use the following declarations which are in >udd>F15D>s1>include>w1.incl.pl1.

```
dcl hcs_$get_link_target entry (char(*), char(*), char(*),
                                char(*), fixed bin(35));
dcl check_star_name_$entry entry (char(*), fixed bin(35));
dcl hcs_$star_ entry (char(*), char(*), fixed bin(2), ptr,
                      fixed bin, ptr, ptr, fixed bin(35));
dcl star_entry_count fixed bin;
dcl star_entry_ptr pointer;
dcl star_names_ptr pointer;
dcl 1 star_entries (star_entry_count) aligned based (star_entry_ptr),
      2 type fixed binary (2) unsigned unaligned,
      2 nnames fixed bin (16) unsigned unaligned,
      2 nindex fixed bin (18) unsigned unaligned;

dcl star_names (sum (star_entries (*).nnames)) char (32)
       based (star_names_ptr);

dcl define_area_ entry (ptr, fixed bin(35));
dcl area_info_ entry (ptr, fixed bin(35));

dcl  area_infop ptr;

dcl 1 area_info aligned based (area_infop),
    2 version fixed bin,
    2 control aligned like area_control,
    2 owner char (32) unal,
    2 n_components fixed bin,    /* returned only */
    2 size fixed bin (18),
    2 version_of_area fixed bin, /* returned only */
    2 areap ptr,
    2 allocated_blocks fixed bin,
    2 free_blocks fixed bin,
    2 allocated_words fixed bin (30),
    2 free_words fixed bin (30);

dcl 1 area_control aligned based,
    2 extend bit (1) unal,
    2 zero_on_alloc bit (1) unal,
    2 zero_on_free bit (1) unal,
    2 dont_free bit (1) unal,
    2 no_freeing bit (1) unal,
    2 system bit (1) unal,
    2 pad bit (30) unal;
```

Changing the Command Environment

The command environment can be shaped in several ways by using cu_ entry points. In this workshop, you will write a procedure which will 'intercept' commands before they get to your usual command processor, and by so doing, restrict yourself to a limited set of commands. In addition, your own version of a ready procedure will be installed.

In order to use some internal static variables it is recommended that you write just one procedure with alternate entry points. Your procedure should have the following skeleton:

```
change_env: proc;

   <declarations>

   <code to save entry value of current command processor
    and install interceptor and new ready procedure>

restore_cp: entry;

   <code to reinstall the command processor whose entry
    value you saved>

my_ready: entry;

   <code for new ready procedure>

command_interceptor: entry (a_line_ptr, a_line_len, code);

   <code for your command interceptor>

end change_env;
```

1. The 'my_ready' entry point should determine if you are in 'ready on' mode. If so, it should output the following line:

   'Next Command: '

   You should not append a new-line character at the end of this 'ready message'.

2. The 'command_interceptor' entry point will intercept commands before they get to your usual command processor (typically 'command_processor_' or 'abbrev_'). It should examine the a_line to determine whether the command line contains any of the following commands:

   ```
   new_proc
   logout
   probe, pb
   exec_com, ec
   ready_on, rdn
   ready_off, rdf
   restore_cp
   ```

   If one of these commands is input to command_interceptor, it should pass the parameters along to whatever was the previous command processor. If any other command is input, the command_interceptor should print out a message informing the user that the command is not allowed, and should then return.

3. The 'change_env' entry point should establish command_interceptor as the current command processor and the my_ready procedure as the current ready procedure. Note that this procedure should first determine the value of the current command processor and make that value available to the 'command_interceptor' and 'restore_cp' entry points.

4. The 'restore_cp' entry point should restore your usual command processor environment (ie. disable the activity of command_interceptor).

Test out your solutions by executing 'change_env'. You should now be running under the modified command environment. Try typing some disallowed commands and some allowed commands. Especially observe the behavior of ready_on and ready_off. Verify your ability to return to your usual command processor environment by executing 'restore_cp'.

You will probably want to use the following declarations which are in >udd>F15D>s1>include>w2.incl.pl1.

```
   dcl  a_line_ptr ptr;
   dcl  a_line_len fixed bin (21);
   dcl  a_line char (a_line_len) based (a_line_ptr);
   dcl  com_err_ entry options (variable);
   dcl  (ioa_$nnl, ioa_) entry options (variable);
   dcl  (ltrim, rtrim, substr, index, null, codeptr) builtin;
   dcl  cu_$set_ready_procedure entry (entry);
   dcl  cu_$get_command_processor entry (entry);
   dcl  cu_$set_command_processor entry (entry);
   dcl  cu_$get_ready_mode entry (1 aligned, 2 bit (1) unaligned,
                                  2 bit (35) unaligned);
   dcl  old_command_processor entry variable options (variable)
                                                 internal static;
   dcl  command_processor_$command_processor_ entry (ptr,
                           fixed bin (21), fixed bin (35));
   dcl 1 mode aligned,
       2 ready_sw bit (1) unaligned,
       2 mbz bit (35) unaligned;
   dcl  code fixed bin (35);
   dcl  end fixed bin (21);
   dcl test_string char (80) init (" new_proc logout probe pb exec_com
ec ready_on rdn ready_off rdf restore_cp ")
                           internal static options (constant);
```

```
change_env:  proc;

    /*  Variable and subroutine declarations  */

    /*  Get old command processor and old ready procedure and
        store them in internal entries          */

    /*  Set the new command processor and new ready
        procedure.  These both are entries which are internal
        to procedure 'change_env'               */

    /*  Pop the 'change_env' frame off the stack via a
        'return' statement                      */

my_ready:  entry;

    /*  'my_ready' is the entry specified above in the
        subroutine call to set the new ready procedure  */

    /*  check the value of the 'ready_sw'.  If ready_sw is
        true ("1"b) then print your ready message       */

    /*  pop the frame off the stack             */

restore_ready:  proc;

    /*  'restore_ready' should simply make a call to the
        appropriate subroutine to set the ready message back
        to the entry value that was stored above        */

    /*  Pop the frame off the stack             */

command_interceptor:  entry(a_line_ptr, a_line_len, code);

    /*  'command_interceptor' is the entry specified above in
        the call to the subroutine to set the new command
        processor                               */

    /*  Process the command.  If it is a valid command, i.e.
        is a command contained in the variable 'test_string',
        then pass the command to the subroutine which
        accesses the default command processor.  Otherwise,
        print an error message                  */

    /*  Pop the frame off the stack             */

restore_cp:  entry;

    /*  'restore_cp' should simply make a call to the
        appropriate subroutine to set the command processor
        back to the entry value that was stored above   */

    /*  Pop the frame off the stack             */

end change_env;
```

WORKSHOP THREE

Synonyming and IOCB's

1. Use the command 'io_call print_iocb <switchname>' to examine the IOCB's for user_i/o, user_input, user_output and error_output. Verify that the diagram showing synonyming at the end of Topic 6 is correct.

2. Look at the declaration of an 'IOCB' in Topic 6. List the structure members where a synonymed switch's IOCB may differ from the IOCB of the switch to which it is synonymed, assuming no inhibition in effect. There are 7 of them.

3. Refering back to part 1, look carefully at the entry value for iocb.put_chars for all four switches examined. Try doing a 'io_call put_chars <name> hello' using each of the four switches. You should be able to explain exactly what happened, in terms of what IOCB's were referenced, what arguments were passed to which entry point of which I/O module and how the code in that module caused the observed result. If not, ask your instructor.

   What will happen if you execute 'io_call close error_output'? Try it.

4. Detach user_output. Attach it using the syn_ module, however, this time inhibit the close I/O operation.

   'io_call attach user_output syn_ user_i/o -inhibit close'

   Use io_call to examine the IOCB for user_output. What two places do you see evidence that the close operation has been inhibited?

   Try to close user_output.

Using io_call is an excellent way to learn about and test the Multics I/O mechanism. In any remaining time, feel free to experiment, but be aware of the fact that you may get yourself into some situations which require that you hang up the phone to get out of them.

WORKSHOP FOUR

An Interprocess Communication Workshop

The interprocess communication facility is used to coordinate processing between separate processes. In this workshop, you will design and inplement a procedure which will communicate with the instructor's process as follows:

Write a pl1 procedure called 'get_message.pl1' which will:

1) Create an event-call channel specifying that the procedure entry point 'reverse_message' is to be invoked when a wakeup is received on the event-call channel. The handler called 'reverse_message' will be an entry point in the 'get_message' procedure itself (see step 3 below).

2) Obtain the process id and channel id of the instructor's event_call channel by initiating the segment >udd>F15D>s1>channel_info and using the following structure:

```
dcl 1 channel_info_overlay based(channel_info_ptr),
    2 process_id    bit(36),
    2 channel_id    fixed bin (71);
```

for base of segment

Using this information, send a wakeup to the instructor's process providing the channel id of your event_call channel as the 'message'.

3) Your 'reverse_message' handler will be invoked when the instructor's process sends your process a wakeup. This handler should interpret the 'message' from the instructor as a 'char(8)' string; the handler should reverse this string (the pl1 'reverse' builtin function can be used) and finally, your process should send another wakeup to the instructor's process using the reversed string as the 'message'.

Note that the instructor's process will be able to determine whether or not you have successfully accomplished the task. If you have, you will receive immediate notification from the instructor's process (via 'send_mail'). Therefore, make sure you are in 'accept message' mode prior to testing your solution. In addition, use the 'who' command to verify that the instructor's absentee process is indeed running prior to testing your solution.

You will probably want to use the following declarations which are in >udd>F15D>s1>include>w4.incl.pl1.

FSOED >WKSPS> F15D>S1>incl>W4.incl.pl1

```
dcl (ioa_, com_err_) entry options (variable),
     hcs_$wakeup entry (bit (36), fixed bin (71), fixed bin (71),
     fixed bin (35)),
     ipc_$create_ev_chn entry (fixed bin (71), fixed bin (35)),
     ipc_$decl_ev_call_chn entry (fixed bin (71), entry, ptr, fixed bin,
     fixed bin (35)),
     hcs_$initiate entry (char (*), char (*), char (*),
     fixed bin (1), fixed bin (2), ptr, fixed bin (35)),
     hcs_$terminate_noname entry (ptr, fixed bin (35));

dcl  my_chid fixed bin (71),
     c_ptr ptr static,
     i_ptr ptr,
     code fixed bin (35);

dcl 1 channel_info based (c_ptr),
    2 his_pid bit (36),
    2 his_chid fixed bin (71);

dcl 1 event_info based (i_ptr),
    2 channel_id fixed bin (71),
    2 message fixed bin (71),
    2 sender bit (36),
    2 origin,
      3 dev_signal bit (18) unal,
      3 ring bit (18) unal,
    2 data_ptr ptr;

dcl (null, reverse) builtin;
```

Timers

Write a program called lock.pl1 that implements the following ficticious command:

no args = default = 5 mins

USAGE: lock {-min minutes} lock -min 5

FUNCTION: Prompts the user for a password and locks the user's terminal for the number of minutes specified (default = 10 minutes). To regain control of the terminal the user hits the break key and is again prompted for the password. If he does not supply the correct password, the terminal remains locked. = incorrect password

When the time specified expires, the user is logged out.

NOTES: This command prints out the following message:

→ It is MM/DD/YY hhmm.m zzz www.
→ This terminal is locked, please find another terminal.

This message is printed out when lock is first invoked and every 5 minutes thereafter.

Essentially your program should go to sleep for a specified length of time. However, during that period, you must also repeat the message every 5 minutes.

This command takes either 0 or 2 arguments.

For the sake of debugging you should probably use a time interval shorter than 5 minutes and rather than logging out the user, simply print some message.

Sample terminal session:

```
! lock -min 20
  Password:
!            <- user supplies password

  It is 07/21/81  0812.2 mst Tue.
  This terminal is locked, please find another terminal.

  It is 07/21/81  0817.2 mst Tue.
  This terminal is locked, please find another terminal.

! <QUIT>
  Password:
!                <- user gives wrong password

  Incorrect password given.

  It is 07/21/81  0822.2 mst Tue.
  This terminal is locked, please find another terminal.

  It is 07/21/81  0827.2 mst Tue.
  This terminal is locked, please find another terminal.

! <QUIT>
  Password:
!              <- user gives correct password

  Terminal unlocked.
```

```
! lock -min 1
  Password:
!
  It is 07/21/81  0905.1 mst Tue.
  This terminal is locked, please find another terminal.

  The lock time for your terminal has expired.
  You will be logged out.

  NDibble MED logged out 07/21/81 0906.3 mst Tue.
  CPU usage 12 sec, memory usage 97.3 units, cost $3.55.
  hangup
```

You will probably want to use the following declarations which are in >udd>F15D>s1>include>w5.incl.pl1.

```
dcl cu_$arg_count entry (fixed bin, fixed bin(35));
dcl cu_$arg_ptr entry (fixed bin, ptr, fixed bin(21), fixed bin(35));
dcl nargs;
dcl aptr ptr;
dcl arg char(arg_length) based (aptr);
dcl arg_length fixed bin (21);
dcl read_password_ entry (char(*), char(*));
dcl timer_manager_$sleep entry (fixed bin(71), bit(2));
dcl timer_manager_$alarm_call entry (fixed bin(71), bit(2), entry);
dcl timer_manager_$reset_alarm_call entry (entry);
dcl time fixed bin(71) init (10);
dcl cv_dec_check_ entry (char(*), fixed bin(35)) returns(fixed bin(35));
dcl quit condition;
dcl clock_ entry() returns(fixed bin(71));
dcl date_time_ entry (fixed bin(71), char(*));
dcl (password1, password2) char(8);
dcl code fixed bin(35);
dcl (com_err_, ioa_) entry() options(variable);
```

WORKSHOP SIX

Generating Calls and Creating Status Tables

1. Write a procedure called 'please_execute.pl1' which will ask the user, "What do you want me to execute for you? ". The user may then respond with an entryname. The entryname could be a command or a user written program. To keep things simple, no arguments are allowed (ie. the user may respond 'list', but not 'list -d'). Also, you may assume the user's response is both the entryname and the entry point name. Your program should then generate a call to execute the user's request.

