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)
<table>
<thead>
<tr>
<th>DAY</th>
<th>MORNING TOPICS</th>
<th>AFTERNOON TOPICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SUBSYSTEM WRITING</td>
<td>STORAGE SYSTEM SUBROUTINES (CONTINUED)</td>
</tr>
<tr>
<td></td>
<td>STORAGE SYSTEM SUBROUTINES</td>
<td>WORKSHOP #1</td>
</tr>
<tr>
<td>2</td>
<td>WORKSHOP #1 (CONT)</td>
<td>THE COMMAND ENVIRONMENT</td>
</tr>
<tr>
<td></td>
<td>MULTICS SECURITY</td>
<td>WORKSHOP #2</td>
</tr>
<tr>
<td>3</td>
<td>ADVANCED MULTICS I/O</td>
<td>INTERPROCESS COMMUNICATION</td>
</tr>
<tr>
<td></td>
<td>WRITING I/O MODULES</td>
<td>WORKSHOP #4</td>
</tr>
<tr>
<td></td>
<td>WORKSHOP #3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>INTERPROCESS DATA BASE SHARING</td>
<td>THE STACK AND ARGUMENT LISTS</td>
</tr>
<tr>
<td></td>
<td>INTRAPROCESS TIMER MANAGEMENT</td>
<td>SPECIAL PROGRAMMING TECHNIQUES</td>
</tr>
<tr>
<td></td>
<td>WORKSHOP #5</td>
<td>WORKSHOP #6</td>
</tr>
<tr>
<td>5</td>
<td>THE PROCESS ENVIRONMENT</td>
<td>DIAL FACILITY DEMONSTRATION</td>
</tr>
<tr>
<td></td>
<td>WORKSHOP #7</td>
<td>MESSAGE SEGMENT FACILITY</td>
</tr>
<tr>
<td></td>
<td>DIAL FACILITY</td>
<td>PROGRAM LIBRARY MANAGEMENT</td>
</tr>
<tr>
<td>Topic</td>
<td>Contents</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Topic I</td>
<td>Subsystem Writing</td>
<td>1-1</td>
</tr>
<tr>
<td></td>
<td>Introduction</td>
<td>1-1</td>
</tr>
<tr>
<td></td>
<td>Terminology</td>
<td>1-2</td>
</tr>
<tr>
<td></td>
<td>Design Concerns</td>
<td>1-3</td>
</tr>
<tr>
<td></td>
<td>Capabilities for Subsystem Design in Multics</td>
<td>1-5</td>
</tr>
<tr>
<td></td>
<td>Subsystem Design Tools</td>
<td>1-7</td>
</tr>
<tr>
<td></td>
<td>Storage System Subroutines</td>
<td>1-8</td>
</tr>
<tr>
<td>Topic II</td>
<td>Storage System Subroutines</td>
<td>2-1</td>
</tr>
<tr>
<td></td>
<td>Obtaining Status Information</td>
<td>2-1</td>
</tr>
<tr>
<td></td>
<td>Multisegment Files</td>
<td>2-8</td>
</tr>
<tr>
<td></td>
<td>Temporary Segments</td>
<td>2-14</td>
</tr>
<tr>
<td>Topic III</td>
<td>Storage System Subroutines (cont)</td>
<td>3-1</td>
</tr>
<tr>
<td></td>
<td>Star and Equal Conventions</td>
<td>3-1</td>
</tr>
<tr>
<td></td>
<td>Area Manipulation</td>
<td>3-8</td>
</tr>
<tr>
<td></td>
<td>Introduction</td>
<td>3-8</td>
</tr>
<tr>
<td></td>
<td>Area Format</td>
<td>3-9</td>
</tr>
<tr>
<td></td>
<td>Area Manipulating Subroutines</td>
<td>3-12</td>
</tr>
<tr>
<td></td>
<td>Area Related Commands</td>
<td>3-16</td>
</tr>
<tr>
<td>Topic IV</td>
<td>Multics Security</td>
<td>4-1</td>
</tr>
<tr>
<td></td>
<td>Introduction</td>
<td>4-1</td>
</tr>
<tr>
<td></td>
<td>Initial ACL’s</td>
<td>4-2</td>
</tr>
<tr>
<td></td>
<td>Rings</td>
<td>4-4</td>
</tr>
<tr>
<td></td>
<td>Introduction</td>
<td>4-4</td>
</tr>
<tr>
<td></td>
<td>Ring Brackets</td>
<td>4-5</td>
</tr>
<tr>
<td></td>
<td>Ring Bracket Subroutines</td>
<td>4-8</td>
</tr>
<tr>
<td></td>
<td>Gates</td>
<td>4-10</td>
</tr>
<tr>
<td></td>
<td>Validation Level</td>
<td>4-15</td>
</tr>
<tr>
<td></td>
<td>Cross Ring I/O</td>
<td>4-17</td>
</tr>
<tr>
<td>Topic V</td>
<td>The Command Environment</td>
<td>5-1</td>
</tr>
<tr>
<td></td>
<td>Introduction</td>
<td>5-1</td>
</tr>
<tr>
<td></td>
<td>Modifying the Standard Command Environment</td>
<td>5-2</td>
</tr>
<tr>
<td></td>
<td>Current Ready Procedure</td>
<td>5-6</td>
</tr>
<tr>
<td></td>
<td>Current Command Processor</td>
<td>5-8</td>
</tr>
<tr>
<td></td>
<td>Command Level Intermediary</td>
<td>5-10</td>
</tr>
<tr>
<td></td>
<td>Some Miscellaneous cu_Entry Points</td>
<td>5-12</td>
</tr>
<tr>
<td></td>
<td>An Example</td>
<td>5-15</td>
</tr>
<tr>
<td>Topic VI</td>
<td>Advanced Multics I/O</td>
<td>6-1</td>
</tr>
<tr>
<td></td>
<td>Review</td>
<td>6-1</td>
</tr>
<tr>
<td></td>
<td>Control Orders</td>
<td>6-2</td>
</tr>
<tr>
<td></td>
<td>Useful tty_Control Orders</td>
<td>6-3</td>
</tr>
</tbody>
</table>
## CONTENTS (con't)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Writing I/O Modules</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI</td>
<td>Usefile Control Orders</td>
<td>6-7</td>
</tr>
<tr>
<td></td>
<td>Review of IOC's</td>
<td>6-9</td>
</tr>
<tr>
<td></td>
<td>Synonyming</td>
<td>6-14</td>
</tr>
<tr>
<td></td>
<td>iox Entry Points Used in I/O Modules</td>
<td>6-19</td>
</tr>
<tr>
<td>VII</td>
<td>Writing I/O Modules</td>
<td>7-1</td>
</tr>
<tr>
<td></td>
<td>Introduction</td>
<td>7-1</td>
</tr>
<tr>
<td></td>
<td>Implementation Rules</td>
<td>7-2</td>
</tr>
<tr>
<td></td>
<td>Entry Points of an I/O Module</td>
<td>7-4</td>
</tr>
<tr>
<td></td>
<td>Example of an I/O Module</td>
<td>7-7</td>
</tr>
<tr>
<td>VIII</td>
<td>Interprocess Communication</td>
<td>8-1</td>
</tr>
<tr>
<td></td>
<td>Overview</td>
<td>8-1</td>
</tr>
<tr>
<td></td>
<td>IPC Terminology</td>
<td>8-4</td>
</tr>
<tr>
<td></td>
<td>IPC Protocol</td>
<td>8-8</td>
</tr>
<tr>
<td></td>
<td>Sending Wakeups</td>
<td>8-10</td>
</tr>
<tr>
<td></td>
<td>IPC Subroutines</td>
<td>8-11</td>
</tr>
<tr>
<td></td>
<td>ipc_Error Codes</td>
<td>8-13</td>
</tr>
<tr>
<td></td>
<td>Creating and Destroying Event Channels</td>
<td>8-15</td>
</tr>
<tr>
<td></td>
<td>Invoking an Event-Call Procedure</td>
<td>8-17</td>
</tr>
<tr>
<td></td>
<td>Going Blocked on an Event Channel</td>
<td>8-18</td>
</tr>
<tr>
<td></td>
<td>Reading an Event-Wait Channel</td>
<td>8-20</td>
</tr>
<tr>
<td></td>
<td>Control Functions</td>
<td>8-23</td>
</tr>
<tr>
<td></td>
<td>Masking or Assigning Priority to Event Channels</td>
<td>8-27</td>
</tr>
<tr>
<td></td>
<td>An Example Using Event-Wait Channels</td>
<td>8-29</td>
</tr>
<tr>
<td></td>
<td>An Example Using Event-Call Channels</td>
<td>8-35</td>
</tr>
<tr>
<td>IX</td>
<td>Interprocess Data Base Sharing</td>
<td>9-1</td>
</tr>
<tr>
<td></td>
<td>Introduction</td>
<td>9-1</td>
</tr>
<tr>
<td></td>
<td>The Locking Mechanism</td>
<td>9-2</td>
</tr>
<tr>
<td></td>
<td>The set_lock Subroutine</td>
<td>9-4</td>
</tr>
<tr>
<td></td>
<td>An Example of Locking</td>
<td>9-6</td>
</tr>
<tr>
<td>X</td>
<td>Intraprocess Timer Management</td>
<td>10-1</td>
</tr>
<tr>
<td></td>
<td>Introduction</td>
<td>10-1</td>
</tr>
<tr>
<td></td>
<td>Timer Management Terminology</td>
<td>10-2</td>
</tr>
<tr>
<td></td>
<td>timer_manager_Generic Arguments</td>
<td>10-3</td>
</tr>
<tr>
<td></td>
<td>timer_manager_Entry Points</td>
<td>10-5</td>
</tr>
<tr>
<td></td>
<td>Blocking a Process</td>
<td>10-7</td>
</tr>
<tr>
<td></td>
<td>Using Call Timers</td>
<td>10-8</td>
</tr>
<tr>
<td></td>
<td>Using Wakeup Timers</td>
<td>10-10</td>
</tr>
<tr>
<td></td>
<td>Resetting and Inhibiting Timers</td>
<td>10-12</td>
</tr>
<tr>
<td></td>
<td>Standard System Handlers</td>
<td>10-15</td>
</tr>
<tr>
<td></td>
<td>Two Examples Using Timers</td>
<td>10-16</td>
</tr>
<tr>
<td></td>
<td>timer_manager_Summary</td>
<td>10-22</td>
</tr>
<tr>
<td>XI</td>
<td>The Stack and Argument Lists</td>
<td>11-1</td>
</tr>
<tr>
<td></td>
<td>The Process Stack Segment</td>
<td>11-1</td>
</tr>
<tr>
<td></td>
<td>The Stack Header</td>
<td>11-2</td>
</tr>
<tr>
<td></td>
<td>The Stack Frame</td>
<td>11-4</td>
</tr>
</tbody>
</table>
## CONTENTS (con't)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic XII</td>
<td>Special Programming Techniques</td>
<td>12-1</td>
</tr>
<tr>
<td></td>
<td>Calling a Procedure on the Fly</td>
<td>12-1</td>
</tr>
<tr>
<td></td>
<td>Building Data Segments</td>
<td>12-4</td>
</tr>
<tr>
<td></td>
<td>Creating an Error Table</td>
<td>12-9</td>
</tr>
<tr>
<td>Topic XIII</td>
<td>The Process Environment</td>
<td>13-1</td>
</tr>
<tr>
<td></td>
<td>The Standard Process Environment</td>
<td>13-1</td>
</tr>
<tr>
<td></td>
<td>Process Creation in the Supervisor Ring</td>
<td>13-3</td>
</tr>
<tr>
<td></td>
<td>Process Initialization in the User Ring</td>
<td>13-5</td>
</tr>
<tr>
<td></td>
<td>Modifying the Process Environment</td>
<td>13-9</td>
</tr>
<tr>
<td></td>
<td>Project Administration</td>
<td>13-11</td>
</tr>
<tr>
<td></td>
<td>Closed Subsystems</td>
<td>13-13</td>
</tr>
<tr>
<td></td>
<td>Limited Subsystems</td>
<td>13-14</td>
</tr>
<tr>
<td></td>
<td>New Subsystems</td>
<td>13-18</td>
</tr>
<tr>
<td>Topic XIV</td>
<td>Dialing Terminals to a Process</td>
<td>14-1</td>
</tr>
<tr>
<td></td>
<td>Overview</td>
<td>14-1</td>
</tr>
<tr>
<td></td>
<td>Implementation of the Dial Facility</td>
<td>14-2</td>
</tr>
<tr>
<td></td>
<td>dial_manager</td>
<td>14-3</td>
</tr>
<tr>
<td></td>
<td>Dialing Terminals to a Process</td>
<td>14-4</td>
</tr>
<tr>
<td></td>
<td>Subroutines</td>
<td>14-5</td>
</tr>
<tr>
<td></td>
<td>The 'dial' Command</td>
<td>14-7</td>
</tr>
<tr>
<td></td>
<td>An Example</td>
<td>14-8</td>
</tr>
<tr>
<td></td>
<td>Dialing Out to a Terminal</td>
<td>14-13</td>
</tr>
<tr>
<td></td>
<td>dial_manager Entry Points</td>
<td>14-14</td>
</tr>
<tr>
<td>Topic XV</td>
<td>Message Segment Facility</td>
<td>15-1</td>
</tr>
<tr>
<td></td>
<td>What Is It?</td>
<td>15-1</td>
</tr>
<tr>
<td></td>
<td>Applications</td>
<td>15-2</td>
</tr>
<tr>
<td></td>
<td>The Message Segment</td>
<td>15-3</td>
</tr>
<tr>
<td></td>
<td>Layered Design</td>
<td>15-6</td>
</tr>
<tr>
<td></td>
<td>Primitive Message Segment Facility</td>
<td>15-8</td>
</tr>
<tr>
<td></td>
<td>Extended Access</td>
<td>15-9</td>
</tr>
<tr>
<td></td>
<td>message_segment_Subroutine Summary</td>
<td>15-10</td>
</tr>
<tr>
<td></td>
<td>Message Segment Facility Illustrative Example</td>
<td>15-12</td>
</tr>
<tr>
<td>Topic XVI</td>
<td>Program Library Management</td>
<td>16-1</td>
</tr>
<tr>
<td></td>
<td>Introduction</td>
<td>16-1</td>
</tr>
<tr>
<td></td>
<td>Organization of Program Libraries</td>
<td>16-2</td>
</tr>
<tr>
<td></td>
<td>Naming Conventions</td>
<td>16-4</td>
</tr>
<tr>
<td></td>
<td>A Typical Program Library</td>
<td>16-6</td>
</tr>
<tr>
<td></td>
<td>Program Library Management Tools</td>
<td>16-9</td>
</tr>
<tr>
<td></td>
<td>Installation Tools</td>
<td>16-10</td>
</tr>
<tr>
<td></td>
<td>Library Descriptor Tools</td>
<td>16-20</td>
</tr>
<tr>
<td>Appendix A</td>
<td>AIM</td>
<td>A-1</td>
</tr>
<tr>
<td></td>
<td>Concepts</td>
<td>A-1</td>
</tr>
<tr>
<td></td>
<td>Commands and Subroutines</td>
<td>A-3</td>
</tr>
</tbody>
</table>
## CONTENTS (con't)