   To save time, you will probably want to make a copy of the segment, >udd>F15D>s1>generate_pwd.pl1 and simply edit in the necessary changes.

2. Try out your procedure responding with commands such as 'pwd', 'hmu' and 'list'. Then use it to execute some simple programs you have written or copied into your working directory during this course. You might even like to ask 'please_execute' to execute 'please_execute'.

3. Ask 'please_execute' to execute the 'list' command and when it starts to list your segments, hit the break/quit/interrupt key. Use the 'stack' request from within 'probe' to examine the user stack.

NUMBERS 4 AND 5 ARE OPTIONAL

4. Next ask 'please_execute' to execute some program that does not exist (make up a name). Notice the error message returned when your program calls com_err_.

5. Create a status table and alter your program so that when you ask your program to execute a program that does not exist, it will still call com_err_, however, it will not behave as in part 4. Instead it should print some other error message (one you have made up and put in a status table you have built).

The Limited Service Subsystem

In workshop two you modified the user's environment such that only certain commands were acceptable. Utilize the 'enter_lss' command to achieve the same result. The following list of commands should be accepted (you may expand upon this list if you want to):

new_proc
logout
probe, pb
exec_com, ec
ready_on, rdn
ready_off, rdf

Test your solution.

Large Information Systems Div, Honeywell Information Systems, Inc.
Multics Computer Center, Phoenix Az.

Printout of the 5 Entries
of the
handout Library

Which Match the Search Name
gw.archive

Printed on: 04/06/81 0804.0
Printed by: NDibble.MED.a
Descriptor: handout_desc

```
        COMPILATION LISTING OF SEGMENT pfgu
        Compiled by: Multics PL/I Compiler, Release 25c, of February 18, 1980
        Compiled at: Honeywell LISD Phoenix, System M
        Compiled on: 03/19/80  1043.5 mst Wed
            Options: map

      1 find: proc (last_name, first_name, emp_struc, code);
      2
      3 /*  Modified by R. Frommer on June 12, 1979 to do elegant ORing, reference mode bits directly, and use -extend option */
      4 /* THESE DECLARATIONS SHARED BY BOTH ENTRYPOINTS */
      5
      6 dcl  acs_names_array (4) char (32) internal static options (constant)
      7      init ("soc_sec_no.acs", "manager.acs", "mail_station.acs", "salary.acs");
      8 dcl  access_allowed (4) bit (1);
      9 dcl  access_allowed_overlay bit(4) unaligned defined (access_allowed);
     10 dcl  dir_name char (168);
     11 dcl  user_id char (32);
     12 dcl  ring fixed bin (17);
     13 dcl  mode fixed bin (5);
     14 dcl 1 bit_mode_struc aligned based (mode_ptr),
     15      2 pad bit (30) unaligned,
     16      2 two_bits bit(2) unaligned, /* don't care */
     17      2 read bit (1) unaligned,    /* On if read effective access */
     18      2 fourth bit (1) unaligned, /* don't care */
     19      2 write bit(1) unaligned; /* On if write effective access */
     20 dcl  ME char (4);
     21 dcl  find bit (1);
     22 dcl 1 emp_struc,
     23      2 soc_sec_no pic "(9)9",
     24      2 manager pic "(9)9",
     25      2 mail_station char (3),
     26      2 salary fixed dec (8, 2);
     27 dcl (code, dummy_code) fixed bin (35);
     28 dcl (first_name, last_name) char (15);
     29 dcl (substr, null, addr, size, rtrim) builtin;
     30 dcl  error_table_$moderr ext static fixed bin (35);
     31 dcl  error_table_$no_record ext static fixed bin (35);
     32 dcl  i fixed bin;
     33 dcl (iocbp, mode_ptr) ptr init (null ());
     34 dcl  unused_rec_len fixed bin (21);
     35 dcl  old_level fixed bin;
     36
  1   1 dcl
  1   2 iox_$attach_ptr entry (ptr, char(*), ptr, fixed bin (35)),
  1   3 iox_$attach_name entry (char(*), ptr, char(*), ptr, fixed bin (35)),
  1   4 iox_$close entry (ptr, fixed bin (35)),
  1   5 iox_$control entry (ptr, char(*), ptr, fixed bin(35)),
  1   6 iox_$delete_record entry (ptr, fixed bin(35)),
  1   7 iox_$detach_iocb entry (ptr, fixed bin(35)),
```

```
1      8 iox_$find_iocb entry (char(*), ptr, fixed bin(35)),
1      9 iox_$get_chars entry (ptr, ptr, fixed bin(21), fixed bin(21), fixed bin(35)),
1     10 iox_$get_line entry (ptr, ptr, fixed bin(21), fixed bin (21), fixed bin (35)),
1     11 iox_$modes entry (ptr, char(*), char(*), fixed bin (35)),
1     12 iox_$move_attach entry (ptr, ptr, fixed bin(35)),
1     13 iox_$open entry (ptr, fixed bin, bit(1) aligned, fixed bin(35)),
1     14 iox_$position entry (ptr, fixed bin, fixed bin(21), fixed bin (35)),
1     15 iox_$put_chars entry (ptr, ptr, fixed bin(21), fixed bin(35)),
1     16 iox_$read_key entry (ptr, char(256) varying, fixed bin(21), fixed bin (35)),
1     17 iox_$read_length entry (ptr, fixed bin(21), fixed bin(35)),
1     18 iox_$read_record entry (ptr, ptr, fixed bin(21), fixed bin(21), fixed bin (35)),
1     19 iox_$rewrite_record entry (ptr, ptr, fixed bin (21), fixed bin(35)),
1     20 iox_$seek_key entry (ptr, char(256) varying, fixed bin(21), fixed bin(35)),
1     21 iox_$write_record entry (ptr, ptr, fixed bin(21), fixed bin (35)),
1     22 iox_$destroy_iocb entry (ptr, fixed bin(35));
      37
      38
      39 dcl  cu_$level_get entry (fixed bin);
      40 dcl  cu_$level_set entry (fixed bin);
      41 dcl  get_ring_ entry () returns (fixed bin (3));
      42 dcl  hcs_$get_user_effmode entry (char (*), char (*), char (*), fixed bin, fixed bin (5), fixed bin (35));
      43 dcl  get_group_id_$tag_star entry () returns (char (32));
      44
      45           ME = "find";
      46           find = "1"b;
      47           goto COMMON_CODE;
      48
      49 put:     entry (last_name, first_name, emp_struc, code);
      50           ME = "put";
      51           find = "0"b;
      52
      53 COMMON_CODE:
      54
      55           call cu_$level_get (old_level);
      56           ring = get_ring_ ();
      57           call cu_$level_set (ring);
      58
      59           code = 0;
      60
      61           dir_name = ">udd>F15d>s1";
      62           user_id = get_group_id_$tag_star ();
      63           call hcs_$get_user_effmode (dir_name, "emp", user_id, ring, mode, code);
      64           if code ^= 0 then go to return_point;
      65           mode_ptr = addr (mode);
      66           if
      67           (^find & ^(bit_mode_struc.read & bit_mode_struc.write)) |  /* if put, we must have read and write on emp */
      68           (find & ^bit_mode_struc.read) then do;           /* If find, we must have read on emp */
      69                code = error_table_$moderr;
      70                goto return_point;
      71           end;
      72
      73 /* If we get here,  then it must be the case that we have status on
      74*   our current working directory. */
      75
      76 /*  Obtain the effective access on the access control segments of interest */
```

```
 77
 78          do i = 1 to 4;                                  /* Once for each access control seg */
 79
 80               call hcs_$get_user_effmode (dir_name, acs_names_array (i), user_id, ring, mode, code);
 81               if find then access_allowed (i) = bit_mode_struc.read;
 82               else access_allowed (i) = bit_mode_struc.write;
 83          end;
 84
 85 /* ONE OF TWO BLOCKS OF CODE, DEPENDING UPON ENTRYPOINT TAKEN... */
 86
 87          if find then do;
 88
 89               if access_allowed_overlay then do;          /* elegant ORing */
 90                    call iox_$find_iocb ("emp_sw", iocbp, code);
 91                    call iox_$attach_ptr (iocbp, "vfile_ " || rtrim (dir_name) || ">emp", null (), code);
 92                    call iox_$open (iocbp, 11 /* direct input */, "0"b, code);
 93                    call iox_$seek_key (iocbp, (last_name || first_name), 4 * size (emp_struc), code);
 94                    if code = 0 then do;
 95                         call iox_$read_record (iocbp, addr (emp_struc), 4 *size (emp_struc), unused_rec_len, code);
 96                         if ^access_allowed (1) then soc_sec_no = 0;
 97                         if ^access_allowed (2) then manager = 0;
 98                         if ^access_allowed (3) then mail_station = "";
 99                         if ^access_allowed (4) then salary = 0;
100                    end;
101                    else;                                 /* else nothing */
102                    call iox_$close (iocbp, dummy_code);
103                    call iox_$detach_iocb (iocbp, dummy_code);
104                    goto return_point;                    /* To reset validation level */
105               end;
106
107               else code = 1;                             /* This special code indicates that no fields were accessible
\c*/
108
109          end;
110          else do;
111
112               if access_allowed (1) & access_allowed (2) & access_allowed (3) & access_allowed (4) then do; /* If all of
\cthem are write-able */
113                    call iox_$find_iocb ("emp_sw", iocbp, code);
114                    call iox_$attach_ptr (iocbp, "vfile_ " || rtrim (dir_name) || ">emp -extend", null (), code);
115                    call iox_$open (iocbp, 12 /* direct output */, "0"b, code);
116                    call iox_$seek_key (iocbp, (last_name || first_name), 4 * size (emp_struc), code);
117                    if code = error_table_$no_record then call iox_$write_record (iocbp, addr (emp_struc), 4*size (emp_str
\cuc), code);
118                    call iox_$close (iocbp, dummy_code);
119                    call iox_$detach_iocb (iocbp, dummy_code);
120                    goto return_point;
121
122               end;
123
124               else code = 1;                             /* This special code indicates that some fields were inaccessi
\cble */
125
126 /* FALL THROUGH TO return_point */
127          end;
```

```
128
129 return_point:
130           call cu_$level_set (old_level);
131      end find;
```

SOURCE FILES USED IN THIS COMPILATION.

LINE	NUMBER	DATE MODIFIED	NAME	PATHNAME
	0	07/19/79 1110.0	pfgu.pl1	>user_dir_dir>F15D>Student_01>pfgu.pl1
37	1	02/13/79 1714.4	dcl_iox_entries.incl.pl1	>user_dir_dir>F15D>Student_01>dcl_iox_entries.incl.pl1

NAMES DECLARED IN THIS COMPILATION.

IDENTIFIER	OFFSET	LOC	STORAGE CLASS	DATA TYPE	ATTRIBUTES AND REFERENCES (* indicates a set context)
NAMES DECLARED BY DECLARE STATEMENT.					
ME		000165	automatic	char(4)	unaligned dcl 20 set ref 45* 50*
access_allowed		000100	automatic	bit(1)	array unaligned dcl 8 set ref 81* 82* 89 89 96 97 98 99 112 112 112 112
access_allowed_overlay			defined	bit(4)	unaligned dcl 9 ref 89
acs_names_array		000000	constant	char(32)	initial array unaligned dcl 6 set ref 80*
addr				builtin function	dcl 29 ref 65 95 95 117 117
bit_mode_struc			based	structure	level 1 dcl 14
code			parameter	fixed bin(35,0)	dcl 27 set ref 1 49 59* 63* 64 69* 80* 90* 91* 92* 93* 94 95* 107* 113* 114* 115* 116* 117 117* 124*
cu_$level_get		000034	constant	entry	external dcl 39 ref 53
cu_$level_set		000036	constant	entry	external dcl 40 ref 57 129
dir_name		000101	automatic	char(168)	unaligned dcl 10 set ref 61* 63* 80* 91 114
dummy_code		000167	automatic	fixed bin(35,0)	dcl 27 set ref 102* 103* 118* 119*
emp_struc			parameter	structure	level 1 unaligned dcl 22 set ref 1 49 93 95 95 95 116 117 117 117
error_table_$moderr		000010	external static	fixed bin(35,0)	dcl 30 ref 69
error_table_$no_record		000012	external static	fixed bin(35,0)	dcl 31 ref 117
find		000166	automatic	bit(1)	unaligned dcl 21 set ref 46* 51* 66 66 81 87
first_name			parameter	char(15)	unaligned dcl 28 ref 1 49 93 116
get_group_id_$tag_star		000044	constant	entry	external dcl 43 ref 62
get_ring_		000040	constant	entry	external dcl 41 ref 56
hcs_$get_user_effmode		000042	constant	entry	external dcl 42 ref 63 80
i		000170	automatic	fixed bin(17,0)	dcl 32 set ref 78* 80 81 82*
iocbp		000172	automatic	pointer	initial dcl 33 set ref 33* 90* 91* 92* 93* 95* 102* 103* 113* 114* 115* 116* 117* 118* 119*
iox_$attach_ptr		000014	constant	entry	external dcl 1-1 ref 91 114
iox_$close		000016	constant	entry	external dcl 1-1 ref 102 118
iox_$detach_iocb		000020	constant	entry	external dcl 1-1 ref 103 119
iox_$find_iocb		000022	constant	entry	external dcl 1-1 ref 90 113
iox_$open		000024	constant	entry	external dcl 1-1 ref 92 115
iox_$read_record		000026	constant	entry	external dcl 1-1 ref 95
iox_$seek_key		000030	constant	entry	external dcl 1-1 ref 93 116
iox_$write_record		000032	constant	entry	external dcl 1-1 ref 117
last_name			parameter	char(15)	unaligned dcl 28 ref 1 49 93 116
mail_station	4(18)		parameter	char(3)	level 2 packed unaligned dcl 22 set ref 98*
manager	2(09)		parameter	picture(9)	level 2 packed unaligned dcl 22 set ref 97*
mode		000164	automatic	fixed bin(5,0)	dcl 13 set ref 63* 65 80*
mode_ptr		000174	automatic	pointer	initial dcl 33 set ref 33* 65* 66 66 66 81 82
null				builtin function	dcl 29 ref 33 33 91 91 114 114
old_level		000177	automatic	fixed bin(17,0)	dcl 35 set ref 53* 129*
read	0(32)		based	bit(1)	level 2 packed unaligned dcl 14 ref 66 66 81
ring		000163	automatic	fixed bin(17,0)	dcl 12 set ref 56* 57* 63* 80*
rtrim				builtin function	dcl 29 ref 91 114
salary	6		parameter	fixed dec(8,2)	level 2 dcl 22 set ref 99*
size				builtin function	dcl 29 ref 93 95 116 117
soc_sec_no			parameter	picture(9)	level 2 packed unaligned dcl 22 set ref 96*
unused_rec_len		000176	automatic	fixed bin(21,0)	dcl 34 set ref 95*
user_id		000153	automatic	char(32)	unaligned dcl 11 set ref 62* 63* 80*
write	0(34)		based	bit(1)	level 2 packed unaligned dcl 14 ref 66 82

```
NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.
iox_$attach_name                000000 constant        entry                    external dcl 1-1
iox_$control                    000000 constant        entry                    external dcl 1-1
iox_$delete_record              000000 constant        entry                    external dcl 1-1
iox_$destroy_iocb               000000 constant        entry                    external dcl 1-1
iox_$get_chars                  000000 constant        entry                    external dcl 1-1
iox_$get_line                   000000 constant        entry                    external dcl 1-1
iox_$modes                      000000 constant        entry                    external dcl 1-1
iox_$move_attach                000000 constant        entry                    external dcl 1-1
iox_$position                   000000 constant        entry                    external dcl 1-1
iox_$put_chars                  000000 constant        entry                    external dcl 1-1
iox_$read_key                   000000 constant        entry                    external dcl 1-1
iox_$read_length                000000 constant        entry                    external dcl 1-1
iox_$rewrite_record             000000 constant        entry                    external dcl 1-1
substr                                                 builtin function         dcl 29

NAMES DECLARED BY EXPLICIT CONTEXT.
COMMON_CODE                     000145 constant        label                    dcl 53 ref 47
find                            000117 constant        entry                    external dcl 1
put                             000134 constant        entry                    external dcl 49
return_point                    001200 constant        label                    dcl 129 ref 64 70 104 120

THERE WERE NO NAMES DECLARED BY CONTEXT OR IMPLICATION.

STORAGE REQUIREMENTS FOR THIS PROGRAM.

          Object     Text      Link      Symbol     Defs      Static
Start          0        0      1414        1462     1214        1424
Length      1700     1214        46         201      200           0

BLOCK NAME                    STACK SIZE     TYPE              WHY NONQUICK/WHO SHARES STACK FRAME
find                                 233 external procedure  is an external procedure.

STORAGE FOR AUTOMATIC VARIABLES.

STACK FRAME                      LOC IDENTIFIER                BLOCK NAME
find                          000100 access_allowed            find
                              000101 dir_name                  find
                              000153 user_id                   find
                              000163 ring                      find
                              000164 mode                      find
                              000165 ME                        find
                              000166 find                      find
                              000167 dummy_code                find
                              000170 i                         find
                              000172 iocbp                     find
                              000174 mode_ptr                  find
                              000176 unused_rec_len            find
                              000177 old_level                 find

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.
alloc_cs            cat_realloc_cs      call_ext_out_desc   call_ext_out        return              shorten_stack
ext_entry
```

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

cu_$level_get	cu_$level_set	get_group_id_$tag_star	get_ring_
hcs_$get_user_effmode	iox_$attach_ptr	iox_$close	iox_$detach_iocb
iox_$find_iocb	iox_$open	iox_$read_record	iox_$seek_key
iox_$write_record			

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

error_table_$moderr	error_table_$no_record

LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC
33	000105	1	000112	45	000125	46	000127	47	000131	49	000132	50	000142
51	000144	53	000145	56	000154	57	000165	59	000174	61	000176	62	000201
63	000210	64	000250	65	000254	66	000256	69	000303	70	000306	78	000307
80	000314	81	000352	82	000362	83	000367	87	000371	89	000373	90	000376
91	000422	92	000507	93	000532	94	000570	95	000574	96	000617	97	000630
98	000641	99	000651	102	000661	103	000672	104	000703	107	000704	109	000707
112	000710	113	000732	114	000756	115	001043	116	001066	117	001124	118	001152
119	001163	120	001174	124	001175	129	001200	131	001207				

```
COMPILATION LISTING OF SEGMENT put
Compiled by: Multics PL/I Compiler, Release 25c, of February 18, 1980
Compiled at: Honeywell LISD Phoenix, System M
Compiled on: 03/19/80  1039.0 mst Wed
    Options: map

 1 /*  Command interface to the put_find_gate_ gate entrypoints */
 2
 3 /*  Modified on June 12, 1979 by R. Frommer to produce better diagnostic messages.  */
 4 find: proc;
 5
 6 /* DECLARATIONS */
 7
 8 dcl  ME char (4) varying;
 9 dcl  nargs fixed bin;
10 dcl  argl fixed bin (21);
11 dcl (first_name, last_name) char (15);
12 dcl  code fixed bin (35);
13 dcl  arg char (argl) based (argp);
14 dcl  argp ptr;
15 dcl 1 emp_struc,
16     2 soc_sec_no pic "(9)9",
17     2 manager pic "(9)9",
18     2 mail_station char (3),
19     2 salary fixed dec (8, 2);
20 dcl (error_table_$bigarg, error_table_$wrong_no_of_args) ext static fixed bin (35);
21 dcl (fixedoverflow, conversion, size) condition;
22
23 dcl  cu_$arg_ptr entry (fixed bin, ptr, fixed bin (21), fixed bin (35));
24 dcl  cu_$arg_count entry (fixed bin);
25 dcl (com_err_, ioa_) entry options (variable);
26 dcl (put_find_gate_$find, put_find_gate_$put) entry
27     (char (15), char (15), 1 structure, 2 pic "(9)9" member, 2 pic "(9)9" member,
28      2 char (3) member, 2 fixed dec (8, 2) member, fixed bin (35));
29 dcl  cv_dec_check_ entry (char(*), fixed bin(35)) returns (fixed bin(35));
30
31          ME = "find";
32          call cu_$arg_count (nargs);
33          if nargs ^= 2 then do;
34               call com_err_ (error_table_$wrong_no_of_args, ME,
35                    "^/Correct invocation:  find First_name Last_name");
36               return;
37          end;
38          call cu_$arg_ptr (1, argp, argl, code);
39          if argl > 15 then do;
40               call com_err_ (error_table_$bigarg, ME, "^a exceeds 15 characters.", arg);
41               return;
42          end;
43          else first_name = arg;
```

```
44          call cu_$arg_ptr (2, argp, argl, code);
45          if argl > 15 then do;
46               call com_err_ (error_table_$bigarg, ME, "^a exceeds 15 characters.", arg);
47               return;
48          end;
49          else last_name = arg;
50
51 /* Initialize emp_struc prior to calling the gate entrypoint */
52
53          soc_sec_no, manager, salary = 0;
54          mail_station = "";
55
56          call put_find_gate_$find (last_name, first_name, emp_struc, code);
57          if code = 1 /* this is a special case */ then
58               call com_err_ (0, ME, "You don't have proper access on the fields in the employee file.");
59          else if code ^= 0 then call com_err_ (code, ME);
60          else call ioa_ ("^[^s^;SOCIAL SECURITY NUMBER = ^i
61 ^]^[^s^;MANAGER'S SOCIAL SECURITY NUMBER = ^i
62 ^]^[^s^;MAIL STATION = ^a
63 ^]^[^s^;SALARY = $^i
64 ^]", (soc_sec_no = 0), soc_sec_no,
65               (manager = 0), manager,
66               (mail_station = ""), mail_station,
67               (salary = 0), salary);
68          return;                                        /* END OF find PROCEDURE */
69
70 put:     entry ();
71
72          ME = "put";
73          call cu_$arg_count (nargs);
74          if nargs ^= 6 then do;
75               call com_err_ (error_table_$wrong_no_of_args, ME,
76                    "^/Correct invocation: put First_name Last_name Employee's_ss_no Manager's_ss_no Mail_Station Salary")
\c;
77               return;
78          end;
79
80 /* OBTAIN SIX REQUIRED ARGUMENTS */
81
82          call cu_$arg_ptr (1, argp, argl, code);
83          if argl > 15 then do;
84               call com_err_ (error_table_$bigarg, ME, "^a", arg);
85               return;
86          end;
87          else first_name = arg;
88
89          call cu_$arg_ptr (2, argp, argl, code);
90          if argl > 15 then do;
91               call com_err_ (error_table_$bigarg, ME, "^a", arg);
92               return;
93          end;
94          else last_name = arg;
95
96          call cu_$arg_ptr (3, argp, argl, code);
97          if argl ^= 9 | verify (arg, "0123456789") ^= 0 then do;
```

```
 98               call com_err_ (0, ME, "^a is not a suitable social security number for ^a.",
 99                    arg, last_name);
100               return;
101          end;
102          else soc_sec_no = cv_dec_check_(arg, code);
103
104          call cu_$arg_ptr (4, argp, argl, code);
105          if argl ^= 9 | verify (arg, "0123456789") ^= 0 then do;
106               call com_err_ (0, ME, "^a is not a suitable social security number for ^a's manager.", arg, last_name);
107               return;
108          end;
109          else manager = cv_dec_check_(arg, code);
110
111          call cu_$arg_ptr (5, argp, argl, code);
112          if argl ^= 3 then do;
113               call com_err_ (0, ME, "^a is not a suitable 3-character mail drop.", arg);
114               return;
115          end;
116          else mail_station = arg;
117
118          call cu_$arg_ptr (6, argp, argl, code);
119
120          on fixedoverflow, size begin;
121               call com_err_ (0, ME, "^a is not a proper salary. It must resemble 999999.99", arg);
122               goto bottom;
123          end;
124          salary = cv_dec_check_(arg, code);
125          if code ^= 0 then do;
126               call com_err_ (0, ME, "^a is not a proper salary. It must resemble 999999.99", arg);
127               goto bottom;
128          end;
129
130          call put_find_gate_$put (last_name, first_name, emp_struc, code);
131          if code = 1 then call com_err_ (0, ME, "You lack access on one or more of the fields in the database.");
132          else if code ^= 0 then call com_err_ (code, ME);
133          else call ioa_ ("Record written.");
134
135 bottom:
136          return;
137
138     end find;
```

SOURCE FILES USED IN THIS COMPILATION.

LINE NUMBER	DATE MODIFIED	NAME	PATHNAME
0	07/19/79 1109.5	put.pl1	>udd>F15D>Student_01>put.pl1

NAMES DECLARED IN THIS COMPILATION.

IDENTIFIER	OFFSET	LOC	STORAGE CLASS	DATA TYPE	ATTRIBUTES AND REFERENCES (* indicates a set context)
NAMES DECLARED BY DECLARE STATEMENT.					
ME		000100	automatic	varying char(4)	dcl 8 set ref 31* 34* 40* 46* 57* 59* 72* 75* 84* 91* 98* 106* 113* 121* 126* 131* 132*
arg			based	char	unaligned dcl 13 set ref 40* 43 46* 49 84* 87 91* 94 97 98* 102* 105 106* 109* 113* 116 121* 124* 126*
argl		000103	automatic	fixed bin(21,0)	dcl 10 set ref 38* 39 40 40 43 44* 45 46 46 49 82* 83 84 84 87 89* 90 91 91 94 96* 97 97 98 98 102 102 104* 105 105 106 106 109 109 111* 112 113 113 116 118* 121 121 124 124 126 126
argp		000116	automatic	pointer	dcl 14 set ref 38* 40 43 44* 46 49 82* 84 87 89* 91 94 96* 97 98 102 104* 105 106 109 111* 113 116 118* 121 124 126
code		000114	automatic	fixed bin(35,0)	dcl 12 set ref 38* 44* 56* 57 59 59* 82* 89* 96* 102* 104* 109* 111* 118* 124* 125 130* 131 132 132*
com_err_		000020	constant	entry	external dcl 25 ref 34 40 46 57 59 75 84 91 98 106 113 121 126 131 132
cu_$arg_count		000016	constant	entry	external dcl 24 ref 32 73
cu_$arg_ptr		000014	constant	entry	external dcl 23 ref 38 44 82 89 96 104 111 118
cv_dec_check_		000030	constant	entry	external dcl 29 ref 102 109 124
emp_struc		000120	automatic	structure	level 1 unaligned dcl 15 set ref 56* 130*
error_table_$bigarg		000010	external static	fixed bin(35,0)	dcl 20 set ref 40* 46* 84* 91*
error_table_$wrong_no_of_args		000012	external static	fixed bin(35,0)	dcl 20 set ref 34* 75*
first_name		000104	automatic	char(15)	unaligned dcl 11 set ref 43* 56* 87* 130*
fixedoverflow		000132	stack reference	condition	dcl 21 ref 120
ioa_		000022	constant	entry	external dcl 25 ref 60 133
last_name		000110	automatic	char(15)	unaligned dcl 11 set ref 49* 56* 94* 98* 106* 130*
mail_station	4(18)	000120	automatic	char(3)	level 2 packed unaligned dcl 15 set ref 54* 60 60* 116*
manager	2(09)	000120	automatic	picture(9)	level 2 packed unaligned dcl 15 set ref 53* 60 60* 109*
nargs		000102	automatic	fixed bin(17,0)	dcl 9 set ref 32* 33 73* 74
put_find_gate_$find		000024	constant	entry	external dcl 26 ref 56
put_find_gate_$put		000026	constant	entry	external dcl 26 ref 130
salary	6	000120	automatic	fixed dec(8,2)	level 2 dcl 15 set ref 53* 60 60* 124*
size		000140	stack reference	condition	dcl 21 ref 120
soc_sec_no		000120	automatic	picture(9)	level 2 packed unaligned dcl 15 set ref 53* 60 60* 102*
NAME DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.					
conversion		000000	stack reference	condition	dcl 21
NAMES DECLARED BY EXPLICIT CONTEXT.					
bottom		002025	constant	label	dcl 135 ref 122 127
find		000315	constant	entry	external dcl 4
put		000753	constant	entry	external dcl 70
NAME DECLARED BY CONTEXT OR IMPLICATION.					
verify				builtin function	ref 97 105

STORAGE REQUIREMENTS FOR THIS PROGRAM.