<table>
<thead>
<tr>
<th>Appendix</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix B</td>
<td>Program Listings.</td>
<td>B-1</td>
</tr>
<tr>
<td></td>
<td>user_init_admin</td>
<td>B-1</td>
</tr>
<tr>
<td></td>
<td>user_real_init_admin</td>
<td>B-3</td>
</tr>
<tr>
<td></td>
<td>process_overseer</td>
<td>B-6</td>
</tr>
<tr>
<td></td>
<td>project_start_up</td>
<td>B-10</td>
</tr>
<tr>
<td></td>
<td>listen</td>
<td>B-15</td>
</tr>
<tr>
<td>Appendix C</td>
<td>Encoding of Channel Names</td>
<td>C-1</td>
</tr>
<tr>
<td>Appendix D</td>
<td>Instructor Code for IPC Workshop.</td>
<td>D-1</td>
</tr>
<tr>
<td>Appendix E</td>
<td>Standard Process Overseers</td>
<td>E-1</td>
</tr>
<tr>
<td>Appendix F</td>
<td>Gate and Message Segment Examples</td>
<td>F-1</td>
</tr>
<tr>
<td>Appendix G</td>
<td>Advanced Dial Facility Example</td>
<td>G-1</td>
</tr>
<tr>
<td>Appendix W</td>
<td>Workshops</td>
<td>W-1</td>
</tr>
<tr>
<td></td>
<td>Workshop One</td>
<td>W-1</td>
</tr>
<tr>
<td></td>
<td>Workshop Two</td>
<td>W-3</td>
</tr>
<tr>
<td></td>
<td>Workshop Three</td>
<td>W-6</td>
</tr>
<tr>
<td></td>
<td>Workshop Four</td>
<td>W-7</td>
</tr>
<tr>
<td></td>
<td>Workshop Five</td>
<td>W-9</td>
</tr>
<tr>
<td></td>
<td>Workshop Six</td>
<td>W-12</td>
</tr>
<tr>
<td></td>
<td>Workshop Seven</td>
<td>W-13</td>
</tr>
</tbody>
</table>
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)
STUDENT BACKGROUND
Multics Subsystem Programming (F15D)

3. Check the boxes for which you have any related experience. (May be other than Honeywell's)
[ ] PL/1  [ ] 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:
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.

<table>
<thead>
<tr>
<th>CAN'T JUDGE</th>
<th>TOO SHORT</th>
<th>ABOUT RIGHT</th>
<th>TOO LONG</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 3 4 5 6 7 8 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

COMMENTS

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

<table>
<thead>
<tr>
<th>CAN'T JUDGE</th>
<th>NOT TECH ENOUGH</th>
<th>ABOUT RIGHT</th>
<th>TOO TECH</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 3 4 5 6 7 8 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

COMMENTS

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

<table>
<thead>
<tr>
<th>CAN'T JUDGE</th>
<th>POOR</th>
<th>GOOD</th>
<th>EXCELLENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 3 4 5 6 7 8 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

COMMENTS

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

<table>
<thead>
<tr>
<th>CAN'T JUDGE</th>
<th>POOR</th>
<th>GOOD</th>
<th>EXCELLENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 3 4 5 6 7 8 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

COMMENTS

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

<table>
<thead>
<tr>
<th>CAN'T JUDGE</th>
<th>POOR</th>
<th>GOOD</th>
<th>EXCELLENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

COMMENTS

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

<table>
<thead>
<tr>
<th>CAN'T JUDGE</th>
<th>TOO</th>
<th>LITTLE</th>
<th>ABOUT</th>
<th>TOO</th>
<th>MUCH</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

COMMENTS

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

<table>
<thead>
<tr>
<th>CAN'T JUDGE</th>
<th>POOR</th>
<th>GOOD</th>
<th>EXCELLENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

COMMENTS

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

<table>
<thead>
<tr>
<th>CAN'T JUDGE</th>
<th>POOR</th>
<th>GOOD</th>
<th>EXCELLENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>CAN'T JUDGE</th>
<th>POOR</th>
<th>GOOD</th>
<th>EXCELLENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

COMMENTS

---

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

<table>
<thead>
<tr>
<th>CAN'T JUDGE</th>
<th>POOR</th>
<th>GOOD</th>
<th>EXCELLENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

COMMENTS

---

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

<table>
<thead>
<tr>
<th>CAN'T JUDGE</th>
<th>POOR</th>
<th>GOOD</th>
<th>EXCELLENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

COMMENTS

---

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

<table>
<thead>
<tr>
<th>CAN'T JUDGE</th>
<th>POOR</th>
<th>GOOD</th>
<th>EXCELLENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

COMMENTS

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

<table>
<thead>
<tr>
<th>CAN'T JUDGE</th>
<th>POOR</th>
<th>GOOD</th>
<th>EXCELLENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 3 4 5 6 7 8 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

COMMENTS

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

<table>
<thead>
<tr>
<th>CAN'T JUDGE</th>
<th>POOR</th>
<th>GOOD</th>
<th>EXCELLENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 3 4 5 6 7 8 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

COMMENTS

15. What did you like most about this course?

16. What did you like least 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, ...

• 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
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).
### STORAGE SYSTEM SUBROUTINES

<table>
<thead>
<tr>
<th>F15C</th>
<th>F15D</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CREATING STORAGE SYSTEM ENTITIES</strong></td>
<td></td>
</tr>
<tr>
<td>hcs$_append_branch</td>
<td></td>
</tr>
<tr>
<td>hcs$_append_branchx</td>
<td></td>
</tr>
<tr>
<td>hcs$_append_link</td>
<td></td>
</tr>
<tr>
<td>hcs$<em>create_branch</em></td>
<td></td>
</tr>
<tr>
<td>hcs$_make_seg</td>
<td></td>
</tr>
<tr>
<td><strong>DELETING STORAGE SYSTEM ENTITIES</strong></td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td></td>
</tr>
<tr>
<td>hcs$_delentry_file</td>
<td></td>
</tr>
<tr>
<td>hcs$_delentry_seg</td>
<td></td>
</tr>
<tr>
<td>hcs$_del_dir_tree (AK92)</td>
<td></td>
</tr>
<tr>
<td><strong>OBTAINING STATUS INFORMATION</strong></td>
<td></td>
</tr>
<tr>
<td>hcs$_status</td>
<td>hcs$_get_author</td>
</tr>
<tr>
<td>hcs$_status_long</td>
<td>hcs$_get_bc_author</td>
</tr>
<tr>
<td>hcs$_status_short</td>
<td>hcs$_get_link_target</td>
</tr>
<tr>
<td>hcs$_status_minf</td>
<td>hcs$_get_max_length</td>
</tr>
<tr>
<td>hcs$_status_mins</td>
<td>hcs$_get_max_length_seg</td>
</tr>
<tr>
<td></td>
<td>hcs$_get_safety_sw</td>
</tr>
<tr>
<td></td>
<td>hcs$_get_safety_sw_seg</td>
</tr>
<tr>
<td></td>
<td>(hcs$_set_max_length)</td>
</tr>
<tr>
<td></td>
<td>(hcs$_set_max_length_seg)</td>
</tr>
<tr>
<td></td>
<td>(hcs$_set_safety_sw)</td>
</tr>
<tr>
<td></td>
<td>(hcs$_set_safety_sw_seg)</td>
</tr>
<tr>
<td>F15C</td>
<td>F15D</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>WORKING, DEFAULT, AND PROCESS DIRECTORIES</strong></td>
<td></td>
</tr>
<tr>
<td>change_wdir</td>
<td></td>
</tr>
<tr>
<td>get_default_wdir (AK92)</td>
<td></td>
</tr>
<tr>
<td>get_pdir</td>
<td></td>
</tr>
<tr>
<td>get_wdir</td>
<td></td>
</tr>
<tr>
<td><strong>MANIPULATING THE ADDRESS AND NAME SPACES</strong></td>
<td></td>
</tr>
<tr>
<td>hcs $fs_get_path_name</td>
<td></td>
</tr>
<tr>
<td>hcs $fs_get_ref_name</td>
<td></td>
</tr>
<tr>
<td>hcs $fs_get_seg_ptr</td>
<td></td>
</tr>
<tr>
<td>hcs $initiate</td>
<td></td>
</tr>
<tr>
<td>hcs $initiate_count</td>
<td></td>
</tr>
<tr>
<td>hcs $make_seg</td>
<td></td>
</tr>
<tr>
<td>hcs $terminate_file</td>
<td></td>
</tr>
<tr>
<td>hcs $terminate_name</td>
<td></td>
</tr>
<tr>
<td>hcs $terminate_noname</td>
<td></td>
</tr>
<tr>
<td>hcs $terminate_seg</td>
<td></td>
</tr>
<tr>
<td>term $refname</td>
<td></td>
</tr>
<tr>
<td>term $seg_ptr</td>
<td></td>
</tr>
<tr>
<td>term $single_refname</td>
<td></td>
</tr>
<tr>
<td>term $term</td>
<td></td>
</tr>
<tr>
<td>term $unsnap</td>
<td></td>
</tr>
<tr>
<td><strong>MULTISEGMENT FILES</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>msf_manager $acl_add</td>
</tr>
<tr>
<td></td>
<td>msf_manager $acl_delete</td>
</tr>
<tr>
<td></td>
<td>msf_manager $acl_list</td>
</tr>
<tr>
<td></td>
<td>msf_manager $acl_replace</td>
</tr>
<tr>
<td></td>
<td>msf_manager $adjust</td>
</tr>
<tr>
<td></td>
<td>msf_manager $close</td>
</tr>
<tr>
<td></td>
<td>msf_manager $get_ptr</td>
</tr>
<tr>
<td></td>
<td>msf_manager $open</td>
</tr>
</tbody>
</table>