	Object	Text	Link	Symbol	Defs	Static
Start	0	0	2364	2416	2237	2374
Length	2610	2237	32	155	125	0

BLOCK NAME	STACK SIZE	TYPE	WHY NONQUICK/WHO SHARES STACK FRAME
find	226	external procedure	is an external procedure.
on unit on line 120	98	on unit	

STORAGE FOR AUTOMATIC VARIABLES.

STACK FRAME	LOC	IDENTIFIER	BLOCK NAME
find	000100	ME	find
	000102	nargs	find
	000103	argl	find
	000104	first_name	find
	000110	last_name	find
	000114	code	find
	000116	argp	find
	000120	emp_struc	find

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.

r_e_as	r_ne_as	call_ext_out_desc	call_ext_out	return	tra_ext
enable	ext_entry	int_entry	unpack_ptr		

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

com_err_	cu_$arg_count	cu_$arg_ptr	cv_dec_check_
ioa_	put_find_gate_$find	put_find_gate_$put	

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

error_table_$bigarg	error_table_$wrong_no_of_args

LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC
4	000314	31	000322	32	000326	33	000334	34	000337	36	000363	38	000364
39	000403	40	000406	41	000440	43	000441	44	000445	45	000464	46	000467
47	000521	49	000522	53	000526	54	000541	56	000544	57	000561	59	000612
60	000632	68	000751	70	000752	72	000760	73	000764	74	000773	75	000776
77	001022	82	001023	83	001042	84	001045	85	001076	87	001077	89	001103
90	001122	91	001125	92	001156	94	001157	96	001163	97	001202	98	001224
100	001263	102	001264	104	001315	105	001334	106	001356	107	001415	109	001416
111	001447	112	001466	113	001471	114	001524	116	001525	118	001531	120	001550
121	001564	122	001620	120	001623	124	001630	125	001661	126	001663	127	001717
130	001720	131	001735	132	001766	133	002006	135	002025				

```
"       Gate procedure written by Raphael Frommer of Marketing Education
"       to be used in conjunction with the "find" and "put" command
"       procedures.  Transfers control to the pl1 procedure "put_find_gate_util_.
"
        include   gate_macros
        gate_info
        gate      find,put_find_gate_util_,find,4
        gate      put,put_find_gate_util_,put,4
        end
```

```
ASSEMBLY LISTING OF SEGMENT >udd>F15D>s1>put_find_gate_.alm
ASSEMBLED ON:       03/18/80  1557.0 mst Tue
OPTIONS USED:       list
ASSEMBLED BY:       ALM Version 6.2, June 1979
ASSEMBLER CREATED:  01/28/80  0927.8 mst Mon

                          1  "         Gate procedure written by Raphael Frommer of Marketing Education
                          2  "         to be used in conjunction with the "find" and "put" command
                          3  "         procedures.  Transfers control to the pl1 procedure "put_find_gate_util_.
                          4  "
                          5           include   gate_macros
                        1-1
                        1-2  " BEGIN INCLUDE FILE ..... gate_macros.incl.alm
                        1-3  "   Last modified 6/77 by N. Morris, B. Greenberg, & T. VanVleck
                        1-4  "   Modified 741212 by PG to inhibit while computing virtual CPU time
                        1-5
                        1-6  " This file contains several macros used when generating gate segments.
                        1-7  " The major macros that are defined are:
                        1-8  "
                        1-9  "         gate_info           general setup code for normal gates
                       1-10  "         hardcore_gate_info  general setup code for hardcore gates
                       1-11  "         bad_dir_handler     setup and handler for bad_dir_ condition (goes at bottom)
                       1-12  "         bad_dir_handler_entry        entrypoint for above (goes at top)
                       1-13  "         gate                to define a normal gate
                       1-14  "         hgate               to define a hardcore gate
                       1-15  "         fgate               to define a fast hardcore gate
                       1-16  "
                       1-17
                       1-18
000000                 1-19           maclist   off
                       1-20
                       1-21
```

```
    1-22  " HGATE - define a hardcore gate entry
    1-23  "
    1-24  "         hgate     gatename,procedure,entry,args[,bad_dir_trap]
    1-25  "
    1-26  " The entrypoint gatename is defined in the gate segment.  If args
    1-27  " is nonzero, the number of arguments passed to gatename must be
    1-28  " equal to args.  When gatename is called, it will in turn call
    1-29  " procedure$entry.
    1-30  "
    1-31
    1-32  macro     hgate
  1 1-33            gentry    &1,&4*2,&1.t
  2 1-34            tsx2      .setup
  3 1-35  &=&5,bad_dir_trap&[ tsx0      .set_dir_trap
  4 1-36  &]        short_call &2$&3(ap|0)
  5 1-37            eppbp     lp|&1.t
  6 1-38            tra       .return_code
  7 1-39            maclist   restore
  8 1-40            use       linkage
  9 1-41            even
 10 1-42            maclist   on,save
 11 1-43  &1.t:
 12 1-44            bss       ,6
 13 1-45
 14 1-46            maclist   restore
 15 1-47  &end
    1-48
    1-49
    1-50  " FGATE - define a fast hardcore gate
    1-51  "
    1-52  "         fgate     gatename,procedure,entry
    1-53  "
    1-54
    1-55  macro     fgate
  1 1-56            gentry    &1,0,0
  2 1-57            epplp     .my_lp,*
  3 1-58            tra       &2$&3
  4 1-59
  5 1-60            maclist   off
  6 1-61  &end
    1-62
    1-63
    1-64  " GATE - define a normal gate entry
    1-65  "
    1-66  "         gate      gatename,procedure,entry,args
    1-67  "
    1-68
    1-69  macro     gate
  1 1-70            gentry    &1,&4*2,0
  2 1-71            tsx2      .setup
  3 1-72            short_call &2$&3(ap|0)
  4 1-73            return
  5 1-74
  6 1-75            maclist   restore
```

```
7    1-76  &end
     1-77
     1-78
```

```
     1-79
     1-80  macro     gentry
  1  1-81            maclist   on,save
  2  1-82            segdef    &1
  3  1-83            maclist   restore
  4  1-84            use       transfer_vector
  5  1-85            maclist   on,save
  6  1-86  &1:
  7  1-87            tra       &1.e
  8  1-88            maclist   restore
  9  1-89            use       main
 10  1-90            maclist   on,save
 11  1-91            zero      &2,&3
 12  1-92  &1.e:
 13  1-93  &end
     1-94
     1-95
```

```
    1-96  " HARDCORE_GATE_INFO - general info for hardcore gates
    1-97
    1-98  macro     hardcore_gate_info
 1  1-99            maclist   on,save
 2 1-100            name      &1
 3 1-101
 4 1-102            include   stack_header
 5 1-103
 6 1-104            include   stack_frame
 7 1-105
 8 1-106            maclist   restore
 9 1-107            eject
10 1-108
11 1-109            tempd     .temp
12 1-110            tempd     .label_variable(0)
13 1-111            tempd     .time1,.time2
14 1-112            tempd     .unwinder_arglist(0)
15 1-113            tempd     .on_unit(5)
16 1-114            temp      .pf,.entryp
17 1-115            tempd     .vfl_arglist(2)
18 1-116            tempd     .mcptr
19 1-117
20 1-118            use       transfer_vector
21 1-119            equ       .tv_begin,*
22 1-120
23 1-121            tra       .actor
24 1-122
25 1-123            use       tv_end
26 1-124
27 1-125            segdef    .tv_end
28 1-126  .tv_end:  vfd       14/(*-.tv_begin)
29 1-127
30 1-128            use       main
31 1-129
32 1-130            segdef    .my_lp
33 1-131            even
34 1-132  .my_lp:   bss       ,2
35 1-133
36 1-134            join      /text/transfer_vector,tv_end,main
37 1-135
38 1-136            use       linkage
39 1-137            join      /link/linkage
40 1-138
41 1-139            use       main
42 1-140
43 1-141            maclist   restore
44 1-142            eject
45 1-143
46 1-144  .actor:   epplp     .my_lp,*
47 1-145            maclist   restore
48 1-146            gate_actor
49 1-147
50 1-148
```

```
51   1-149
52   1-150  .setup:   push
53   1-151           epplp     .my_lp,*
54   1-152           sprilp    sp|stack_frame.lp_ptr
55   1-153
56   1-154           maclist   restore
57   1-155           gcheck
58   1-156
59   1-157           inhibit   on        <+><+><+><+><+><+><+><+><+><+><+><+><+>
60   1-158           rccl      sys_info$clock_,*   calculate times
61   1-159           sbaq      pds$cpu_time
62   1-160           staq      .time1
63   1-161           sbaq      pds$virtual_delta
64   1-162           staq      .time2
65   1-163           lda       pds$page_waits
66   1-164           sta       .pf
67   1-165           inhibit   off       <->><->><->><->><->><->><->><->><->><->
68   1-166           tra       0,2
69   1-167
70   1-168           maclist   restore
71   1-169           eject
72   1-170
73   1-171  .return_code:
74   1-172           inhibit   on        <+><+><+><+><+><+><+><+><+><+><+><+><+>
75   1-173           rccl      sys_info$clock_,*   calculate times
76   1-174           sbaq      pds$cpu_time
77   1-175           staq      .temp
78   1-176           sbaq      .time1
79   1-177           adaq      bp|0
80   1-178           staq      bp|0
81   1-179           ldaq      .temp
82   1-180           sbaq      pds$virtual_delta
83   1-181           sbaq      .time2
84   1-182           adaq      bp|2
85   1-183           staq      bp|2
86   1-184           lda       pds$page_waits
87   1-185           sbla      .pf
88   1-186           asa       bp|4
89   1-187           aos       bp|5
90   1-188           inhibit   off       <->><->><->><->><->><->><->><->><->><->
91   1-189
92   1-190           return
93   1-191
94   1-192           maclist   restore
95   1-193           eject
96   1-194           maclist   restore
97   1-195  &end
     1-196
     1-197
```

```
    1-198  " BAD_DIR_HANDLER - code to setup and handle bad_dir_ condition
    1-199  "                          put this after the last hgate macro
    1-200
    1-201  macro     bad_dir_handler
 1  1-202           maclist   on,save
 2  1-203
 3  1-204           include   on_unit
 4  1-205
 5  1-206           use       transfer_vector
 6  1-207  .handler_entry:
 7  1-208           tra       .handler
 8  1-209  .handler_restart_entry:
 9  1-210           tra       .handler_restart_point
10  1-211
11  1-212           use       main
12  1-213  .set_dir_trap:
13  1-214           stx0      .entryp             save for restart
14  1-215
15  1-216           mlr       (),(pr),fill(000)
16  1-217           desc9a    0,0
17  1-218           desc9a    .on_unit,10*4
18  1-219           eppbp     .bad_dir_name
19  1-220           spribp    .on_unit+on_unit.name
20  1-221           eppbp     .handler_entry
21  1-222           spribp    .on_unit+on_unit.body
22  1-223           lxl1      .bad_dir_desc
23  1-224           sxl1      .on_unit+on_unit.size
24  1-225           eaa       .on_unit            set up on-unit for bad_dir_
25  1-226           sbla      sp|0,du             .. make rel to sp
26  1-227           sta       sp|stack_frame.on_unit_rel_ptrs
27  1-228           lda       stack_frame.condition_bit,dl
28  1-229           orsa      sp|stack_frame.flag_word
29  1-230           tra       0,0
30  1-231
31  1-232           string    bad_dir_
32  1-233
33  1-234
```

```
34  1-235
35  1-236  .handler: epaq      sp|0                verify that call came from ring 0
36  1-237            cana      -1,dl               check ring number in AL
37  1-238            tze       *+2
38  1-239            zero      0                   go way kid you bother me
39  1-240
40  1-241            push                          " ok, we like the call
41  1-242            epplp     .my_lp,*
42  1-243            ldx0      ap|0                get display
43  1-244            eppbp     ap|2,0*
44  1-245            lda       bp|stack_frame.prev_sp
45  1-246            cana      =o700000,dl         from another ring?
46  1-247            tze       .continue_signal    if not, back to signal_
47  1-248            eppap     ap|2,*              Get mcptr
48  1-249            eppap     ap|0,*              ..
49  1-250            spriap    bp|.mcptr           .. save in gate frame
50  1-251            spribp    .label_variable+2
51  1-252            eppbp     .handler_restart_entry
52  1-253            spribp    .label_variable
53  1-254            eppbp     .label_variable
54  1-255            spribp    .unwinder_arglist+2
55  1-256            fld       =1b24,dl
56  1-257            staq      .unwinder_arglist
57  1-258            call      unwinder_$unwinder_(.unwinder_arglist)
58  1-259
59  1-260  .continue_signal:
60  1-261            lda       =o400000,du         "1"b
61  1-262            sta       ap|10,*             set continue bit
62  1-263            return
63  1-264
64  1-265  .handler_restart_point:
65  1-266            epaq      sp|0                check that call came from ring 0
66  1-267            cana      -1,dl
67  1-268            tze       *+2
68  1-269            zero      1
69  1-270            epplp     .my_lp,*
70  1-271            lca       stack_frame.condition_bit+1,dl  Vanish on-unit
71  1-272            ansa      sp|stack_frame.flag_word
72  1-273            eppbp     .mcptr
73  1-274            spribp    .vfl_arglist+2
74  1-275            fld       =1b24,dl
75  1-276            staq      .vfl_arglist
76  1-277            short_call  verify_lock$verify_lock_bad_dir(.vfl_arglist)
77  1-278            ldx0      .entryp
78  1-279            eppap     sp|stack_frame.arg_ptr,*
79  1-280            tra       0,0                 retry the call
80  1-281
81  1-282            maclist   restore
82  1-283            eject
83  1-284            maclist   restore
84  1-285  &end
    1-286
    1-287
```

```
    1-288  " GATE_INFO - general info for non-hardcore gates
    1-289
    1-290  macro     gate_info   .
  1 1-291           maclist   on,save
  2 1-292           use       transfer_vector
  3 1-293           tra       .actor
  4 1-294
  5 1-295           use       main
  6 1-296           join      /text/transfer_vector,main
  7 1-297
  8 1-298           maclist   restore
  9 1-299           eject
 10 1-300
 11 1-301  .actor:  getlp
 12 1-302           maclist   restore
 13 1-303           gate_actor
 14 1-304
 15 1-305           maclist   restore
 16 1-306           eject
 17 1-307
 18 1-308  .setup:  push
 19 1-309           getlp
 20 1-310           maclist   restore
 21 1-311           gcheck
 22 1-312           tra       0,2
 23 1-313
 24 1-314           maclist   restore
 25 1-315           eject
 26 1-316
 27 1-317           maclist   restore
 28 1-318  &end
    1-319
    1-320
```

```
    1-321  " Macro to generate gate actor.
    1-322
    1-323  macro     gate_actor
  1 1-324           maclist   on,save
  2 1-325           eppbp     ap|2,*
  3 1-326           lda       bp|-1               get length of string
  4 1-327           tze       .return_name        zero length => get name
  5 1-328
  6 1-329           adla      1,dl                include length of acc
  7 1-330           stz       ap|4,*
  8 1-331           tsx0      .search_defs
  9 1-332
 10 1-333           cmpc      (pr,rl),(pr,rl)     compare name
 11 1-334           desc9a    bp|-1(3),al
 12 1-335           desc9a    bb|0,al
 13 1-336           tnz       .next_def
 14 1-337
 15 1-338           lda       ab|1,2              return location
 16 1-339           arl       18
 17 1-340           sta       ap|4,*
 18 1-341
 19 1-342           short_return
 20 1-343
 21 1-344  .return_name:
 22 1-345           lxl3      ap|4,*              get location
 23 1-346           tsx0      .search_defs
 24 1-347
 25 1-348           cmpx3     ab|1,2              compare location
 26 1-349           tnz       .next_def
 27 1-350
 28 1-351           lda       bb|0                get length of name
 29 1-352           arl       27
 30 1-353           sta       bp|-1               set length of varying string
 31 1-354           mlr       (pr,rl),(pr,rl)     return string
 32 1-355           desc9a    bb|0(1),al
 33 1-356           desc9a    bp|0,al
 34 1-357
 35 1-358           short_return
 36 1-359
 37 1-360  .search_defs:
 38 1-361           eax2      0
 39 1-362           eppab     lp|0,*              ab -> defs
 40 1-363  .defs_loop:
 41 1-364           lxl1      ab|1,2              get class and flags
 42 1-365           cmpx1     =o400000,du         must be class 0
 43 1-366           tnz       .next_def
 44 1-367
 45 1-368           ldx7      ab|2,2
 46 1-369           eppbb     ab|0,7              bb -> name
 47 1-370           tra       0,0                 test definition
 48 1-371  .next_def:
 49 1-372           ldx2      ab|0,2              chain to next def
 50 1-373           tnz       .defs_loop
 51 1-374
```

```
52  1-375                   short_return
53  1-376
54  1-377  &end
       1-378
       1-379
```

```
    1-380  " Miscellaneous macros.
    1-381
    1-382  macro     gcheck
 1  1-383           maclist   on,save
 2  1-384           ldx1      -2,2                get number of args expected
 3  1-385           tze       .no_gate_error      if zero, none or doesn't matter
 4  1-386           cmpx1     ap|0                compare against number given
 5  1-387           tze       .no_gate_error      args match, call procedure
 6  1-388
 7  1-389           call      signal_$signal_(signal_arglist)
 8  1-390           oct       0
 9  1-391
10  1-392           even
11  1-393  signal_arglist:
12  1-394           zero      2,4
13  1-395           zero      2,0
14  1-396           arg       .gate_errorname
15  1-397           arg
16  1-398           arg       .gate_errordesc
17  1-399           arg
18  1-400
19  1-401           string    gate_error
20  1-402
21  1-403
22  1-404  .no_gate_error:
23  1-405  &end
    1-406
    1-407  macro     string
 1  1-408  .&1name:
 2  1-409           aci       "&1"
 3  1-410  .&1desc:
 4  1-411           vfd       o9/525,o27/&11
 5  1-412
 6  1-413  &end
    1-414
    1-415  macro     eject
 1  1-416           maclist   on,save
 2  1-417
```

```
                        3  1-418  &end
                           1-419
                           1-420  " END INCLUDE FILE ..... gate_macros.incl.alm
                           1-421
                              6            gate_info
                                           use       transfer_vector
000000  0a   000004 7100 00                 tra       .actor

                                           use       main
                                           join      /text/transfer_vector,main
```

```
000004  aa  7 00046 2721 20        .actor:   getlp
000005  aa  0 00002 3521 20                  eppbp     ap|2,*
000006  aa  2 77777 2351 00                  lda       bp|-1               get length of string
000007  0a   000023 6000 00                  tze       .return_name        zero length => get name

000010  aa   000001 0350 07                  adla      1,dl                include length of acc
000011  aa  0 00004 4501 20                  stz       ap|4,*
000012  0a   000036 7000 00                  tsx0      .search_defs

000013  aa  0 00140 1065 40                  cmpc      (pr,rl),(pr,rl)     compare name
000014  aa   277777 600005                   desc9a    bp|-1(3),al
000015  aa   300000 000005                   desc9a    bb|0,al
000016  0a   000046 6010 00                  tnz       .next_def

000017  aa  1 00001 2351 12                  lda       ab|1,2              return location
000020  aa   000022 7710 00                  arl       18
000021  aa  0 00004 7551 20                  sta       ap|4,*

000022  aa  7 00044 7101 20                  short_return

000023                             .return_name:
000023  aa  0 00004 7231 20                  lxl3      ap|4,*              get location
000024  0a   000036 7000 00                  tsx0      .search_defs

000025  aa  1 00001 1031 12                  cmpx3     ab|1,2              compare location
000026  0a   000046 6010 00                  tnz       .next_def

000027  aa  3 00000 2351 00                  lda       bb|0                get length of name
000030  aa   000033 7710 00                  arl       27
000031  aa  2 77777 7551 00                  sta       bp|-1               set length of varying string
000032  aa  0 00140 1005 40                  mlr       (pr,rl),(pr,rl)     return string
000033  aa   300000 200005                   desc9a    bb|0(1),al
000034  aa   200000 000005                   desc9a    bp|0,al

000035  aa  7 00044 7101 20                  short_return

000036                             .search_defs:
000036  aa   000000 6220 00                  eax2      0
000037  aa  4 00000 3515 20                  eppab     lp|0,*              ab -> defs
000040                             .defs_loop:
000040  aa  1 00001 7211 12                  lxl1      ab|1,2              get class and flags
000041  aa   400000 1010 03                  cmpx1     =o400000,du         must be class 0
000042  0a   000046 6010 00                  tnz       .next_def

000043  aa  1 00002 2271 12                  ldx7      ab|2,2
000044  aa  1 00000 3535 17                  eppbb     ab|0,7              bb -> name
000045  aa   000000 7100 10                  tra       0,0                 test definition
000046                             .next_def:
000046  aa  1 00000 2221 12                  ldx2      ab|0,2              chain to next def
000047  0a   000040 6010 00                  tnz       .defs_loop

000050  aa  7 00044 7101 20                  short_return
```

```
000051  aa   000060 6270 00        .setup:  push
000052  aa  7 00040 2721 20
000053  aa  7 00046 2721 20                 getlp
000054  aa   777776 2210 12                 ldx1     -2,2              get number of args expected
000055  0a   000102 6000 00                 tze      .no_gate_error    if zero, none or doesn't matter
000056  aa  0 00000 1011 00                 cmpx1    ap|0              compare against number given
000057  0a   000102 6000 00                 tze      .no_gate_error    args match, call procedure

000060  aa  6 00000 2541 00                 call     signal_$signal_(signal_arglist)
000061  0a   000070 3500 00
000062  4a  4 00010 3521 20
000063  aa  6 00040 7531 00
000064  aa  7 00036 6701 20
000065  aa  6 00000 1731 00
000066  aa  6 00040 0731 00
000067  aa   000000 000000                 oct      0

                                         even
000070                           signal_arglist:
000070  aa   000002 000004                 zero     2,4
000071  aa   000002 000000                 zero     2,0
000072  0a   000076 0000 00                 arg      .gate_errorname
000073  aa   000000 0000 00                 arg
000074  0a   000101 0000 00                 arg      .gate_errordesc
000075  aa   000000 0000 00                 arg

                                         string   gate_error
000076                           .gate_errorname:
000076  aa  147 141 164 145                aci      "gate_error"
000077  aa  137 145 162 162
000100  aa  157 162 000 000
000101                           .gate_errordesc:
000101  aa   525000 000010                 vfd      o9/525,o27/10


000102                           .no_gate_error:
000102  aa   000000 7100 12                 tra      0,2
```

```
                                    7          gate      find,put_find_gate_util_,find,4
                        000001                 segdef    find
000001                                  find:
000001  0a   000104 7100 00                    tra       find.e
000103  aa   000010 000000                     zero      4*2,0
000104                                  find.e:
000104  0a   000051 7020 00                    tsx2      .setup
000105  4a  4 00012 3521 20                    short_call put_find_gate_util_$find(ap|0)
000106  aa  7 00036 6701 20
000107  aa  6 00030 3701 20
000110  aa  7 00042 7101 20                    return

                                    8          gate      put,put_find_gate_util_,put,4
                        000002                 segdef    put
000002                                  put:
000002  0a   000112 7100 00                    tra       put.e
000111  aa   000010 000000                     zero      4*2,0
000112                                  put.e:
000112  0a   000051 7020 00                    tsx2      .setup
000113  4a  4 00014 3521 20                    short_call put_find_gate_util_$put(ap|0)
000114  aa  7 00036 6701 20
000115  aa  6 00030 3701 20
000116  aa  7 00042 7101 20                    return

                                    9          end
```