_Not To Be Reproduced_ 1-10 F15D
<table>
<thead>
<tr>
<th><strong>F15C</strong></th>
<th><strong>F15D</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NAMING AND MOVING DIRECTORY ENTRIES</strong></td>
<td></td>
</tr>
<tr>
<td>hcs $chname_file</td>
<td></td>
</tr>
<tr>
<td>hcs $chname_seg</td>
<td></td>
</tr>
<tr>
<td>hcs $fs_move_file</td>
<td></td>
</tr>
<tr>
<td>hcs $fs_move_seg</td>
<td></td>
</tr>
<tr>
<td><strong>AFFECTING LENGTH OF ENTRIES</strong></td>
<td></td>
</tr>
<tr>
<td>adjust_bit_count</td>
<td></td>
</tr>
<tr>
<td>hcs $set_bc</td>
<td></td>
</tr>
<tr>
<td>hcs $set_bc_seg</td>
<td></td>
</tr>
<tr>
<td>hcs $truncate_file</td>
<td></td>
</tr>
<tr>
<td>hcs $truncate_seg</td>
<td></td>
</tr>
<tr>
<td><strong>MANIPULATING PATHNAMES</strong></td>
<td></td>
</tr>
<tr>
<td>absolute_pathname</td>
<td></td>
</tr>
<tr>
<td>absolute_pathname $add_suffix</td>
<td></td>
</tr>
<tr>
<td>expand_pathname</td>
<td></td>
</tr>
<tr>
<td>expand_pathname $add_suffix</td>
<td></td>
</tr>
<tr>
<td><strong>MANIPULATING THE STAR AND EQUAL CONVENTION</strong></td>
<td></td>
</tr>
<tr>
<td>check_star_name</td>
<td></td>
</tr>
<tr>
<td>get_equal_name</td>
<td></td>
</tr>
<tr>
<td>hcs $star</td>
<td></td>
</tr>
<tr>
<td>match_star_name</td>
<td></td>
</tr>
<tr>
<td><strong>F15C</strong></td>
<td><strong>F15D</strong></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td><em>AREAS</em></td>
<td></td>
</tr>
<tr>
<td>get_system_free_area_(AK92)</td>
<td>area_info</td>
</tr>
<tr>
<td>get_group_id</td>
<td>area_status *</td>
</tr>
<tr>
<td>get_group_id_$tag_star</td>
<td>create_area *</td>
</tr>
<tr>
<td>hcs_$add_acl_entries</td>
<td>define_area *</td>
</tr>
<tr>
<td>hcs_$delete_acl_entries</td>
<td>release_area *</td>
</tr>
<tr>
<td>hcs_$delete_dir_acl_entries</td>
<td>set_system_storage *</td>
</tr>
<tr>
<td>hcs_$fs_get_mode</td>
<td>set_user_storage *</td>
</tr>
<tr>
<td>hcs_$list_acl</td>
<td></td>
</tr>
<tr>
<td>hcs_$list_dir_acl</td>
<td></td>
</tr>
<tr>
<td>hcs_$replace_acl</td>
<td></td>
</tr>
<tr>
<td>hcs_$replace_dir_acl</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><em>SECURITY</em></td>
<td></td>
</tr>
<tr>
<td>get_group_id</td>
<td>cross_ring</td>
</tr>
<tr>
<td>get_group_id_$tag_star</td>
<td>cross_ring_io $allow_cross</td>
</tr>
<tr>
<td>hcs_$add_acl_entries</td>
<td>cu_$level_get (AG93)</td>
</tr>
<tr>
<td>hcs_$delete_acl_entries</td>
<td>cu_$level_set (AG93)</td>
</tr>
<tr>
<td>hcs_$fs_get_mode</td>
<td>get_ring</td>
</tr>
<tr>
<td>hcs_$list_acl</td>
<td></td>
</tr>
<tr>
<td>hcs_$list_dir_acl</td>
<td></td>
</tr>
<tr>
<td>hcs_$replace_acl</td>
<td></td>
</tr>
<tr>
<td>hcs_$replace_dir_acl</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><em>COMMANDS (INCLUDED FOR COMPLETENESS)</em></td>
<td></td>
</tr>
</tbody>
</table>

*Not To Be Reproduced*  1-12  (End Of Topic)
TOPIC II
Storage System Subroutines

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtaining Status Information</td>
<td>2-1</td>
</tr>
<tr>
<td>Multisegment Files</td>
<td>2-8</td>
</tr>
<tr>
<td>Temporary Segments</td>
<td>2-14</td>
</tr>
</tbody>
</table>
OBTAINING STATUS INFORMATION

- 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
OBTAINING STATUS INFORMATION

- 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
OBTAINING STATUS INFORMATION

- 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
OSTAINING 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
OBTAINING STATUS INFORMATION

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 entry char (argl) based (argp);

dcl dir char (168);

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 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; change
end;

FINISH: end STATUS;
OBTAINING STATUS INFORMATION

!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.

Not To Be Reproduced
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
MULTISEGMENT FILES

- **msf_manager_$open**

```c
// 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**

```c
// 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**
MULTISEGMENT FILES

!pr MSF.p11

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
MULTISEGMENT FILES

!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
**MULTISEGMENT FILES**

- `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

<table>
<thead>
<tr>
<th>hcs</th>
<th>msf_manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>list_acl</td>
<td>acl_list</td>
</tr>
<tr>
<td>replace_acl</td>
<td>acl_replace</td>
</tr>
<tr>
<td>add_acl_entries</td>
<td>acl_add</td>
</tr>
<tr>
<td>delete_acl_entries</td>
<td>acl_delete</td>
</tr>
</tbody>
</table>
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
• 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_
TEMPORARY SEGMENTS

- `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>!BBBJHmQJKkkPNPb.temp.0354
  355 >process_dir_dir>!BcDBdwpbBBBBBB>!BBBJHmQJkkkWlf.temp.0355

! list_temp_segments
  12 Segments, 11 Free
!
BBBJHmQJDkmGxW.temp.0315 command_processor_

Not To Be Reproduced 2-17
(End Of Topic)
# TOPIC III

Storage System Subroutines (cont)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star and Equal Conventions</td>
<td>3-1</td>
</tr>
<tr>
<td>Area Manipulation</td>
<td>3-8</td>
</tr>
<tr>
<td>Introduction</td>
<td>3-8</td>
</tr>
<tr>
<td>Area Format</td>
<td>3-9</td>
</tr>
<tr>
<td>Area Manipulating Subroutines</td>
<td>3-12</td>
</tr>
<tr>
<td>Area Related Commands</td>
<td>3-16</td>
</tr>
</tbody>
</table>

3-1 F15D
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_
STAR AND EQUAL CONVENTIONS

- 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**

Not To Be Reproduced 3-2 F15D
STAR AND EQUAL CONVENTIONS

area_ptr

star_entry_ptr

star_names_ptr

Not To Be Reproduced
STAR AND EQUAL CONVENTIONS

!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;
    free star_names_ptr->star_names;
    free star_entry_ptr->star_entries;
end lspl1;
STAR AND EQUAL CONVENTIONS

13 segments match **.pl1:

dehs.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
gettocl_.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.
STAR AND EQUAL CONVENTIONS

• get_equal_name_

[] call get_equal_name_ (entryname, equal_name, target_name, code);

[] CONSTRUCTS A TARGET NAME FROM AN ENTRYNAME AND AN EQUALNAME

[] EXAMPLE

<table>
<thead>
<tr>
<th>ENTRYNAME</th>
<th>EQUAL NAME</th>
<th>TARGET NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.b.c</td>
<td>new.=.=</td>
<td>new.b.c</td>
</tr>
<tr>
<td>abc.def.ghi</td>
<td>.%%.5</td>
<td>abc.de.5</td>
</tr>
</tbody>
</table>

• 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

Not To Be Reproduced
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

I EXTENSIBILITY

I SOME AREAS ARE LIMITED TO THE SIZE OF A SEGMENT (NON-EXTENSIBLE AREAS)

I OTHERS CAN "GROW" INTO TEMP SEGMENTS IN THE PROCESS DIRECTORY (EXTENSIBLE AREAS)

I FREEING OF SPACE WITHIN AN AREA FOR REUSE

I SOME AREAS HAVE BLOCKS OF FREED SPACE MAINTAINED IN LINKED LISTS AVAILABLE FOR REUSE (FREEING AREAS)

I OTHERS DO NOT REUSE FREED SPACE IN THE AREA -- ALL ALLOCATIONS ARE DONE IN "VIRGIN AREA" (NO-FREEING AREAS)

I NO-FREEING AREAS ARE OBVIOUSLY HANDLED MUCH FASTER BY THE AREA MANAGER

• ALL AREAS HAVE 24 WORD HEADERS

I 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
• WHEN SPACE IS FREED THE AREA MANAGER:

1. LOOKS AT THE FIRST 2 WORDS IN THE BLOCK TO DETERMINE SIZE OF THE BLOCK

2. MERGES SMALLER ADJACENT BLOCKS IF POSSIBLE

3. THREADS THE FREED BLOCKS ONTO THE APPROPRIATE LIST

• NOTE

AREA MANAGER DOES NOT UPDATE BIT COUNT
**AREA FORMAT**

**AREA MANIPULATING SUBROUTINES**

- **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 FORMAT
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.inc.pl1

- Region being initialized

  - Is pointed to by area_info.areap

  - Is automatically acquired from process directory TEMPSEG pool if area_info.areap = null()
• 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)
AREA FORMAT
AREA MANIPULATING SUBROUTINES

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 area_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

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>4-1</td>
</tr>
<tr>
<td>Initial ACL's.</td>
<td>4-2</td>
</tr>
<tr>
<td>Rings.</td>
<td>4-4</td>
</tr>
<tr>
<td>Introduction</td>
<td>4-4</td>
</tr>
<tr>
<td>Ring Brackets</td>
<td>4-5</td>
</tr>
<tr>
<td>Ring Bracket Subroutines</td>
<td>4-8</td>
</tr>
<tr>
<td>Gates.</td>
<td>4-10</td>
</tr>
<tr>
<td>Validation Level</td>
<td>4-15</td>
</tr>
<tr>
<td>Cross Ring I/O</td>
<td>4-17</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>ACL ENTRY POINTS</th>
<th>INACL ENTRY POINTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$add_acl_entries</td>
<td>$add_inacl_entries</td>
</tr>
<tr>
<td>$add_dir_acl_entries</td>
<td>$add_dir_inacl_entries</td>
</tr>
<tr>
<td>$delete_acl_entries</td>
<td>$delete_inacl_entries</td>
</tr>
<tr>
<td>$delete_dir_acl_entries</td>
<td>$delete_dir_inacl_entries</td>
</tr>
<tr>
<td>$list_acl</td>
<td>$list_inacl</td>
</tr>
<tr>
<td>$list_dir_acl</td>
<td>$list_dir_inacl</td>
</tr>
<tr>
<td>$replace_acl</td>
<td>$replace_inacl</td>
</tr>
<tr>
<td>$replace_dir_acl</td>
<td>$replace_dir_inacl</td>
</tr>
</tbody>
</table>
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
  1. PROTECTION OF SUPERVISOR FROM USER PROGRAMS
  2. PROTECTION OF SUBSYSTEM DATA BASE FROM DIRECT ACCESS

• CLARIFICATION OF MISCONCEPTION
  1. ALL SEGMENTS ARE NOT "IN" JUST ONE RING
  2. SEGMENTS MAY "SPAN" SEVERAL RINGS
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

\[ \begin{array}{l}
\text{THE FIRST TWO NUMBERS DELIMIT THE EXECUTE BRACKET} \\
\text{THE SECOND AND THIRD NUMBERS DELIMIT THE GATE BRACKET}
\end{array} \]

EXECUTE AND GATE BRACKETS
RINGS
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

<table>
<thead>
<tr>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>EXECUTE</th>
<th>GATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• 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
RINGS
RING BRACKETS

RING MECHANISM SUMMARY

WRITE BRACKET

READ BRACKET

EXECUTE BRACKET

GATE BRACKET

RING OF EXECUTION

CORRESPONDING PERMITTED ACTIONS*

READ, WRITE, EXECUTE (WITH RING CHANGE)

READ, WRITE, AND EXECUTE

READ, EXECUTE

EXECUTE (IF A GATE ONLY, AND WITH RING CHANGE)