NO LITERALS

```
NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

    000120  5a   000003 000000
    000121  5a   000032 600000
    000122  aa   000000 000000
    000123  55   000012 000002
    000124  5a   000002 400003
    000125  55   000006 000012
    000126  aa  016 160 165 164
    000127  aa  137 146 151 156
    000130  aa  144 137 147 141
    000131  aa  164 145 137 000
    000132  55   000016 000003
    000133  0a   000002 400000
    000134  55   000015 000003
    000135  aa  003 160 165 164        put
    000136  55   000023 000012
    000137  0a   000001 400000
    000140  55   000021 000003
    000141  aa  004 146 151 156        find
    000142  aa  144 000 000 000
    000143  55   000002 000016
    000144  6a   000000 400002
    000145  55   000026 000003
    000146  aa  014 163 171 155        symbol_table
    000147  aa  142 157 154 137
    000150  aa  164 141 142 154
    000151  aa  145 000 000 000

DEFINITIONS HASH TABLE

    000152  aa   000000 000015
    000153  aa   000000 000000
    000154  aa   000000 000000
    000155  5a   000012 000000
    000156  aa   000000 000000
    000157  aa   000000 000000
    000160  aa   000000 000000
    000161  5a   000016 000000
    000162  5a   000023 000000
    000163  aa   000000 000000
    000164  aa   000000 000000
    000165  aa   000000 000000
    000166  aa   000000 000000
    000167  aa   000000 000000

EXTERNAL NAMES

    000170  aa  023 160 165 164        put_find_gate_util_
    000171  aa  137 146 151 156
    000172  aa  144 137 147 141
    000173  aa  164 145 137 165
    000174  aa  164 151 154 137
    000175  aa  007 163 151 147        signal_
    000176  aa  156 141 154 137
```

```
NO TRAP POINTER WORDS

TYPE PAIR BLOCKS

    000177  aa   000004 000000
    000200  55   000050 000015
    000201  aa   000004 000000
    000202  55   000050 000021
    000203  aa   000004 000000
    000204  55   000055 000055
    000205  aa   000001 000000
    000206  aa   000000 000000

INTERNAL EXPRESSION WORDS

    000207  5a   000057 000000
    000210  5a   000061 000000
    000211  5a   000063 000000
```

LINKAGE INFORMATION

```
000000  aa   000000 000000
000001  0a   000120 000000
000002  aa   000000 000000
000003  aa   000000 000000
000004  aa   000000 000000
000005  aa   000000 000000
000006  22   000010 000016
000007  a2   000000 000000
000010  9a   777770 0000 46        signal_|signal_
000011  5a   000071 0000 00
000012  9a   777766 0000 46        put_find_gate_util_|find
000013  5a   000070 0000 00
000014  9a   777764 0000 46        put_find_gate_util_|put
000015  5a   000067 0000 00
```