NONE

* SUBJECT, OF COURSE, TO ACL AND AIM

Not To Be Reproduced
RINGS
RING BRACKET SUBROUTINES

- get_ring_

  \[ current \_ring = \text{get\_ring\_}(); \]

  \[ \text{RETURNS THE USER'S CURRENT RING OF EXECUTION} \]

- hcs\_get\_ring\_brackets

  \[ \text{call hcs\_get\_ring\_brackets (dir\_name, entryname, rb, code);} \]

  \[ \text{RETURNS (r1, r2, r3) FOR A SPECIFIED SEGMENT} \]

- hcs\_get\_dir\_ring\_brackets

  \[ \text{call hcs\_get\_dir\_ring\_brackets (dir\_name, entryname, drb, code);} \]

  \[ \text{RETURNS (r1, r2) FOR A SPECIFIED DIRECTORY} \]
RINGS
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
RINGS

GATES

• 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)

CROSSING RING BOUNDARIES

RING 5

RING 4

put find

Object entrypointnames:
put find
Ring brackets:
\{4, 5, 5\}

CALLS

put_find_gate_

Object entrypointnames:
put find
Ring brackets:
\{4, 4, 5\}

CALLS

put_find_gate_util_
pf

Object entrypointnames:
put find
Ring brackets:
\{4, 4, 4\}

READS AND WRITES

emp

Ring brackets:
\{4, 4, 4\}
• 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_1373
(system_library >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
RINGS
VALIDATION LEVEL

• 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
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_
user_i/o tty_login_channel

user_input syn_user_i/o
user_output _syn_user_i/o
error_output syn_user_i/o

pr cross.pl1

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, "O"b, code);
end cross;

pr gate.alm
#include gate_macros
gate_info

gate allow,allow,allow,0
end
RINGS
CROSS RING I/O

!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, "$file_ \text{>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
  \text{>udd>MED>NDibble>gate>file}
4, 4, 4
  \text{>udd>MED>NDibble>gate>cross}
4, 5, 5
  \text{>udd>MED>NDibble>gate>gate}
4, 4, 5
  \text{>udd>MED>NDibble>gate>allow}
4, 4, 4
RINGS
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>MDibble>gate>file

• TWO MAJOR POINTS TO REMEMBER

  WORKING DIRECTORIES ARE PER RING

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

Not To Be Reproduced 4-20 (End Of Topic)
TOPIC V
The Command Environment

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 aln) IS THE TOOL USED BY SUBSYSTEM DESIGNERS TO ACCOMPLISH THE FOLLOWING BASIC TASKS:

<table>
<thead>
<tr>
<th>WRITING COMMAND OR ACTIVE FUNCTION PROCEDURES</th>
</tr>
</thead>
<tbody>
<tr>
<td>cu $arg_count</td>
</tr>
<tr>
<td>cu $arg_ptr</td>
</tr>
<tr>
<td>cu $at arg_count</td>
</tr>
<tr>
<td>cu $at return arg</td>
</tr>
<tr>
<td>cu $at arg_ptr</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MODIFYING THE STANDARD COMMAND ENVIRONMENT</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>WRITING A COMMAND PROCESSOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>cu $generate_call</td>
</tr>
</tbody>
</table>
MODIFYING THE STANDARD COMMAND ENVIRONMENT

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.
MODIFYING THE STANDARD COMMAND ENVIRONMENT

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

```
listen_$listen_
CALL

cu_$ready_proc
CALL

print_ready_message_
PRINTS READY MESSAGE
IFF cu_$get_ready_mode RETURNS "1" b

RETURN

cu_$ready_proc
RETURN

listen_$listen_
"LISTENS" FOR NEXT LINE
```
MODIFYING THE STANDARD COMMAND ENVIRONMENT

CURRENT READY PROCEDURE

• 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
MODIFYING THE STANDARD COMMAND ENVIRONMENT

CURRENT COMMAND PROCESSOR

USER TYPES "pwd"

 listen $listen

CALL PASSENGER A LINE PTR, LINE LENGTH, & CODE

 cu_$cp

CALL PASSENGER A LINE PTR, LINE LENGTH, & CODE

abbrev $abbrev

THE "CURRENT COMMAND PROCESSOR"

CALL PASSENGER A LINE PTR, LINE LENGTH, & CODE

command _processor $command _processor

CALL

pwdSpwd

PRINTS WORKING DIRECTORY

RETURN

command _processor $command _processor

RETURN

abbrev $abbrev

RETURN

cu_$cp

RETURN

listen $listen

WHERE THE CURRENT COMMAND PROCESSOR FITS IN
MODIFYING THE STANDARD COMMAND ENVIRONMENT

CURRENT COMMAND PROCESSOR

• 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
MODIFYING THE STANDARD COMMAND ENVIRONMENT

COMMAND LEVEL INTERMEDIARY

USER TYPES "print foo"

WHERE THE COMMAND LEVEL INTERMEDIARY FITS IN

CALL

listen_$listen_

CALL

cu_$cp

CALL

abbrev_$abbrev_

CALL

command_processor_$command_processor_

QUIT!

print$print

DURING PRINTING OF foo, USER HITS QUIT KEY

CALL

; { quit CONDITION SIGNALLED AND A HANDLER IS FOUND

default_error_handler_$swall

PRINTS "QUIT" ON TERMINAL

CALL

cu_$cl

CALL

get_to_cl_Sunclaim_signal

CALL

listen_$release_stack

CALL

cu_$ready_proc

CALL

print_ready_message_

PRINT READY MESSAGE WITH "level 2" CLAUSE
IFF cu_$get_ready_mode RETURNS "1" b

RETURN

cu_$ready_proc

"LISTENS" FOR FIRST COMMAND IN THIS LEVEL

RETURN

listen_$release_stack
MODIFYING THE STANDARD COMMAND ENVIRONMENT

COMMAND LEVEL INTERMEDIARY

- MANIPULATING THE COMMAND LEVEL INTERMEDIARY

```
\indent \indent cu_$cl (AG93)
\indent \indent \indent INVOKES THE CURRENT COMMAND LEVEL INTERMEDIARY
\indent \indent \indent CALLED BY THE STANDARD ERROR HANDLERS

\indent \indent cu$_{set\_cl\_intermediary} (AG93)
\indent \indent \indent ESTABLISHES THE SPECIFIED PROCEDURE AS THE CURRENT COMMAND LEVEL INTERMEDIARY
\indent \indent \indent NOTE THAT AN INTERMEDIARY IS USED IN ABSENTEE PROCESSES TO FORCE PROCESS TERMINATION "WHEN AN ATTEMPT IS MADE TO REENTER COMMAND LEVEL"

\indent \indent cu$_{get\_cl\_intermediary} (AG93)
\indent \indent \indent RETURNS A NULL ENTRY VALUE IF THE CURRENT COMMAND LEVEL INTERMEDIARY IS THE DEFAULT (get_to_cl$_{unclaimed\_signal}$)
\indent \indent \indent OTHERWISE RETURNS THE ENTRY VALUE OF THE CURRENT COMMAND LEVEL INTERMEDIARY
```
MODIFYING THE STANDARD COMMAND ENVIRONMENT

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)

```
RETURN 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
MODIFYING THE STANDARD COMMAND ENVIRONMENT
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
MODIFYING THE STANDARD COMMAND ENVIRONMENT

SOME MISCELLANEOUS CU ENTRY POINTS

• 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 hos_make_entry, which accepts pathnames
MODIFYING THE STANDARD COMMAND ENVIRONMENT
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;
MODIFYING THE STANDARD COMMAND ENVIRONMENT

AN EXAMPLE

r 19:26 0.132 0
!
print my_intermediary.pl1 1

my_intermediary: proc;

dcl get cl $unclaimed signal entry;
dcl ioa_entry options (variable);

    call ioa_ ("TYPE 'start' TO RESTART PROCESS
TYPE 'release' OR 'tr' 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
!

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

Not To Be Reproduced 5-16
MODIFYING THE STANDARD COMMAND ENVIRONMENT

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 115 (level 6).

stack
13 simple_command_processor 12211
12 command_processor 111014
11 abbrev 7507
10 release_stack 7755
9 unclaimed_signal 27010
8 my_intermediary (line 8)
7 wall 2602
6 block 154
5 tty_get_line 5763
4 audit_get_line 5073
3 listen 7566
2 project_start_up 41673
1 user_init_admIn To2376 (alm)

! q
r 19:29 0.980 81 level 2

! release
r 19:29 0.044 0

! change cl
Current command processor is 30512676
Current intermediary is 45626
r 19:29 0.046 0

! lrn 456

456 > udd > MED > NDibble > my_intermediary
my intermediary
r 19:30 0.047 0
MODIFYING THE STANDARD COMMAND ENVIRONMENT

AN EXAMPLE

YOU ARE NOW READY FOR WORKSHOP #2
## TOPIC VI

Advanced Multics I/O

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Review</td>
<td>6-1</td>
</tr>
<tr>
<td>Control Orders</td>
<td>6-2</td>
</tr>
<tr>
<td>Useful tty Control Orders</td>
<td>6-3</td>
</tr>
<tr>
<td>Useful vfIe Control Orders</td>
<td>6-7</td>
</tr>
<tr>
<td>Review of IOCB's</td>
<td>6-9</td>
</tr>
<tr>
<td>Synonyming</td>
<td>6-14</td>
</tr>
<tr>
<td>iox Entry Points Used in I/O Modules</td>
<td>6-19</td>
</tr>
</tbody>
</table>
• 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


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

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
CONTROL ORDERS

USEFUL TTY CONTROL ORDERS

\[\text{quit-enable} \]
\[\text{quit-disable} \]

**CAUSE 'quit' SIGNAL PROCESSING TO BE ENABLED OR DISABLED FOR THIS DEVICE (FOR EXAMPLE, THE STANDARD PROCESS CREATION CYCLE PROGRAMS ENABLE QUILTS 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 QUILTS ARE DISABLED, IT IS POSSIBLE TO SIGNAL 'quit' IN A USER PROGRAM, OR VIA THE 'signal' COMMAND**
CONTROL ORDERS

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).
CONTROL ORDERS

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
CONTROL ORDERS

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
CONTROL ORDERS

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
• WHEN \texttt{iox\_attach\_name} IS CALLED IT:

- Initializes some of the elements in the 'IOC\_B' structure

- Calls \texttt{<module\_name>\$<module\_name>attach}

- This entry point in the I/O module finishes the initialization of the 'IOC\_B'

• IT IS THE RESPONSIBILITY OF THE I/O MODULE TO MAINTAIN THE ACCURACY OF THE 'IOC\_B'

• ONLY THE iox ENTRY POINTS RESULTING IN ATTACHMENT OF A SWITCH REQUIRE THE MODULE AS AN INPUT ARGUMENT

- After that time, the 'IOC\_B' "points to" the appropriate entry points in the appropriate module (the user need only provide a pointer to the 'IOC\_B')
REVIEW OF IOCB'S

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) */
REVIEW OF IOCB'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_descrp_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]
REVIEW OF IOCB'S

- `iocb.attach_data_ptr`

  IS OPTIONALLY USED BY THE I/O MODULE TO LOCATE AN INFORMATION STRUCTURE WHOSE FORMAT AND CONTENT IS MODULE-DEPENDENT

- `iocb.open_descip_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
SYNONYMING

!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 >s|>bound_command_loop $err_not_closed (271|2370)
open
"close
get_line
get_chars
put_chars
modes
position
control
read_record
write_record
rewrite_record
delete_record
seek_key
read_key
read_length

!dcn >s|>bound_command_loop 2360
2360 iox_1T14
SYNONYMING

!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)
on_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

SYNONYMING

!io attach inhibited syn_y -inhit close

!io print iocb inhibited
IOCB "inhibited" @ 257|27020 (actual I/O CB is "y" @ 257|27320)
SYN father is "y" @ 257|27320
SYN brother is "x" @ 257|27160
attach description: "syn_y -inhit 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 -inhit 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 -inhit close

Not To Be Reproduced 6-17 F15D
THE HIDDEN PORTION SUPPORTS SYNONYMING

Not To Be Reproduced
THE FOLLOWING iox ENTRY POINTS ARE GENERALLY OF USE TO USERS WRITING I/O MODULES:

ixo_$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

ixo_$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

ixo_$look_iocb

- RETURNS A POINTER TO THE IOCB FOR THE NAMED SWITCH IF IT EXISTS
IOX ENTRY POINTS USED IN I/O MODULES

- 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

Not To Be Reproduced 6-20 (End Of Topic)
TOPIC VII
Writing I/O Modules

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:

  1. 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)

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

  3. 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

  4. 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_

  5. 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:

  I  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_

  I  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))'

  I  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)

  I  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

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

  I  ALL I/O OPERATIONS MUST BE EXTERNAL ENTRY POINTS
IMPLEMENTATION RULES

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

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

  1. ESTABLISH AN any other HANDLER THAT CALLS terminate_process_IF MASKING IS IN EFFECT

  2. CALL hcs_set_ip_mask (0, mask)

  3. CHANGE THE IOCB

  4. CALL hcs_reset_ip_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
ENTRY POINTS OF AN I/O MODULE

MAJOR STEPS OF THE ATTACH OPERATION

1. SET iocb_ptr->iocb.open APPROPRIATELY
2. SET iocb_ptr->iocb.detach_iocb APPROPRIATELY
3. SET iocb_ptr->attach_descrip_ptr APPROPRIATELY
4. CALL iox_$propagate

MAJOR STEPS OF OPEN OPERATION

1. SET actual_iocb_ptr->iocb.<operation> FOR EVERY OPERATION THAT IS ALLOWED APPROPRIATELY
2. SET actual_iocb_ptr->open descrip_ptr APPROPRIATELY
3. CALL iox_$propagate
ENTRY POINTS OF AN I/O MODULE

• MAJOR STEPS OF CLOSE OPERATION

  I SET actual_iocb_ptr->iocb.open APPROPRIATELY

  I SET actual_iocb_ptr->iocb.detach_iocb APPROPRIATELY

  I SET actual_iocb_ptr->open_descrip_ptr TO NULL

  I CLEAN UP (SET BIT COUNTS, FREE STORAGE, ETC.)

  I CALL iox_$propagate

• MAJOR STEPS OF DETACH OPERATION

  I SET iocb_ptr->iocb.attach_descrip_ptr TO NULL

  I 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_descrp aligned,
    3 length fixed bin(17) init.(11),
    3 string char (11) init ("my_discard_"),
    2 open_descrp 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;
EXAMPLE OF AN I/O MODULE

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(optArray,1)<>0 then
    call error (error_table $wrong_no_of_args);
if iocb_ptr->iocb.attach descrpt_ptr = null() then
    call error (error_table $not_detached);
allocate block in (free_area);
iocb_ptr->iocb.attach descrpt_ptr,
iocb_ptr->iocb.attach_data_ptr = addr(attach descrpt);
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.
Call "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 descrpt_ptr = null();
call iox $propagate(iocb_ptr);
call hcs $reset_ips_mask(mask,mask);
return;
 EXAMPLE OF AN I/O MODULE

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;
EXAMPLE OF AN I/O MODULE

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_discardDetach;
    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, infpstr, 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 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

(End Of Topic)
# TOPIC VIII

## Interprocess Communication

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>8-1</td>
</tr>
<tr>
<td>IPC Terminology</td>
<td>8-4</td>
</tr>
<tr>
<td>IPC Protocol</td>
<td>8-8</td>
</tr>
<tr>
<td>Sending Wakeups</td>
<td>8-10</td>
</tr>
<tr>
<td>IPC Subroutines</td>
<td>8-11</td>
</tr>
<tr>
<td>ipc Error Codes</td>
<td>8-13</td>
</tr>
<tr>
<td>Creating and Destroying Event Channels</td>
<td>8-15</td>
</tr>
<tr>
<td>Invoking an Event-Call Procedure</td>
<td>8-17</td>
</tr>
<tr>
<td>Going Blocked on an Event Channel</td>
<td>8-18</td>
</tr>
<tr>
<td>Reading an Event-Wait Channel</td>
<td>8-20</td>
</tr>
<tr>
<td>Control Functions</td>
<td>8-23</td>
</tr>
<tr>
<td>Masking or Assigning Priority to Event Channels</td>
<td>8-27</td>
</tr>
<tr>
<td>An Example Using Event-Wait Channels</td>
<td>8-29</td>
</tr>
<tr>
<td>An Example Using Event-Call Channels</td>
<td>8-35</td>
</tr>
</tbody>
</table>
OVERVIEW

- The subsystem designer is often faced with requirements for sophisticated interprocess and intraprocess communications facilities.

- Subroutines exist which allow the designer to handle:

  1. **Synchronization of several cooperating processes**
     - `ipc` facilitates interprocess communication via "stop and go" signals between processes.

  2. **Protection of concurrently accessed data bases**
     - `set lock` allows cooperating processes to share a critical data base in a controlled manner.

  3. **Intraprocess timing**
     - `timer manager` allows a process to make use of several CPU or real-time timers.
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 DIRECT SYNCHRONIZATION, HOWEVER.

Multics SUPPORTS A FULL INTERPROCESS COMMUNICATION FACILITY WHICH ALLOWS FOR CONTROL 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.

- **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.
**IPC TERMINOLOGY**

<table>
<thead>
<tr>
<th>CHANNEL TYPE</th>
<th>OPTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>event-call</td>
<td>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.</td>
</tr>
<tr>
<td>event-wait</td>
<td>There are two options available:</td>
</tr>
<tr>
<td></td>
<td>1) The process can explicitly go blocked on the channel by calling <code>ipc_$block</code>. In this case, the statement following the call to <code>ipc_$block</code> will not be executed until/unless a wakeup is received on that channel. Notice that this effectively blocks the process itself.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

This polling technique is implemented via the `ipc_$read_ev_chn` entry point.
IPC PROTOCOL

- 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
IPC PROTOCOL

SENDING WAKEUPS

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

偏

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)

[code] = 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)

[code] = 2 IF AN INPUT ARGUMENT WAS INCORRECT, AND SIGNALLING WAS
  ABORTED

[code] = 3 IF THE TARGET PROCESS WAS NOT FOUND, AND SIGNALLING
  WAS ABORTED

[code] = error_table$_invalid_channel IF THE CHANNEL IDENTIFIER
  WAS NOT VALID
IPC SUBROUTINES

- 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
IPC SUBROUTINES

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 SUBROUTINES

IPC ERROR CODES

ALL CALLS TO ipc_ RETURN A NONSTANDARD STATUS CODE

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

**IPC ERROR CODES**

- `convert_ipc_code`

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

**MAPPING**

<table>
<thead>
<tr>
<th>ipcode</th>
<th>returned code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>error_table_$bad_ring_brackets</td>
</tr>
<tr>
<td>2-4</td>
<td>error_table_$no_message</td>
</tr>
<tr>
<td>5</td>
<td>error_table_$argerr</td>
</tr>
</tbody>
</table>
IPC SUBROUTINES
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.
### IPC SUBROUTINES

**CREATING AND DESTROYING EVENT CHANNELS**

```c
// 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
```
WHEN A PROCESS IS AWAKENED ON AN EVENT-CALL CHANNEL, CONTROL IS IMMEDIATELY PASSED TO THE "HANDLER" PROCEDURE PREVIOUSLY SPECIFIED IN THE CALL TO \texttt{ipc\_}$\texttt{decl\_ev\_call\_chn}$.

"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 \texttt{event\_call\_info\_ptr} PARAMETER:

\begin{verbatim}
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;
\end{verbatim}

RECALL: THE VALUE OF \texttt{data\_ptr} WAS SPECIFIED IN THE CALL TO \texttt{ipc\_}$\texttt{decl\_ev\_call\_chn}$ AND POINTS TO STRUCTURE OF THE USER'S CHOOSING.

SEE >\texttt{ldd>include>event\_call\_info.incl.pl1}
IPC SUBROUTINES
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)

  decl wait_list based aligned,
  2 nchan fixed bin,
  2 pad bit(36),
  2 channel_id (n refer wait_list.nchan) fixed bin(71);
IPC SUBROUTINES

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
IPC SUBROUTINES

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_$read_ev_chn (channel_id, ev_occurred,   
              addr (event_info), code);
  if code ^= 0 then call trouble;
  call ioa_ ("read_ev_chn says the event "[has not";has"] occurred, and the message is"[has not";has"] occurred.");
  call ipc_$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 "[has not";has"] occurred.");

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

Not To Be Reproduced 8-21 F15D
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.
IPC SUBROUTINES
CONTROL FUNCTIONS

• 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 SUBROUTINES
CONTROL FUNCTIONS

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_$create_ev_chn (channel_id, code);
  if code ≠ 0 then call trouble;
  call ncs_$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;
IPC SUBROUTINES
CONTROL FUNCTIONS

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
\connect. 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.
IPC SUBROUTINES

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.
IPC SUBROUTINES

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 ≠ 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
IPC SUBROUTINES

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_$initiate ("\udd\F15dw\Auerbach", "ipc_seg1", ",",
    0, 7, seg_ptr, code);
if seg_ptr = null () then call ERROR;
call ipc_$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;

Not To Be Reproduced 8-29
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";
IPC SUBROUTINES
AN EXAMPLE USING EVENT-WAIT CHANNELS

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_ ("^[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;
!print driver_ipc.pl1 1

driver_ipc: proc;

dcl ME char (10) init ("driver ipc") static options (constant);
dcl hcs_$initiate entry (char T*, char *, char (*),
fixed bin (1), fixed bin (2), ptr, fixed bin (35));
dcl code fixed bin (35);
dcl seg_ptr ptr;
dcl 1 ipc_info based (seg_ptr),
  2 his_process_id bit (36),
  2 his_chnl_id fixed bin (71);
dcl hcs_$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, T, 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;

Not To Be Reproduced 8-32
IPC SUBROUTINES

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

Not To Be Reproduced 8-33 F15D
IPC SUBROUTINES

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 9843986496772869469
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
IPC SUBROUTINES

AN EXAMPLE USING EVENT-CALL CHANNELS

!print listen.p11 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 (77),
 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", "ipo_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 (7, 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!!");
IPC SUBROUTINES
AN EXAMPLE USING EVENT-CALL CHANNELS

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", "ipcseg1", 
"", _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;
del 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

Not To Be Reproduced
IPC SUBROUTINES

AN EXAMPLE USING EVENT-CALL CHANNELS

!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

Not To Be Reproduced 8-38
IPC SUBROUTINES

AN EXAMPLE USING EVENT-CALL CHANNELS

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
IPC SUBROUTINES
AN EXAMPLE USING EVENT-CALL CHANNELS

YOU ARE NOW READY FOR WORKSHOP #4

Not To Be Reproduced  8-40
(End Of Topic)   F15D
## TOPIC IX

Interprocess Data Base Sharing

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>9-1</td>
</tr>
<tr>
<td>The Locking Mechanism.</td>
<td>9-2</td>
</tr>
<tr>
<td>The <code>set_lock</code> Subroutine</td>
<td>9-4</td>
</tr>
<tr>
<td>An Example of Locking.</td>
<td>9-6</td>
</tr>
</tbody>
</table>
• CONCEPT

Ⅰ ANOTHER FORM OF INTERPROCESS COMMUNICATION INVOLVES THE SHARING OF COMMON DATA BASE (NOT TO BE CONFUSED WITH MDBM) SEGMENTS

Ⅰ BECAUSE SEVERAL PROCESSES MAY BE ATTEMPTING TO CONCURRENTLY ACCESS AND UPDATE A SHARED DATA BASE, SOME FORM OF CONTROL IS REQUIRED:

Ⅰ TO PREVENT THE DATA BASE FROM BEING LEFT IN AN INCONSISTENT STATE

Ⅰ TO PREVENT ANY PROCESS FROM OPERATING UPON PARTIALLY-UPDATED DATA

• A POSSIBLE SOLUTION

Ⅰ A "LOCKING" FACILITY: THE set_lock_ SUBROUTINE

Ⅰ 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

Not To Be Reproduced 9-1 F15D
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.
THE SET LOCK SUBROUTINE

- 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
THE SET_LOCK SUBROUTINE

set_lock_unlock (AG93)

ATTEMPTS TO RESET A GIVEN LOCK WORD TO "0"

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
AN EXAMPLE OF LOCKING

!pwd
>user_dir_dir>F15dw>Auerbach
r 92:9 0.773

!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

!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;
AN EXAMPLE OF LOCKING

/* SOMETHING IS VERY WRONG - DIE */
call ioa ("FATAL ERROR!!");
call ioa ("NOTIFY DBA IMMEDIATELY!!");
end;
go to WRAPUP;
else;
end;

/* 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;

/* 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;

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

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
• CER TAIN 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**

• **W a t t e s** ALLOWS A PROCESS TO:

  1. BLOCK ITSELF FOR A SPECIFIED REAL TIME PERIOD (SLEEP)
  2. CALL A SPECIFIED PROCEDURE WHEN A SPECIFIED TIME INTERVAL HAS ELAPSED
  3. ISSUE A WAKEUP ON A SPECIFIED EVENT-WAIT CHANNEL WHEN A SPECIFIED TIME INTERVAL HAS ELAPSED

• THE SUBROUTINE INTERFACE IS `timer_manager_` (AK92)

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

  - `clock`
  - `cpu_time_and_paging`
  - `decode_clock_value`
  - `date_time`
  - `virtual_cpu_time`
  - `virtual_cpu_time_`
• 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 "alarm" (STATIC) CONDITION IS SIGNALLED

  • cpu

      "cpu" IS USED TO DESIGNATE A VIRTUAL CPU TIMER; WHEN A "cpu"
      TIMER GOES OFF, THE "cpu" (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_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; alarm 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_$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