SYMBOL INFORMATION

SYMBOL TABLE HEADER

```
000000  aa  000000 000001
000001  aa  163171 155142
000002  aa  164162 145145
000003  aa  000000 000004
000004  aa  000000 106730
000005  aa  414344 011300
000006  aa  000000 107027
000007  aa  564756 217737
000010  aa  141154 155040
000011  aa  040040 040040
000012  aa  000024 000040
000013  aa  000034 000040
000014  aa  000044 000100
000015  aa  000002 000002
000016  aa  000064 000000
000017  aa  000000 000143
000020  aa  000000 000117
000021  aa  000000 000130
000022  aa  000134 000117
000023  aa  000064 000000
000024  aa  101114 115040
000025  aa  126145 162163
000026  aa  151157 156040
000027  aa  066056 062054
000030  aa  040112 165156
000031  aa  145040 061071
000032  aa  067071 040040
000033  aa  040040 040040
000034  aa  106162 157155
000035  aa  155145 162056
000036  aa  115105 104141
000037  aa  144155 151156
000040  aa  056141 040040
000041  aa  040040 040040
000042  aa  040040 040040
000043  aa  040040 040040
000044  aa  154151 163164
000045  aa  040040 040040
000046  aa  040040 040040
000047  aa  040040 040040
000050  aa  040040 040040
000051  aa  040040 040040
000052  aa  040040 040040
000053  aa  040040 040040
000054  aa  040040 040040
000055  aa  040040 040040
000056  aa  040040 040040
000057  aa  040040 040040
000060  aa  040040 040040
000061  aa  040040 040040
000062  aa  040040 040040
```

```
000063  aa   040040 040040
000064  aa   000000 000001
000065  aa   000000 000002
000066  aa   000076 000037
000067  aa   063453 431404
000070  aa   000000 107027
000071  aa   564153 400000
000072  aa   000106 000041
000073  aa   057660 372535
000074  aa   000000 105766
000075  aa   053324 600000
000076  aa   076165 144144          >udd>F15D>s1>put_find_gate_.alm
000077  aa   076106 061065
000100  aa   104076 163061
000101  aa   076160 165164
000102  aa   137146 151156
000103  aa   144137 147141
000104  aa   164145 137056
000105  aa   141154 155040
000106  aa   076154 144144          >ldd>include>gate_macros.incl.alm
000107  aa   076151 156143
000110  aa   154165 144145
000111  aa   076147 141164
000112  aa   145137 155141
000113  aa   143162 157163
000114  aa   056151 156143
000115  aa   154056 141154
000116  aa   155040 040040
```

MULTICS ASSEMBLY CROSS REFERENCE LISTING

Value	Symbol	Source file	Line number
4	.actor	put_find_gate_:	6.
40	.defs_loop	put_find_gate_:	6.
101	.gate_errordesc	put_find_gate_:	6.
76	.gate_errorname	put_find_gate_:	6.
46	.next_def	put_find_gate_:	6.
102	.no_gate_error	put_find_gate_:	6.
23	.return_name	put_find_gate_:	6.
36	.search_defs	put_find_gate_:	6.
51	.setup	put_find_gate_:	6, 7, 8.
1	find	put_find_gate_:	7.
104	find.e	put_find_gate_:	7.
4	main	put_find_gate_:	6, 7, 8.
2	put	put_find_gate_:	8.
112	put.e	put_find_gate_:	8.
	put_find_gate_util_	put_find_gate_:	7, 8.
	signal_	put_find_gate_:	6.
70	signal_arglist	put_find_gate_:	6.
0	transfer_vector	put_find_gate_:	6, 7, 8.