```c
---SECONDS

call timer_manager_$sleep (30, "11"b);

RELATIVE---
```

WOULD CAUSE THIS PROCESS TO GO TO "SLEEP" FOR THIRTY SECONDS
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 ENTRY POINTS
USING CALL TIMERS

- 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

| 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 |
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
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);

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:

```plaintext
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

```c
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.
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.
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_adm_in_ IN APPENDIX B
TWO EXAMPLES USING TIMERS

!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 */

/* Establish 'on unit' for cleanup */
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;
TWO EXAMPLES USING TIMERS

/* 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;

Not To Be Reproduced
TWO EXAMPLES USING TIMERS

!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
TWO EXAMPLES USING TIMERS

!print usage.p11 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;

Not To Be Reproduced 10-19
TWO EXAMPLES USING TIMERS

```plaintext
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;

Usual counters initialized
Total page faults since process began 703
Total pd faults since process began 0
r 12:19 0.083 4

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
```
TWO EXAMPLES USING TIMERS

!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
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

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Process Stack Segment</td>
<td>11-1</td>
</tr>
<tr>
<td>The Stack Header</td>
<td>11-2</td>
</tr>
<tr>
<td>The Stack Frame</td>
<td>11-4</td>
</tr>
<tr>
<td>Argument List Format</td>
<td>11-7</td>
</tr>
<tr>
<td>Argument Descriptors</td>
<td>11-10</td>
</tr>
</tbody>
</table>
EXECUTION IN A STANDARD Multics PROCESS USES A STACK SEGMENT

- THERE IS ONE STACK SEGMENT PER RING WITH ENTRYNNAME '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)

![Stack Diagram]

*Frame size is variable, but not less than 32 words

Not To Be Reproduced 11-1 F15D
THE PROCESS STACK SEGMENT

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 */
THE PROCESS STACK SEGMENT

THE STACK HEADER

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, /* POINTS TO SYSTEM CONDITION TABLE (SCT) */
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
THE PROCESS STACK SEGMENT
THE STACK FRAME

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 O,
           THEN A DEDICATED
           REGISTER CONTAINS
           RETURN LOCATION */
  2 x(0 : 7) bit(18) unaligned, }
  2 a bit(36), }
  2 q bit(36), }
  2 e bit(36), }
  2 timer bit(27) unaligned,
  2 pad bit(6) unaligned,
  2 ring_alarm_reg bit(3) unaligned;

  /* AUTOMATIC VARIABLES */

  /* THREADDED 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 */

Not To Be Reproduced
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
### ARGUMENT LIST FORMAT

#### STANDARD ARGUMENT LIST

<table>
<thead>
<tr>
<th></th>
<th>ARGUMENT COUNT=n</th>
<th>CODE</th>
<th>DESCRIPTOR COUNT=n</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Pointer to Argument 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Pointer to Argument 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2^n</td>
<td>Pointer to Argument n</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Optional Pointer to Stack Frame of Containing Block</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pointer to Descriptor 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pointer to Descriptor 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pointer to Descriptor n</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### STANDARD DESCRIPTOR

- **FLAG**
- **PACKED**
- **NUMBER OF DIMENSIONS**

<table>
<thead>
<tr>
<th>TYPE</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOWER BOUND</td>
<td></td>
</tr>
<tr>
<td>UPPER BOUND</td>
<td></td>
</tr>
<tr>
<td>ELEMENT SEPARATION*</td>
<td></td>
</tr>
</tbody>
</table>

- *Present only if number of dimensions ≠ 0*
- *In bits if packed; in words otherwise*

---

**Not To Be Reproduced**

11-8

F15D
• 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.
ARGUMENT DESCRIPTORS

/* 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_flt_dec_9bit_ls_dtype init (9),
     real_flt_dec_9bit_dt 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_flt_dec_9bit_ls_overp Dtype init (29),
     real_flt_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 Dtype init (39), /* byte-aligned */
     real_flt_dec_4bit_bytealigned uns Dtype init (40), /* COBOL */
     real_flt_dec_4bit ls Dtype init (41), /* digit-aligned */
     real_flt_dec_4bit Dtype init (42), /* digit-aligned */
     real_flt_dec_4bit_bytealigned ls Dtype init (43),
     real_flt_dec_4bit_bytealigned Dtype init (44),
     cplx_flt_dec_4bit_bytealigned ls Dtype init (45),
     cplx_flt_dec_4bit_bytealigned Dtype init (46),
     ... */
ARGUMENT DESCRIPTORS

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.pli */
ARGUMENT DESCRIPTORS

ARGUMENT DESCRIPTORS

arg_list_ptr

0
1
2
3
4
5

arg_count = 1

code = 4
desc_count = 1

0

POINTER TO ARGUMENT 1

POINTER TO DESCRIPTOR 1

dcl 1 s,

2 A fixed bin (17,4),

2 B(5),

3 C,

3 D;

BASIC DESCRIPTOR OF S

BASIC DESCRIPTOR OF A

BASIC DESCRIPTOR OF B

LOWER BOUND OF B

UPPER BOUND OF B

ELEMENT SEPARATION OF B

BASIC DESCRIPTOR OF C

LOWER BOUND OF C

UPPER BOUND OF C

ELEMENT SEPARATION OF C

BASIC DESCRIPTOR OF D

LOWER BOUND OF D

UPPER BOUND OF D

ELEMENT SEPARATION OF D
**TOPIC XII**  
Special Programming Techniques

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calling a Procedure on the Fly</td>
<td>12-1</td>
</tr>
<tr>
<td>Building Data Segments</td>
<td>12-4</td>
</tr>
<tr>
<td>Creating an Error Table</td>
<td>12-9</td>
</tr>
</tbody>
</table>
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"?