NO FATAL ERRORS

```
COMPILATION LISTING OF SEGMENT trans
Compiled by: Multics PL/I Compiler, Release 25c, of February 18, 1980
Compiled at: Honeywell LISD Phoenix, System M
Compiled on: 03/19/80  1043.7 mst Wed
    Options: map

 1 trans: proc;
 2
 3 /*
 4*
 5*
 6*      Name:  trans
 7*
 8*           The trans command adds, counts, deletes, reads, and summarizes
 9*      transactions  resident  in  a  message  segment  whose  pathname  is
10*      >udd>F15d>trans.ms.  The amount of information it may return  and  the
11*      ability  to  add  "transactions"  into  the  segment  are  governed by
12*      extended access.
13*
14*
15*      Usage
16*
17*           trans key {args}
18*
19*      where:
20*
21*      1.   key
22*                is one of the functions listed below.
23*
24*                add, a
25*                          adds a transaction to the message segment.   Three
26*                          additional arguments must be furnished:
27*                               1.   part_name
28*                                    16 characters or less
29*                               2.   unit_price
30*                                    not to exceed 9999.99
31*                               3.   how_many_sold
32*                                    not to exceed 99999
33*
34*                          The  caller  must have 'append' extended access to
35*                          the segment trans.ms.
36*
37*                count, c
38*                          returns the number of transactions  (messages)  in
39*                          the segment.  The caller must have status extended
40*                          access to the segment trans.ms.
41*
42*                delete, d
43*                          deletes  the "current" transaction.  A transaction
```

```
44*                     is made current by some previous  read  operation.
45*                     If  the last transaction read was not added by the
46*                     caller,  the  caller  must  have  delete  extended
47*                     access on trans.ms.  Otherwise, the caller must at
48*                     least have 'own' extended access.
49*
50*             read, r
51*                     reads  one or more transactions.  If an additional
52*                     argument is furnished,  it  must  be  one  of  the
53*                     following:
54*
55*                     1.   all
56*                          If  user  has  'read'  extended  access,
57*                          every transaction is dumped.
58*
59*                          If user has 'own', but not 'read',  only
60*                          'own' transactions are dumped.
61*
62*                     2.   first
63*                          If  user has 'read' extended access, the
64*                          first transaction is dumped.
65*
66*                          If user has 'own', but not  'read',  the
67*                          first 'own' transaction is dumped.
68*
69*                     3.   last
70*                          analogous to 'trans read first'
71*
72*                     4.   next
73*                          If  user has 'read' extended access, the
74*                          next transaction in trans.ms is  dumped.
75*
76*                          If  user  has 'own', but not 'read', the
77*                          caller's next transaction is dumped.
78*
79*                     5.   prior
80*                          analogous to 'trans read next'
81*
82*                     6.   <argument missing>
83*                          reads the current message.
84*
85*             summary, s
86*                     If the caller  has  'read'  extended  access,  the
87*                     dollar  amount  grand total of all transactions in
88*                     the segment is returned.  If the caller has 'own',
89*                     but not 'read', the dollars amount grand total  of
90*                     'own' transactions is returned instead.
91*
92*     2.   args
93*                     is dependent upon key as documented above.
94**/
95 /*
```

```
*/
     96 dcl  ME internal static options (constant) char (5) init ("trans");
     97 dcl  TARGET_SEGMENT int static options (constant) char (8) init ("trans.ms");
     98 dcl  TARGET_DIR int static options (constant) char (9) init (">udd>F15d");
     99 dcl (error_table_$moderr, error_table_$no_message, error_table_$bigarg,
    100      error_table_$bad_arg, error_table_$too_many_args, error_table_$wrong_no_of_args)
    101      ext static fixed bin (35);
    102 dcl  areap ptr;
    103 dcl (argp, messagep, arg_list_ptr) ptr;
  1   1 /* BEGIN INCLUDE FILE . . . mseg_return_args_v3.incl.pl1 */
  1   2
  1   3
  1   4 /* structure returned when message is read from a message segment */
  1   5
  1   6
  1   7 dcl  ms_arg_ptr ptr;
  1   8
  1   9 dcl 1 mseg_return_args based (ms_arg_ptr) aligned,
  1  10     2 ms_ptr ptr,                                        /* pointer to message */
  1  11     2 ms_len fixed bin (18),                             /* length of message  in bits */
  1  12     2 sender_id char (32),                               /* process-group ID of sender */
  1  13     2 level fixed bin,                                   /* validation level of sender */
  1  14     2 ms_id bit (72),                                    /* unique ID of message */
  1  15     2 sender_authorization bit (72),                     /* access authorization of message sender */
  1  16     2 access_class bit (72);                             /* message access class */
  1  17
  1  18
  1  19 /* END INCLUDE FILE . . . mseg_return_args_v3.incl.pl1 */
    104
    105 dcl (code, binary_number_sold) bin (35);
    106 dcl  index fixed bin;
    107
    108 dcl  message_segment_$open entry (char (*), char (*), fixed bin, fixed bin (35));
    109 dcl  message_segment_$close entry (fixed bin, fixed bin (35));
    110 dcl  message_segment_$get_message_count_index entry (fixed bin, fixed bin, fixed bin (35));
    111 dcl  message_segment_$add_index entry (fixed bin, ptr, fixed bin (24), bit (72) aligned, bin (35));
    112 dcl  message_segment_$delete_index entry (fixed bin, bit (72) aligned, fixed bin (35));
    113 dcl  message_segment_$read_index entry (fixed bin, ptr, bit (1) aligned, ptr, fixed bin (35));
    114 dcl  message_segment_$incremental_read_index entry (fixed bin, ptr, bit (2) aligned, bit (72) aligned, ptr, fixed bin (35)
\c);
    115 dcl  message_segment_$own_read_index entry (fixed bin, ptr, bit (1) aligned, ptr, fixed bin (35));
    116 dcl  message_segment_$own_incremental_read_index entry (fixed bin, ptr, bit (2) aligned, bit (72) aligned, ptr, fixed bin
\c(35));
    117
    118 dcl 1 trans_msg based (ms_arg_ptr -> mseg_return_args.ms_ptr), /* Sometimes pointed to by messagep */
    119     2 widget_name char (16),
    120     2 unit_price pic "$$$$9.v99",
    121     2 how_many_sold fixed dec (5, 0),
    122     2 total_cost pic "(7)$9.v99";
    123 dcl  grand_total_dollars internal static pic "(7)$9.v99";
    124 dcl  silent bit (1) internal static init ("0"b);
    125 dcl  read_option (2:6) char (5) internal static options (constant)
    126      init ("all", "next", "prior", "first", "last");
    127 dcl  current_message_id bit (72) internal static aligned init ("0"b);
    128 dcl  looping_index fixed bin (17) init (1);
```

```
129 dcl  own bit (1) init ("0"b);
130 dcl  read_options internal static options (constant) char (121)
131      init ("Your read options are:
132           trans read
133           trans read all
134           trans read next
135           trans read prior
136           trans read first
137           trans read last");
138 dcl (com_err_, ioa_) entry options (variable);
139 dcl  cu_$arg_count entry (fixed bin);
140 dcl  cu_$arg_ptr entry (fixed bin, ptr, fixed bin(21), fixed bin (35));
141 dcl  cu_$arg_list_ptr entry (ptr);
142 dcl  cu_$arg_ptr_rel entry (fixed bin, ptr, fixed bin(21), fixed bin (35), ptr);
143 dcl  nargs fixed bin, argl fixed bin(21);
144 dcl  arg char (argl) based (argp);
145 dcl  auto_area area (2048);              /* Should be large enough for any and all allocated structures */
146 dcl  string char (24);
147 dcl  date_time_ entry (fixed bin (71), char (*));
148 dcl  clock_ entry returns (fixed bin (71));
149 dcl  cv_dec_check_ entry (char (*), fixed bin (35)) returns (fixed bin (35));
150 dcl  conversion condition;
151 dcl  binary_time fixed bin (71);
152 dcl  alternate_binary_time bit (72) aligned based;
153 dcl (size, addr) builtin;
154 dcl  message_count;
155 dcl  iox_$control entry (ptr, char(*), ptr, fixed bin(35));
156 dcl  iox_$user_io ext static ptr;
157                                                    /*
```

```
*/
158                                                          /*  COMMON BEGINNING POINT */
159
160          call cu_$arg_count (nargs);
161          if nargs = 0 then do;
162               call com_err_ (error_table_$wrong_no_of_args, "trans",
163               "^/You have not invoked 'trans' properly.
164 Please type 'trans key {args}', where 'key' is either
165 'add', 'delete', 'read', 'summary', or 'count'.
166 'args' is a function of what key has been supplied.");
167               return;
168          end;
169          call cu_$arg_ptr (1, argp, argl, code);
170          call cu_$arg_list_ptr (arg_list_ptr);   /* Sets an automatic pointer */
171          if arg = "add" | arg = "a" then call trans_add;
172          else if arg = "delete" | arg = "d" then call trans_delete;
173          else if arg = "read" | arg = "r" then call trans_read;
174          else if arg = "summary" | arg = "s" then call trans_summary;
175          else if arg = "count" | arg = "c" then call trans_count;
176          else call com_err_ (error_table_$bad_arg, ME, arg);
177          return;
178                                                          /*
```

```
*/
trans_read: proc;

          if nargs > 2 then do;
               call com_err_ (error_table_$too_many_args, ME, read_options);
               return;
          end;

          /* Goal of the following do group is to set looping_index to
              1,2,3,4,5, or 6 for a subsequent 'goto'.          */
          if nargs = 2 then do;
               call cu_$arg_ptr_rel (2, argp, argl, code, arg_list_ptr);
               do looping_index = 2 to 6 while (arg ^= read_option (looping_index));
               end;

               if looping_index = 7 then do;
                    call com_err_ (error_table_$bad_arg, ME, read_options);
                    return;
               end;
          end;

          call message_segment_$open (TARGET_DIR, TARGET_SEGMENT, index, code);
          if code ^= 0 then do;
               call com_err_ (code, ME, "While attempting to open ^a>^a.", TARGET_DIR, TARGET_SEGMENT);
               return;
          end;

          areap = addr (auto_area);
          allocate mseg_return_args in (auto_area) set (ms_arg_ptr);
               /* No need to worry about freeing above, because area itself is automatic */
          goto read_label (looping_index);

read_label (1):                                             /*   'trans read'   */
          if current_message_id = "0"b then do;
               call com_err_ (0, ME, "There is no current message.");
               return;
          end;
          else do;

               call message_segment_$incremental_read_index (index, areap, "00"b,
                    current_message_id, ms_arg_ptr, code);       /* User must have 'r' extended access */
               if code = error_table_$moderr then call message_segment_$own_incremental_read_index (index,
                    areap, "00"b, current_message_id, ms_arg_ptr, code);
               if code ^= 0 then do;
                    call com_err_ (code, ME);
                    call message_segment_$close (index, code);
                    return;
               end;

               call print_message ();

               call message_segment_$close (index, code);
               return;
          end;
```

```
233 read_label (2):
234                                                          /*   'trans read all'   */
235               call message_segment_$read_index (index, areap, "0"b /* from the first */,
236                    ms_arg_ptr, code);
237               if code = error_table_$moderr then do;
238                    call message_segment_$own_read_index
239                         (index, areap, "0"b, ms_arg_ptr, code);
240                    if code = error_table_$moderr then do;
241                         call com_err_ (code, ME);
242                         call message_segment_$close (index, code);
243                         return;
244                    end;
245                    own = "1"b;
246               end;
247               if code = error_table_$no_message then do;
248                    call com_err_ (code, ME, "^[You have no^;No] messages in ^a>^a.", own, TARGET_DIR, TARGET_SEGMENT);
249                    call message_segment_$close (index, code);
250                    return;
251               end;
252
253               if own then do;
254
255                    do while (code ^= error_table_$no_message);
256
257                         call print_message ();
258                         current_message_id = ms_arg_ptr -> mseg_return_args.ms_id;
259                         call message_segment_$own_incremental_read_index (index, areap, "01"b /* next */,
260                              current_message_id, ms_arg_ptr, code);
261                    end;
262
263               end;
264               else do;
265
266                    do while (code ^= error_table_$no_message);
267
268                         call print_message ();
269                         current_message_id = ms_arg_ptr -> mseg_return_args.ms_id;
270                         call message_segment_$incremental_read_index (index, areap, "01"b /* next */,
271                              current_message_id, ms_arg_ptr, code);
272                    end;
273               end;
274
275
276               call message_segment_$close (index, code);
277               return;
278 read_label (3):
279                                                          /*   'trans read next'   */
280
281               if current_message_id = "0"b then do;
282                    call com_err_ (0, ME, "There is no current message.");
283                    return;
284               end;
285               else do;
286
287                    call message_segment_$incremental_read_index (index, areap, "01"b,
```

```
288                      current_message_id, ms_arg_ptr, code);
289                 if code = error_table_$moderr then call message_segment_$own_incremental_read_index (index,
290                      areap, "01"b, current_message_id, ms_arg_ptr, code);
291                 if code = error_table_$moderr then do;
292                      call com_err_ (code, ME);
293                      call message_segment_$close (index, code);
294                      return;
295                 end;
296                 if code = error_table_$no_message then do;
297                      call com_err_ (code, ME, "There is no next message.");
298                      call message_segment_$close (index, code);
299                      return;
300                 end;
301
302                 call print_message ();
303
304                 current_message_id = ms_arg_ptr -> ms_id; /* Update what is now the current msg */
305                 call message_segment_$close (index, code);
306                 return;
307            end;
308 read_label (4):                                                    /*   'trans read prior'   */
309
310
311            if current_message_id = "0"b then do;
312                 call com_err_ (0, ME, "There is no current message.");
313                 return;
314            end;
315            else do;
316
317                 call message_segment_$incremental_read_index (index, areap, "10"b,
318                      current_message_id, ms_arg_ptr, code);
319                 if code = error_table_$moderr then call message_segment_$own_incremental_read_index (index,
320                      areap, "10"b, current_message_id, ms_arg_ptr, code);
321                 if code = error_table_$moderr then do;
322                      call com_err_ (code, ME);
323                      call message_segment_$close (index, code);
324                      return;
325                 end;
326                 if code = error_table_$no_message then do;
327                      call com_err_ (code, ME, "There is no prior message.");
328                      call message_segment_$close (index, code);
329                      return;
330                 end;
331
332                 call print_message ();
333
334                 current_message_id = ms_arg_ptr -> mseg_return_args.ms_id; /* Update what is now the current msg */
335                 call message_segment_$close (index, code);
336                 return;
337            end;
338 read_label (5):                                                    /*   'trans read first'   */
339
340
341            call message_segment_$read_index (index, areap, "0"b /* the first */,
342                 ms_arg_ptr, code);
```

```
343               if code = error_table_$moderr then do;
344                    call message_segment_$own_read_index (index, areap, "0"b /* the first */,
345                         ms_arg_ptr, code);
346                    if code = error_table_$moderr then do;
347                         call com_err_ (code, ME);
348                         call message_segment_$close (index, code);
349                         return;
350                    end;
351                    own = "1"b;
352               end;
353               if code = error_table_$no_message then do;
354                    call com_err_ (code, ME, "^[You have no^;No] messages in ^a>^a.", own, TARGET_DIR, TARGET_SEGMENT);
355                    call message_segment_$close (index, code);
356                    return;
357               end;
358
359               call print_message ();
360               current_message_id = ms_arg_ptr -> mseg_return_args.ms_id; /* For a possible
361*                                                        'trans read' in the future */
362               call message_segment_$close (index, code);
363               return;
364 read_label (6):
365                                                       /*   'trans read last'   */
366
367               call message_segment_$read_index (index, areap, "1"b /* the last */,
368                    ms_arg_ptr, code);
369               if code = error_table_$moderr then do;
370                    call message_segment_$own_read_index (index, areap, "1"b /* the last */,
371                         ms_arg_ptr, code);
372                    own = "1"b;
373               end;
374               if code = error_table_$no_message then do;
375                    call com_err_ (code, ME, "^[You have no^;No] messages in ^a>^a.", own, TARGET_DIR, TARGET_SEGMENT);
376                    call message_segment_$close (index, code);
377                    return;
378               end;
379
380               call print_message ();
381               current_message_id = ms_arg_ptr -> mseg_return_args.ms_id; /* For a possible
382*                                                        'trans read' in the future */
383               call message_segment_$close (index, code);
384               return;
385          end trans_read;
386                                                       /*
```

```
*/
     387 trans_add: proc;
     388                if nargs ^= 4 then do;
     389                     call com_err_ (error_table_$wrong_no_of_args, ME, "Proper invocation:
     390           trans add widget_name unit_price how_many_sold");
     391                     return;
     392                end;
     393                                                  /*  Assign the elements in the structure trans_msg one by one
\c*/
     394                allocate trans_msg in (auto_area) set (messagep);
     395                call cu_$arg_ptr_rel (2, argp, argl, code, arg_list_ptr);
     396                if argl > 16 then do;
     397                     call com_err_ (error_table_$bigarg, ME, "Widget_name mustn't exceed 16 characters.");
     398                     return;
     399                end;
     400                else messagep -> trans_msg.widget_name = arg;
     401
     402                call cu_$arg_ptr_rel (3, argp, argl, code, arg_list_ptr);
     403                on conversion begin;
     404                     call com_err_ (error_table_$bad_arg, ME, "Unit price specified does not look something like 9999.99");
     405                     goto add_return_point;
     406                end;
     407                messagep -> trans_msg.unit_price = arg;
     408                revert conversion;                /* If we get here, conversion went fine */
     409                call cu_$arg_ptr_rel (4, argp, argl, code, arg_list_ptr);
     410                binary_number_sold = cv_dec_check_ (arg, code);
     411                if code ^= 0 then do;
     412                     call com_err_ (error_table_$bad_arg, ME, "Number sold could not be converted into an integer.");
     413                     return;
     414                end;
     415                else if binary_number_sold > 99999 then do;
     416                     call com_err_ (error_table_$bad_arg, ME, "Number sold exceeds 99999.");
     417                     return;
     418                end;
     419                else messagep -> trans_msg.how_many_sold = binary_number_sold;
     420
     421                messagep -> trans_msg.total_cost = messagep -> trans_msg.unit_price *
     422                     messagep -> trans_msg.how_many_sold;
     423                                                  /*  trans_msg structure ready for addition to >udd>F15d>trans.
\cms  */
     424
     425                call message_segment_$open (TARGET_DIR, TARGET_SEGMENT, index, code);
     426                if code ^= 0 then do;
     427                     call com_err_ (code, ME, "While attempting to open ^a>^a.", TARGET_DIR, TARGET_SEGMENT);
     428                     return;
     429                end;
     430                binary_time = clock_ (); /* Need a unique id for message_id, hence clock_ */
     431                call message_segment_$add_index (index, messagep, 36 * size (messagep -> trans_msg),
     432                     addr (binary_time) -> alternate_binary_time, code);
     433                if code ^= 0 then call com_err_ (code, ME, "While attempting to add a transaction.");
     434                else do;
     435                     current_message_id = addr (binary_time) -> alternate_binary_time;
     436                     call date_time_ ((current_message_id), string);
     437                     call ioa_ ("Transaction added at ^a", string);
     438                end;
```

```
439                call message_segment_$close (index, code);
440 add_return_point:
441           end trans_add;
442                                                          /*
```

```
*/
443 trans_delete: proc;
444
445               if nargs ^= 1 then do;
446                    call com_err_ (error_table_$wrong_no_of_args, ME, "Proper invocation:
447          trans delete ");
448                    return;
449               end;
450               if current_message_id = "0"b then do;
451                    call com_err_ (0, ME, "There is no current transaction to delete.
452          Try some kind of read first.");
453                    return;
454               end;
455               else do;
456                    call message_segment_$open (TARGET_DIR, TARGET_SEGMENT, index, code);
457                    if code ^= 0 then do;
458                         call com_err_ (code, ME, "While attempting to open ^a>^a.", TARGET_DIR, TARGET_SEGMENT);
459                         return;
460                    end;
461                    call message_segment_$delete_index (index, current_message_id, code);
462                    if code = 0 then do;
463                         current_message_id = "0"b;
464                         call ioa_ ("Transaction deleted.");
465                    end;
466                    else call com_err_ (code, ME, "While attempting to delete the current message.");
467                    call message_segment_$close (index, code);
468               end;
469          end trans_delete;
470                                                  /*
```

```
*/
471 trans_summary: proc;
472
473 dcl  trans entry options (variable);
474
475                 if nargs ^= 1 then do;
476                     call com_err_ (error_table_$wrong_no_of_args, ME, "Proper invocation:
477           trans summary");
478                     return;
479                 end;
480                     /* We can not tolerate a quit, so ... */
481                 call iox_$control (iox_$user_io, "quit_disable", null (), code);
482                 silent = "1"b;
483                 grand_total_dollars = 0;
484
485                 call trans ("read", "all");
486
487                 call ioa_ ("Grand total = ^a", grand_total_dollars);
488
489                 silent = "0"b;
490                 grand_total_dollars = 0;
491                     /* Now safe to unmask quits */
492                 call iox_$control (iox_$user_io, "quit_enable", null (), code);
493
494           end trans_summary;
495                                                          /*
```

```
*/
     496 trans_count: proc;
     497
     498 /* User must have status extended access on trans.ms */
     499
     500               if nargs ^= 1 then do;
     501                    call com_err_ (error_table_$wrong_no_of_args, ME, "Proper invocation:
     502          trans count");
     503                    return;
     504               end;
     505
     506               call message_segment_$open (TARGET_DIR, TARGET_SEGMENT, index, code);
     507               if code ^= 0 then do;
     508                    call com_err_ (code, ME, "While attempting to open ^a>^a.", TARGET_DIR, TARGET_SEGMENT);
     509                    return;
     510               end;
     511
     512               call message_segment_$get_message_count_index (index, message_count, code);
     513               if code = 0 then call ioa_ ("There are ^d messages in ^a", message_count, TARGET_SEGMENT);
     514               else call com_err_ (code, ME);
     515
     516               call message_segment_$close (index, code);
     517
     518          end trans_count;
     519                                                            /*
```

```
*/
print_message: proc ();

                    /* Can tally dollars OR print transaction info */

               call date_time_ ((ms_arg_ptr -> ms_id), string);
               if ^silent then call ioa_ ("At ^a, ^a sold ^d ^a,^/ at a total cost of ^a, (unit price = ^a)",
                    string, ms_arg_ptr -> mseg_return_args.sender_id, how_many_sold, widget_name, total_cost, unit_price);
               else grand_total_dollars = grand_total_dollars + total_cost;

          end print_message;

bottom_of_trans:

     end trans;
```

SOURCE FILES USED IN THIS COMPILATION.

LINE	NUMBER	DATE MODIFIED	NAME	PATHNAME
	0	03/11/80 1122.9	trans.pl1	>user_dir_dir>F15D>Student_01>trans.pl1
104	1	08/03/77 1651.9	mseg_return_args_v3.incl.pl1	>ldd>Include>mseg_return_args_v3.incl.pl1

NAMES DECLARED IN THIS COMPILATION.