```
void hcs_make_entry

void 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

```
void cu_generate_call

void 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)
CALLING A PROCEDURE ON THE FLY

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

- demo_cds.pl1 created by some text editor
- add_name demo_cds.pl1 = .cds
- demo_cds.pl1
- demo_cds.cds
- cds demo_cds.cds
- demo_cds
- data_seg

- This executable object program is created when cds calls the PL/1 compiler
- Creates
- Created when cds command procedure automatically calls demo_cds
BUILDING DATA SEGMENTS

! 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 def_ptr_link bit (1) unal, /* says put def's in linkage */
         3 separate_static bit (1) unal, /* separate static section */
         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.sections (2).len = 0;
  cds_arg_ptr -> cds_args.sections (2).struct_name = "";
  cds_arg_ptr -> cds_args.sections (2).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;
BUILDING DATA SEGMENTS

!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
rw       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

<table>
<thead>
<tr>
<th>Object</th>
<th>Text</th>
<th>Defs</th>
<th>Link</th>
<th>Symb</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>52</td>
<td>62</td>
</tr>
<tr>
<td>Length</td>
<td>224</td>
<td>6</td>
<td>44</td>
<td>10</td>
<td>126</td>
</tr>
</tbody>
</table>

6 Definitions:

  segname:    data_seg

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

No Links.

Not To Be Reproduced
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 PARENTHESES)

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 PARENTHESES
CREATING AN ERROR TABLE

• ! 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.pl

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.pl

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

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Standard Process Environment</td>
<td>13-1</td>
</tr>
<tr>
<td>Process Creation in the Supervisor Ring.</td>
<td>13-3</td>
</tr>
<tr>
<td>Process Initialization in the User Ring.</td>
<td>13-5</td>
</tr>
<tr>
<td>Modifying the Process Environment.</td>
<td>13-9</td>
</tr>
<tr>
<td>Project Administration</td>
<td>13-11</td>
</tr>
<tr>
<td>Closed Subsystems</td>
<td>13-13</td>
</tr>
<tr>
<td>Limited Subsystems</td>
<td>13-14</td>
</tr>
<tr>
<td>New Subsystems</td>
<td>13-18</td>
</tr>
</tbody>
</table>
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

THIS IS THE EVENT-CALL HANDLER THAT WAS SET UP AS SUCH BY sue_Save_listener

- PRINT LOGIN BANNER
- VERIFIES PERSON_ID AND PASSWORD
- REFERENCES THE PROJECT DEFINITION TABLE (pdt)
- VERIFIES PROJECT_ID
- INITIALIZES SOME ACCOUNTING DATA BASES
- PROCESS DIRECTORY CREATED AND SCREECH-NAMED
- PROCESS INITIALIZATION TABLE (pitt) CREATED/INITIALIZED
- ACTIVE PROCESS TABLE ENTRY (pitt) CREATED/INITIALIZED
- KNOWN SEGMENT TABLE (est) CREATED/INITIALIZED
- PROCESS DATA SEGMENT (pdt) CREATED/INITIALIZED
- 32-BIT PROCESS ID DEVELOPED

RING CHANGE

stack_4 CREATED CURRENT VALIDATION LEVEL ADJUSTED

IF DAEMON
- daemon_real_init_admin_

IF ABSENTEE
- absentee_real_init_admin_

IF INTERACTIVE
- MAKE ATTACHMENTS OF 4 STANDARD SWITCHES
- DETERMINE init_proc FROM pdt
- ESTABLISH STATIC HANDLERS FOR term, cpu, and alarm CONDITIONS

RETURN

CALL 'init_proc' PROCEDURE (NORMALLY process_overseer_) PASSING addr(pdt) IFF SYSTEM-SUPPLIED OVERSEER BEING CALLED

IF A start_up.exe IS TO BE EXECUTED, COMMAND LINE IS
- as_start_up.exe login interactive
OTHERWISE, COMMAND LINE IS NULL

- SET UP DEFAULT HANDLER (null) FOR THE any_other CONDITION
- PRINT MOTO IF start_up.exe NOT TO BE EXECUTED AND LOGIN NOT-active
- DEVELOP A COMMAND LINE AND PASS IT TO listen_Bisten_

- ESTABLISHES LEVEL 1
- INVOKES CURRENT READY PROCEDURE BY CALLING cu_freezy_proc
- CALLS os_Scp, WHICH PASSES COMMAND LINE TO CURRENT COMMAND PROCESSOR
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
THE STANDARD PROCESS ENVIRONMENT
PROCESS CREATION IN THE SUPERVISOR RING

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
THE STANDARD PROCESS ENVIRONMENT

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
THE STANDARD PROCESS ENVIRONMENT

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

Not To Be Reproduced 13-6 F15D
THE STANDARD PROCESS ENVIRONMENT
PROCESS INITIALIZATION IN THE USER RING

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
THE STANDARD PROCESS ENVIRONMENT

PROCESS INITIALIZATION IN THE USER RING

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'
MODIFYING THE PROCESS ENVIRONMENT

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
MODIFYING THE PROCESS ENVIRONMENT

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: 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;
MODIFYING THE PROCESS ENVIRONMENT

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

Not To Be Reproduced 13-11 F15D
MODIFYING THE PROCESS ENVIRONMENT

PROJECT ADMINISTRATION

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

<table>
<thead>
<tr>
<th>homedir</th>
<th>ABSOLUTE PATHNAME OF USER'S HOME DIRECTORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>initproc</td>
<td>NAME OF THE USER'S PROCESS OVERSEER PROCEDURE</td>
</tr>
</tbody>
</table>

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 FORTHAN" 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 QUNTS 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
MODIFYING THE PROCESS ENVIRONMENT
LIMITED SUBSYSTEMS

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 ENTRYPNAME 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):
(delete name dn):
(list ls):
(logout):
(print pr):
(program interrupt pi):
(rename rn):
(start sr):
(edit:

> udd>MED>nd>list;
> sss>qedx;
MODIFYING THE PROCESS ENVIRONMENT

LIMITED SUBSYSTEMS

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

ls -first 1
Segments = 72, Lengths = 80.

who
who is not a legal command
QUIT

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

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)
IMPLEMENTATION OF THE DIAL FACILITY

dial_manager_ 

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$_registered_server WAS ORIGINALLY USED TO ALLOW DIALS
SET_UP.Dial:  proc;  
dcl ipc_$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;
DIALING TERMINALS TO A PROCESS
AN EXAMPLE

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;
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

1. ESTABLISH EVENT-WAIT CHANNEL

2. REQUEST ANSWERING SERVICE TO DIAL A SPECIFIED PHONE NUMBER

3. ONE CAN CHANGE EVENT-WAIT TO EVENT-CALL CHANNEL AT THIS TIME

4. 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:

1. 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

2. 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**

<table>
<thead>
<tr>
<th>What Is It?</th>
<th>15-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
<td>15-2</td>
</tr>
<tr>
<td>The Message Segment</td>
<td>15-3</td>
</tr>
<tr>
<td>Layered Design</td>
<td>15-6</td>
</tr>
<tr>
<td>Primitive Message Segment Facility</td>
<td>15-8</td>
</tr>
<tr>
<td>Extended Access</td>
<td>15-9</td>
</tr>
<tr>
<td>message_segment Subroutine Summary</td>
<td>15-10</td>
</tr>
<tr>
<td>Message Segment Facility Illustrative Example</td>
<td>15-12</td>
</tr>
</tbody>
</table>
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

\[ \begin{array}{l}
\text{I HEADER} \\
\text{I LOCK WORD} \\
\text{I 36-BIT MESSAGE SEGMENT ID BIT PATTERN} \\
\text{I OFFSET TO FIRST MESSAGE} \\
\text{I OFFSET TO LAST MESSAGE} \\
\text{I MESSAGE COUNT} \\
\text{I SWITCHES} \\
\text{I MSEG INCONSISTENT} \\
\text{I MSEG HAS BEEN SALVAGED} \\
\text{I ALLOCATION BIT STRING SAYS WHICH BLOCKS ARE USED} \\
\text{I LENGTH (ALLOCATION BIT STRING)} \\
\text{I MESSAGE BLOCK SIZE} \\
\text{I UNUSED BLOCK COUNT} \\
\text{I ...AND OTHER INFO} \\
\end{array} \]
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
• 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
THE PRIMITIVE MESSAGE FACILITY IS COMPRISED OF MODULES WHICH

- CREATE AND DELETE MSEGS
- MANIPULATE EXTENDED ACCESS
- LOCK AND UNLOCK MSEGS
- MANIPULATE 'OWN' MESSAGES
- SALVAGE MSEGS
- CONVERT MSEGS FROM A PREVIOUS FORMAT
• BOTH QUEUE AND MAILBOX MSEGs EMPLOY THESE ATTRIBUTES:

  I a
  ALLOWS USER TO ADD A MESSAGE

  I d
  ALLOWS USER TO DELETE ANY MESSAGE

  I r
  ALLOWS USER TO READ ANY MESSAGE

  I o
  ALLOWS USER TO READ/DELETE 'OWN' MESSAGES

  I s
  ALLOWS USER TO DETERMINE WHETHER MSEG HAS BEEN SALVAGED AND MESSAGE COUNT

• IN ADDITION, MAILBOX MESSAGE SEGMENTS EMPLOY:

  I 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

Not To Be Reproduced 15-11 F15D
MESSAGE SEGMENT FACILITY ILLUSTRATIVE EXAMPLE

trans: proc; /* {4, 4, 4} */
... 
trans_add: proc;
... 
end;
trans_delete: proc;
... 
end;
trans_read: proc;
... 
end;
trans_count: proc;
... 
end;
trans_summary: proc;
... 
end trans;

message_segment: proc; /* {1, 1, 5} */
open: entry ... ;
close: entry ... ;
get_message_count: entry ... ;
add_index: entry ... ;
delete_index: entry ... ;
read_index: entry ... ;
incremental_read_index: entry ... ;
own_read_index: entry ... ;
own_incremental_read_index: entry ... ;
end message_segment;

> udd>F15d>trans.ms
{1, 1, 1}

COMMAND PROCEDURE

CALLS

MANIPULATES

THE MESSAGE SEGMENT

MESSAGE SEGMENT FACILITY ILLUSTRATIVE EXAMPLE

Not To Be Reproduced 15-12 (End Of Topic)
**TOPIC XVI**

**Program Library Management**

<table>
<thead>
<tr>
<th>Introduction</th>
<th>16-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization of Program Libraries</td>
<td>16-2</td>
</tr>
<tr>
<td>Naming Conventions</td>
<td>16-4</td>
</tr>
<tr>
<td>A Typical Program Library</td>
<td>16-6</td>
</tr>
<tr>
<td>Program Library Management Tools</td>
<td>16-9</td>
</tr>
<tr>
<td>Installation Tools</td>
<td>16-10</td>
</tr>
<tr>
<td>Library Descriptor Tools</td>
<td>16-20</td>
</tr>
</tbody>
</table>
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
INTRODUCTION

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
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

THE OBJECT PROGRAM SUBDIRECTORY IS GENERALLY GIVEN THE NAMES

object
  o

THE LISTINGS SUBDIRECTORY IS GENERALLY GIVEN THE NAMES

lists
  l

THE INCLUDE FILE SUBDIRECTORY IS GENERALLY GIVEN THE NAMES

include
  incl

THE HELP FILES SUBDIRECTORY IS GENERALLY GIVEN THE NAMES

info
  INFO
INTRODUCTION
NAMING CONVENTIONS

NAMING CONVENTIONS FOR BOUND SEGMENTS AND CORRESPONDING ARCHIVES

[1] THE BOUND SEGMENT ITSELF IS GIVEN AN ENTRYNAMES "bound ???????", WHERE ??????? IS A NAME CHOSEN BY THE DESIGNER (E.G., bound_command_loop_)

[2] 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)

[3] THE ARCHIVE WHICH CONTAINS THE OBJECT PROGRAMS AND WHICH WAS INPUT TO THE BINDER IS NAMED "bound_????????archive" (E.G., bound_command_loop_.archive)
INTRODUCTION
A TYPICAL PROGRAM LIBRARY

!list >udd>F15dw>Auerbach>user_library_1 -all
Directories = 1.
sma user_library_1
   ulT

!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
sma object
   o
   O
sma source
   s
   S

!cwd >udd>F15dw>Auerbach>ul1
INTRODUCTION

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

Not To Be Reproduced

16-7

F15D
INTRODUCTION

A TYPICAL PROGRAM LIBRARY

!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
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
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.
PROGRAM LIBRARY MANAGEMENT TOOLS

INSTALLATION TOOLS

- 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

Not To Be Reproduced 16-13 F15D
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.
Program Library Management Tools

Installation Tools

!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

Not To Be Reproduced 16-15
!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.*.*
rew * SysDaemon.*
Names:
test

!list -first 2

Segments = 75, Lengths = 126.

r w 64 example.io
r w 04/06/81.audit

!us list

!list -first 2

Segments = 76, Lengths = 127.

r w 1 example.il
r w 64 example.io
PROGRAM LIBRARY MANAGEMENT TOOLS

INSTALLATION TOOLS

!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.*.*
rew *.SysDaemon.*
Names:
  test

Not To Be Reproduced 16-17 F15D
!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.i1
   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 1BBBJKzgHHgZMDK
   r w 1 1BBBJKzgHHFjDLd

!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)
  03/14781 -1121.1 mst Sat   PL/I

>user dir dir>MED>NDibble>junk>test: (segment 2)
  03/14781 -1121.1 mst Sat   PL/I

The 2 segments match.

!us list

!dp example.il
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)
program library management tools

library descriptor tools

- 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
PROGRAM LIBRARY MANAGEMENT TOOLS

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

Not To Be Reproduced 16-22 F15D
• **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

THE Multics STORAGE HIERARCHY

ASCII DESCRIPTOR SOURCE SEGMENT .x.ld IS TRANSLATED BY
library_descriptor_compiler

WHICH PRODUCES
ALM SOURCE SEGMENT .x.nlm

WHICH IS ASSEMBLED BY
alm ASSEMBLER

WHICH PRODUCES
LIBRARY DESCRIPTOR .x

COULD ALSO REFERENCE
ANOTHER LIBRARY DESCRIPTOR

USES

VARIABLE SEARCH PROCEDURES

COULD ALSO REFERENCE
SYSTEM-PROVIDED LIBRARY DESCRIPTOR
multics_libraries

SPECIFIES

library_fetch COMMAND

CALLS fetch:

library_info COMMAND

CALLS info:

library_map COMMAND

CALLS map:

library_print COMMAND

CALLS print:

library_cleanup COMMAND

CALLS cleanup:
!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
PROGRAM LIBRARY MANAGEMENT TOOLS

LIBRARY DESCRIPTOR TOOLS

!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

!li gw.archive

1 gw.archive     path:  >udd>F15D>s1>handout
    contents modified:  02/18/81   1648.4

!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

Not To Be Reproduced 16-26
(End Of Topic) F15D
APPENDIX A

AIM

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Commands and Subroutines</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-1</td>
<td>A-3</td>
</tr>
</tbody>
</table>
CONCEPTS

• 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 > 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

Not To Be Reproduced
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
COMMANDS AND SUBROUTINES

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 ENTRYNNAME, 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_init_admin</td>
<td>B-1</td>
</tr>
<tr>
<td>user_real_init_admin</td>
<td>B-3</td>
</tr>
<tr>
<td>process_overseer</td>
<td>B-6</td>
</tr>
<tr>
<td>project_start_up</td>
<td>B-10</td>
</tr>
<tr>
<td>listen</td>
<td>B-15</td>
</tr>
</tbody>
</table>
**user init admin**

```

******************************************************************************
*
*
* 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:    <user_real_init_admin_>
                    [user_real_init_admin_]
            eppab
            tra
                    join

daemon_init_admin:  <daemon_real_init_admin_>
                    [daemon_real_init_initadmin_]
            eppab
            tra
                    join

absentee_init_admin:  <absentee_real_init_admin_>
                    [absentee_real_init_admin_]
            eppab
            tra
                    join

join:
push           po_ptr                           prepare argument list
            eppbp  arg+2
            spribp arg+4
            eppbp  pit_ptr
            spribp  arg+1
            fld   =2b24,d1
            staq arg

call              real_init_admin_ (po_ptr, pit_ptr)
call ab:0(arg)
call the process overseer ... always pass it the pit pointer
```
user_init_admin

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  process_killing_pointer
  "  get a pointer to -2!0 which will blow us away
  arg  0
  even
process_killing_pointer:
  its  -2,0

end
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,
io$attach entry (char (*), char (*), char (*), char (*),
bit (72) aligned),
find_command_quick_no_message entry (ptr, fixed bin, ptr,
fixed bin (35)),
io$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 (sus_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;
user real init_admin

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
    procedure: ^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;
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 caught otherwise uncaught conditions. The
* supplied handler, default_error_handler_$wall prints any message provided
* for the error condition, establishes a condition wall, and calls the
* listener to get a new listener level. A condition wall is just another
* any_other handler; this intercepts conditions that might otherwise be
* caught by other handlers present on the stack.
*
* setting up a static handler for the mme2 condition. The mme2 condition
* is raised when the mme2 instruction is executed. It is used by debug
* to establish breakdots. The handler transfers control to debug when
* the condition is signalled.
*
* allowing the ".." escape to command query. This is enabled by calling
* command_query_$set_cp_escape 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. */
process overseer

/* Automatic */

dcl initial_command_line char (104) var init ("");
dcl pit_ptr_ptr;
dcl code fixed bin (35);
dcl unaligned_homeder char (64) unaligned based 
  (addr (pit_ptr -> pit.homeder));
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 (odeptr, 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 */
process overseer

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*/
process_ overseer

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_;
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_sioa_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");
project start up

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 (7), 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 = n>user_dir_dir> rtrm (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);
project start up

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$_cp (adr (ps_ec_cl), lenglh (rtrim (ps_ec_cl)), (0));
revert command_error;
call cu$_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 " , 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);
project start up

else do;
call hcs statutory 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 statutory 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 "=" " \n" " \n
first ec arg;
initial command line = initial command line "=" " \n" " \n
second ec arg;
end;

no_start_up:
call hcs statutory terminate_noname (pit ptr, code);
call sct manager statutory get (mme2 sct index, mme2 handler, (0));
if mme2 handler = saved mme2 handler
then call sct manager statutory set (mme2 sct index, codeptr
(process overseer statutory mme2 fault handler_), code);
end;
call listen (initial command line);
do while ("T"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 7(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_at_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_;
listen

#include <stdio.h>

int main()
{
    printf("Hello, world!
    ");
    return 0;
}

Not To Be Reproduced
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 bet 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$grow_stack_frame entry (fixed bin (21), ptr, fixed bin (35)),
cu$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;
listen

/* 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 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;
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 = "$50s5;"
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;

Not To Be Reproduced
listen

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;
listen

/* *******************************************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,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****************************************** */
listen

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.stackptr;
        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
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, 1 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Old form</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.1000</td>
<td>FNP a, LSLA 0, subchannel 0</td>
<td>tty000</td>
</tr>
<tr>
<td>a.h108</td>
<td>FNP a, HSLA 1, subchannel 8</td>
<td>tty708</td>
</tr>
<tr>
<td>b.h016</td>
<td>FNP b, HSLA 0, subchannel 16</td>
<td>ttyG16</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b.h016.01</td>
<td>FNP b, HSLA 0, subchannel 16, concentrator subchannel 1</td>
</tr>
<tr>
<td>b.h016.09</td>
<td>same physical channel, concentrator subchannel 9</td>
</tr>
</tbody>
</table>

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.pl` the call to `get_userid`. For this call to successfully return, it is required that your instructor obtain read access to `>sc1>answer_table`.