IDENTIFIER	OFFSET	LOC	STORAGE CLASS	DATA TYPE	ATTRIBUTES AND REFERENCES (* indicates a set context)
NAMES DECLARED BY DECLARE STATEMENT.					
ME		000062	constant	char(5)	initial unaligned dcl 96 set ref 176* 182* 194* 201* 213* 223* 241* 248* 282* 292* 297* 312* 322* 327* 347* 354* 375* 389* 397* 404* 412* 416* 427* 433* 446* 451* 458* 466* 476* 501* 508* 514*
TARGET_DIR		000054	constant	char(9)	initial unaligned dcl 98 set ref 199* 201* 248* 354* 375* 425* 427* 456* 458* 506* 508*
TARGET_SEGMENT		000060	constant	char(8)	initial unaligned dcl 97 set ref 199* 201* 248* 354* 375* 425* 427* 456* 458* 506* 508* 513*
addr				builtin function	dcl 153 ref 205 431 435
alternate_binary_time			based	bit(72)	dcl 152 set ref 431* 435
areap		000100	automatic	pointer	dcl 102 set ref 205* 218* 220* 233* 238* 259* 270* 287* 289* 317* 319* 338* 344* 364* 370*
arg			based	char	unaligned dcl 144 set ref 171 171 172 172 173 173 174 174 175 175 176* 190 400 407 410*
arg_list_ptr		000106	automatic	pointer	dcl 103 set ref 170* 189* 395* 402* 409*
argl		000120	automatic	fixed bin(21,0)	dcl 143 set ref 169* 171 171 172 172 173 173 174 174 175 175 176 176 189* 190 395* 396 400 402* 407 409* 410 410
argp		000102	automatic	pointer	dcl 103 set ref 169* 171 171 172 172 173 173 174 174 175 175 176 189* 190 395* 400 402* 407 409* 410
auto_area		000122	automatic	area(2048)	dcl 145 set ref 145* 205 206 394
binary_number_sold		000113	automatic	fixed bin(35,0)	dcl 105 set ref 410* 415 419
binary_time		004130	automatic	fixed bin(71,0)	dcl 151 set ref 430* 431 435
clock_		000072	constant	entry	external dcl 148 ref 430
code		000112	automatic	fixed bin(35,0)	dcl 105 set ref 169* 189* 199* 200 201* 218* 220 220* 222 223* 224* 230* 233* 237 238* 240 241* 242* 247 248* 249* 255 259* 266 270* 276* 287* 289 289* 291 292* 293* 296 297* 298* 305* 317* 319 319* 321 322* 323* 326 327* 328* 335* 338* 343 344* 346 347* 348* 353 354* 355* 362* 364* 369 370* 374 375* 376* 383* 395* 402* 409* 410* 411 425* 426 427* 431* 433 433* 439* 456* 457 458* 461* 462 466* 467* 481* 492* 506* 507 508* 512* 513 514* 516*
com_err_		000054	constant	entry	external dcl 138 ref 162 176 182 194 201 213 223 241 248 282 292 297 312 322 327 347 354 375 389 397 404 412 416 427 433 446 451 458 466 476 501 508 514
conversion		000000	stack reference	condition	dcl 150 ref 403 408
cu_$arg_count		000060	constant	entry	external dcl 139 ref 160
cu_$arg_list_ptr		000064	constant	entry	external dcl 141 ref 170
cu_$arg_ptr		000062	constant	entry	external dcl 140 ref 169
cu_$arg_ptr_rel		000066	constant	entry	external dcl 142 ref 189 395 402 409
current_message_id		000014	internal static	bit(72)	initial dcl 127 set ref 210 218* 220* 258* 259* 269* 270* 278 287* 289* 304* 308 317* 319* 334* 360* 381* 435* 436 450 461* 463*
cv_dec_check_		000074	constant	entry	external dcl 149 ref 410
date_time_		000070	constant	entry	external dcl 147 ref 436 524
error_table_$bad_arg		000024	external static	fixed bin(35,0)	dcl 99 set ref 176* 194* 404* 412* 416*

```
error_table_$bigarg              000022 external static fixed bin(35,0)          dcl 99 set ref 397*
error_table_$moderr              000016 external static fixed bin(35,0)          dcl 99 ref 220 237 240 289 291 319 321 343 346 369
error_table_$no_message          000020 external static fixed bin(35,0)          dcl 99 ref 247 255 266 296 326 353 374
error_table_$too_many_args       000026 external static fixed bin(35,0)          dcl 99 set ref 182*
error_table_$wrong_no_of_args    000030 external static fixed bin(35,0)          dcl 99 set ref 162* 389* 446* 476* 501*
grand_total_dollars              000010 internal static picture(11)              unaligned dcl 123 set ref 483* 487* 490* 527* 527
how_many_sold                 6         based           fixed dec(5,0)          level 2 dcl 118 set ref 419* 421 525*
index                            000114 automatic       fixed bin(17,0)         dcl 106 set ref 199* 218* 220* 224* 230* 233* 238*
                                                                                  242* 249* 259* 270* 276* 287* 289* 293* 298* 305*
                                                                                  317* 319* 323* 328* 335* 338* 344* 348* 355* 362*
                                                                                  364* 370* 376* 383* 425* 431* 439* 456* 461* 467*
                                                                                  506* 512* 516*
ioa_                             000056 constant        entry                   external dcl 138 ref 437 464 487 513 525
iox_$control                     000076 constant        entry                   external dcl 155 ref 481 492
iox_$user_io                     000100 external static pointer                 dcl 156 set ref 481* 492*
looping_index                    000115 automatic       fixed bin(17,0)         initial dcl 128 set ref 128* 190* 190* 193 208
message_count                    004132 automatic       fixed bin(17,0)         dcl 154 set ref 512* 513*
message_segment_$add_index       000040 constant        entry                   external dcl 111 ref 431
message_segment_$close           000034 constant        entry                   external dcl 109 ref 224 230 242 249 276 293 298 305
                                                                                  323 328 335 348 355 362 376 383 439 467 516
message_segment_$delete_index    000042 constant        entry                   external dcl 112 ref 461
message_segment_$get_message_count_index
                                 000036 constant        entry                   external dcl 110 ref 512
message_segment_$incremental_read_index
                                 000046 constant        entry                   external dcl 114 ref 218 270 287 317
message_segment_$open            000032 constant        entry                   external dcl 108 ref 199 425 456 506
message_segment_$own_incremental_read_index
                                 000052 constant        entry                   external dcl 116 ref 220 259 289 319
message_segment_$own_read_index  000050 constant        entry                   external dcl 115 ref 238 344 370
message_segment_$read_index      000044 constant        entry                   external dcl 113 ref 233 338 364
messagep                         000104 automatic       pointer                 dcl 103 set ref 394* 400 407 419 421 421 421 431*
                                                                                  431
ms_arg_ptr                       000110 automatic       pointer                 dcl 1-7 set ref 206* 218* 220* 233* 238* 258 259*
                                                                                  269 270* 287* 289* 304 317* 319* 334 338* 344* 360
                                                                                  364* 370* 381 524 525 525 525 525 525 527
ms_id                         14         based           bit(72)                 level 2 dcl 1-9 ref 258 269 304 334 360 381 524
ms_ptr                                   based           pointer                 level 2 dcl 1-9 ref 525 525 525 525 527
mseg_return_args                         based           structure               level 1 dcl 1-9 set ref 206
nargs                            000117 automatic       fixed bin(17,0)         dcl 143 set ref 160* 161 181 188 388 445 475 500
own                              000116 automatic       bit(1)                  initial unaligned dcl 129 set ref 129* 245* 248* 253
                                                                                  351* 354* 372* 375*
read_option                      000045 constant        char(5)                 initial array unaligned dcl 125 ref 190
read_options                     000006 constant        char(121)               initial unaligned dcl 130 set ref 182* 194*
sender_id                     3         based           char(32)                level 2 dcl 1-9 set ref 525*
silent                           000013 internal static bit(1)                  initial unaligned dcl 124 set ref 482* 489* 525
size                                                    builtin function        dcl 153 ref 431
string                           004122 automatic       char(24)                unaligned dcl 146 set ref 436* 437* 524* 525*
total_cost                    10         based           picture(11)             level 2 packed unaligned dcl 118 set ref 421* 525*
                                                                                  527
trans                            000102 constant        entry                   external dcl 473 ref 485
trans_msg                                based           structure               level 1 unaligned dcl 118 set ref 394 431
unit_price                    4          based           picture(8)              level 2 packed unaligned dcl 118 set ref 407* 421
                                                                                  525*
widget_name                              based           char(16)                level 2 packed unaligned dcl 118 set ref 400* 525*
```

```
NAMES DECLARED BY EXPLICIT CONTEXT.
add_return_point                003465 constant        label                   dcl 440 ref 405
bottom_of_trans                 001025 constant        label                   dcl 531
print_message                   004330 constant        entry                   internal dcl 520 ref 228 257 268 302 332 359 380
read_label                      000000 constant        label                   array(6) dcl 210 ref 208
trans                           000570 constant        entry                   external dcl 1
trans_add                       002644 constant        entry                   internal dcl 387 ref 171
trans_count                     004122 constant        entry                   internal dcl 496 ref 175
trans_delete                    003466 constant        entry                   internal dcl 443 ref 172
trans_read                      001026 constant        entry                   internal dcl 179 ref 173
trans_summary                   003727 constant        entry                   internal dcl 471 ref 174

NAMES DECLARED BY CONTEXT OR IMPLICATION.
empty                                                  builtin function        ref 145
null                                                   builtin function        ref 481 481 492 492

STORAGE REQUIREMENTS FOR THIS PROGRAM.

            Object    Text      Link      Symbol    Defs      Static
Start            0       0      5024        5130    4501        5034
Length        5462    4501       104         315     322           6

BLOCK NAME                    STACK SIZE     TYPE            WHY NONQUICK/WHO SHARES STACK FRAME
trans                               2555 external procedure  is an external procedure.
trans_read                             internal procedure    shares stack frame of external procedure trans.
trans_add                           262 internal procedure   enables or reverts conditions.
on unit on line 403                  94 on unit
trans_delete                           internal procedure    shares stack frame of external procedure trans.
trans_summary                          internal procedure    shares stack frame of external procedure trans.
trans_count                            internal procedure    shares stack frame of external procedure trans.
print_message                          internal procedure    shares stack frame of external procedure trans.

STORAGE FOR INTERNAL STATIC VARIABLES.

   LOC IDENTIFIER                BLOCK NAME
000010 grand_total_dollars       trans
000013 silent                    trans
000014 current_message_id        trans

STORAGE FOR AUTOMATIC VARIABLES.

STACK FRAME                LOC IDENTIFIER                BLOCK NAME
trans                   000100 areap                     trans
                        000102 argp                      trans
                        000104 messagep                  trans
                        000106 arg_list_ptr              trans
                        000110 ms_arg_ptr                trans
                        000112 code                      trans
                        000113 binary_number_sold        trans
                        000114 index                     trans
                        000115 looping_index             trans
                        000116 own                       trans
                        000117 nargs                     trans
                        000120 argl                      trans
                        000122 auto_area                 trans
```

```
                    004122 string                        trans
                    004130 binary_time                   trans
                    004132 message_count                 trans

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.
r_e_as              call_ext_out_desc   call_ext_out     call_int_this      return             tra_ext
enable              ext_entry           int_entry        any_to_any_tr      unpack_pic         alloc_based
empty

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.
clock_                        com_err_                   cu_$arg_count                 cu_$arg_list_ptr
cu_$arg_ptr                   cu_$arg_ptr_rel            cv_dec_check_                 date_time_
ioa_                          iox_$control               message_segment_$add_index    message_segment_$close
message_segment_$delete_index message_segment_$get_message_count_index
message_segment_$incremental_read_index                  message_segment_$open
message_segment_$own_incremental_read_index              message_segment_$own_read_index
message_segment_$read_index   trans

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.
error_table_$bad_arg          error_table_$bigarg        error_table_$moderr           error_table_$no_message
error_table_$too_many_args    error_table_$wrong_no_of_args iox_$user_io


WARNING 235 ON LINE 407
"arg" has been converted from a string value to an arithmetic value.

WARNING 235 ON LINE 436
"current_message_id" has been converted from a string value to an arithmetic value.

WARNING 235 ON LINE 524
"ms_id" has been converted from a string value to an arithmetic value.

   LINE    LOC      LINE    LOC      LINE    LOC      LINE    LOC      LINE    LOC      LINE    LOC      LINE    LOC
      1 000567       128 000575       129 000577       145 000601       160 000604       161 000613       162 000615
    167 000643       169 000644       170 000663       171 000672       172 000715       173 000732       174 000747
    175 000764       176 001001       177 001024       531 001025       179 001026       181 001027       182 001032
    183 001053       188 001054       189 001055       190 001076       191 001112       193 001114       194 001117
    195 001140       199 001141       200 001166       201 001170       202 001224       205 001225       206 001227
    208 001234       210 001236       213 001241       214 001265       218 001266       220 001310       222 001336
    223 001340       224 001355       225 001366       228 001367       230 001370       231 001401       233 001402
    237 001423       238 001427       240 001447       241 001453       242 001467       243 001500       245 001501
    247 001503       248 001505       249 001544       250 001555       253 001556       255 001561       257 001565
    258 001566       259 001573       261 001615       263 001616       266 001617       268 001623       269 001624
    270 001631       272 001653       276 001654       277 001664       278 001665       282 001670       283 001714
    287 001715       289 001737       291 001765       292 001771       293 002005       294 002016       296 002017
    297 002021       298 002044       299 002055       302 002056       304 002057       305 002064       306 002074
    308 002075       312 002100       313 002124       317 002125       319 002147       321 002175       322 002201
    323 002215       324 002226       326 002227       327 002231       328 002254       329 002265       332 002266
    334 002267       335 002274       336 002304       338 002305       343 002326       344 002332       346 002352
    347 002356       348 002372       349 002403       351 002404       353 002406       354 002410       355 002447
    356 002460       359 002461       360 002462       362 002467       363 002477       364 002500       369 002521
    370 002525       372 002545       374 002547       375 002553       376 002612       377 002623       380 002624
    381 002625       383 002632       384 002642       387 002643       388 002651       389 002655       391 002700
```

394 002701	395 002710	396 002731	397 002735	398 002761	400 002762	402 002767
403 003010	404 003024	405 003047	407 003052	408 003071	409 003072	410 003113
411 003137	412 003142	413 003166	415 003167	416 003172	417 003216	419 003217
421 003223	425 003244	426 003271	427 003274	428 003330	430 003331	431 003340
433 003362	435 003412	436 003415	437 003432	439 003453	440 003465	443 003466
445 003467	446 003472	448 003516	450 003517	451 003522	453 003546	456 003547
457 003573	458 003575	459 003631	461 003632	462 003645	463 003647	464 003652
465 003670	466 003671	467 003715	469 003726	471 003727	475 003730	476 003733
478 003757	481 003760	482 004012	483 004015	485 004021	487 004041	489 004062
490 004064	492 004070	494 004121	496 004122	500 004123	501 004126	503 004152
506 004153	507 004200	508 004202	509 004236	512 004237	513 004252	514 004301
516 004316	518 004327	520 004330	524 004331	525 004352	527 004425	529 004453

pfgu.list 1
put.list 9
put_find_gate_.alm 15
put_find_gate_.list 16
trans.list 39