```plaintext
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^/\n");
  do i = 1 to index;
    if name (i) = "" then
      call ioa ("^a.^a      ^a" name (i), proj (i), time (i));
    end;
  end;
end ipc_report;
```
ipc_driver: proc;

dcl send_mail_entry (char (*), char (*), ptr, fixed bin (35)),
get_wdir_entry returns (char (168)),
host_make_seg_entry (char (*), char (*), char (*), fixed bin (5),
ptr, fixed bin (35)),
host_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;

# 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);

Not To Be Reproduced  D-3  F15D
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 'F' 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_mail_.
    occurred while trying to complain about full table in 'ipc_status'.");
        return;
    end;
end;
return;

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 hos ($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 entry returns (char (168));
dcl 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
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_iode_.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
APPENDIX G

Advanced Dial Facility Example
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_$decl_ev_call_chn entry (fixed bin (71), entry,
ptr, fixed bin, fixed bin (35)),
hcs$sassign_channel entry (fixed bin (71), fixed bin (35)),
dial_manager$allow_dials entry (ptr, fixed bin (35)),
(ioa_, com_err$, ioa$sioa_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 (72) 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$sassign_channel (wasted_channel, code);
end;
/* Channel must be obtained for notifying this process of all hangups and dialups. */
call ipc_file:svg create ev chnl (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_file:svg $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_file:svg $decl ev call cpn (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_file:svg "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 $user_output, 
    "[DIALED UP]; HUNG UP" AT "a.",
    which_channel, status_flags.dialed_up, time);
/* Restart any interrupted io to the master terminal */
call iox$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 (ioa$user_output, 
            "[terminal]; terminals" AT "a.",
            dialed.no_dialed, (dialed.no_dialed = 1),
            (dialed.no_dialed = 1));
/* Close and detach the switch */
call iox$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
end; */
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 */
      called.devname (i) = which_channel;
      dialed.no_dialed = dialed.no_dialed + 1;
      call ioa $ioa_switch (iox $user_output,
      "At this instant, ^i ^"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 $attach ptr (dialed.sw (i).iocb_ptr,
      "tty "^i$devname, null ()), code);
      if code $= 0 then call ERROR (9);

      call iox $open (dialed.sw (i).iocb_ptr,
      3 /* stream io */,
      "0", /* 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;

ERROR: proc (error_number);
    /* Internal proc to report errors */
dcl error_number;
call com_err (code, ME, "Check call ^1 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;

FINITE:
   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_desc_ptr */

dcl 1 attach_description unaligned based
    (iocb_ptr -> iocb.attach_desc_ptr),
    2 descr_length fixed bin (35),
    2 desc char (0
    refer (attach_description.descr_length)) unal;

    call ioa $ioa_switch (iocb_ptr,
        "Dialed to 'astra' on channel "a.",
        after (attach_description.descr, "\""));
    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 */
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 entry name in the pathname to be a star name. To first determine whether the star name is valid (i.e., 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 "set_temp_segments_" and "release_temp_segments_" routines, without having to create one when needed.

   a. temp sess belongs to a single procedure when in use
   b. the maximum size of a temp ses is 16k
   c. the above mentioned free pool is system wide
   d. if all temp sess 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 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 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 "set_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 safe 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_rings()
   d. cross_rings
   e. none of the above

3. Many subsystems honor an "e" request (edx, calc, probe etc.) What entry point 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 rings 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 (e.g. 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 rings brackets of {3, 3, 4}

a. a segment can be in one rings only
b. the user can only read the segment subject of course to ACL and AIM
c. the user's process cannot execute in rings 3
d. the user can execute the segment with a rings 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_rings" io_module which allows cross ring attachments of switches:

a. must be given the outer rings switchname as an argument

b. is used to attach the inner rings switch to a previously existing outer rings switch

c. enables use of the "cross_rings_io_allow_cross" subroutine to do cross rings io

d. the inner rings 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 be 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 now 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 occurrence 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
Quiz 3

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_$cutoff`
b. `create_ipc_mask_
c. `ipc_$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 strings of 36 binary 1's
b. bit strings 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_iris_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 rings 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

<table>
<thead>
<tr>
<th>Workshop</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workshop One</td>
<td>W-1</td>
</tr>
<tr>
<td>Workshop Two</td>
<td>W-3</td>
</tr>
<tr>
<td>Workshop Three</td>
<td>W-6</td>
</tr>
<tr>
<td>Workshop Four</td>
<td>W-7</td>
</tr>
<tr>
<td>Workshop Five</td>
<td>W-9</td>
</tr>
<tr>
<td>Workshop Six</td>
<td>W-12</td>
</tr>
<tr>
<td>Workshop Seven</td>
<td>W-13</td>
</tr>
</tbody>
</table>
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 segment 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;
WORKSHOP TWO

Changing the Command Environment

The command environment can be shaped in several ways by using alternate 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;
```
WORKSHOP TWO

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 $nul, 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 ('l"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.
An Interprocess Communication Workshop

The interprocess communication facility is used to coordinate processing between separate processes. In this workshop, you will design and implement a procedure which will communicate with the instructor's process as follows:

Write a PL/I 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>Fl5D>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 use 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 PL/I '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>Fl5D>s1>include>w4.incl.pl1.

Not To Be Reproduced
dcl (sca_, 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.pl that implements the following fictitious command:

```
USAGE: lock [-min minutes]

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. 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.

Not To Be Reproduced W-10 F15D
WORKSHOP FIVE

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 $arg;

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);
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).
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_deso
COMPILATION LISTING OF SEGMENT pfgu

Compiled by: Multios PL/I Compiler, Release 25c, of February 18, 1980
Compiled at: Honeywell LISD Phoenix, System M
Compiled on: 03/19/80 1043.5 1st Wed
Options: map

1 find: proo (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
dol aos_names_array (4) char (32) internal static options (constant)
6 init ("soc_sec_no.aos", "manager.aos", "mail_station.aos", "salary.aos");
7
dol access_allowed(N) bit (1);
8
dol access_allowed override bit(4) unaligned defined (access_allowed);
9
dol dir_name char (168);
10 dol user_id char (32);
11 dol ring fixed bin (17);
12 dol mode fixed bin (5);
13
dol 1 bit_mode_struc aligned based (mode_ptr),
14
dol 1 bit_mode_struc aligned based (mode_ptr),
15
dol 2 pad_bit (30) unaligned,
16
dol 2 two_bits bit(2) unaligned, /* don't care */
17
dol 2 read_bit (1) unaligned, /* On if read effective access */
18
dol 2 fourth_bit (1) unaligned, /* don't care */
19
dol 2 write_bit (1) unaligned; /* On if write effective access */
20 dol ME char (4);
21 dol find bit (1);
22 dol 1 emp_struc,
23
dol 2 soc_sec_no pic "(9)9",
24
dol 2 manager_pic "(9)9",
25
dol 2 mail_station char (3),
26
dol 2 salary fixed dec (8, 2);
27 dol (code, dummy_code) fixed bin (35);
28 dol (first_name, last_name) char (15);
29
do stub(null, addr, size, rtrim) builtln;
30 dol (error_table $moderr ext static fixed bin (35));
31 dol (error_table $moderr ext static fixed bin (35));
32 dol 1 fixed bin;
33 dol (iocbp, mode_ptr) ptr init (null ());
34
do (ioobp, mode_ptr) ptr init (null ());
35
do (ioobp, mode_ptr) ptr init (null ());
36
do (ioobp, mode_ptr) ptr init (null ());

1 dol
2 dol $attach_ptr entry (ptr, char(*), ptr, fixed bin (35)),
3 dol $attach_name entry (char(*), ptr, char(*), ptr, fixed bin (35)),
4 dol $close entry (ptr, fixed bin (35)),
5 dol $control entry (ptr, char(*), ptr, fixed bin(35)),
6 dol $delete_record entry (ptr, fixed bin(35)),
7 dol $detach_iob entry (ptr, fixed bin(35)),

8   lox$_find_loob entry (char(*), ptr, fixed bin(35)),
9   lox$_get_chars entry (ptr, ptr, fixed bin(21), fixed bin(21), fixed bin(35)),
10  lox$_get_line entry (ptr, ptr, fixed bin(21), fixed bin (21), fixed bin (35)),
11  lox$_modes entry (ptr, char(*), char(*), fixed bin (35)),
12  lox$_move_attach entry (ptr, ptr, fixed bin(35)),
13  lox$_open entry (ptr, ptr, fixed bin(21) aligned, fixed bin(35)),
14  lox$_position entry (ptr, fixed bin, fixed bin(21), fixed bin (35)),
15  lox$_put_chars entry (ptr, ptr, fixed bin(35)),
16  lox$_read_key entry (ptr, char(256) varying, fixed bin(21), fixed bin (35)),
17  lox$_read_length entry (ptr, fixed bin(21), fixed bin(35)),
18  lox$_read_record entry (ptr, ptr, fixed bin(21), fixed bin(21), fixed bin (35)),
19  lox$_rewind_record entry (ptr, ptr, fixed bin (21), fixed bin(35)),
20  lox$_seek_key entry (ptr, char(256) varying, fixed bin(21), fixed bin(35)),
21  lox$_write_record entry (ptr, ptr, fixed bin(21), fixed bin (35)),
22  lox$_destroy_loob entry (ptr, fixed bin(35));
37
39  dol $ cu$ level_get entry (fixed bin);
40  dol $ cu$ level_set entry (fixed bin);
41  dol get_rings_entry () returns (fixed bin (3));
42  dol hos$_get_user_effmode entry (char (*), char (*), char (*), fixed bin, fixed bin (5), fixed bin (35));
43  dol get_group_id$_tag_star entry () returns (char (32));
44
45       ME = "find";
46  find = "$b; 
47  got0 COMMON_CODE;
48
49  put: entry (last_name, first_name, emp_struct, code);
50      ME = "put";
51  find = "$b;
52
53 COMMON_CODE:
54
55  call cu$ level_get (qld_level);
56  ring = get_rings ();
57  call cu$ level_set (ring);
58
59  code = 0;
60
61  dir_name = "\\uxd9f15d>\$t"
62  user_id = get_group_id$_tag_star ()
63  call hos$_get_user_effmode (dir_name, "emp", user_id, ring, mode, code);
64  if code $= 0 then goto return_point;
65  mode_ptr = addr (mode);
66  if (*find & "(bit_mode_struct.read & bit_mode_struct.write)) | /* if put, we must have read and write on emp */
67       (find & "bit_mode_struct.read then do; */ If find, we must have read on emp */
68  code = error_table$_moderr;
69  goto return_point;
70
71 /* If we get here, then it must be the case that we have status on
72 * our current working directory. */
73 /* Obtain the effective access on the access control segments of interest */
do i = 1 to 4; /* Once for each access control seg */
call hcs $get_user_effmode (dir_name, aos_names_array (1), user_id, ring, mode, code);
if find then access_allowed (i) = bit_mode_struc_read;
else access_allowed (i) = bit_mode_struc_write;
end;

/\ ONE OF TWO BLOCKS OF CODE, DEPENDING UPON EN'TRYPOINT TAKEN... */
if find then do;
if access_allowed_overlay then do; /* elegant ORing */
call lox $find_loob ("emp_sw", loobp, code);
call lox $attach_ptr (loobp, "vfile " || rtrim (dir_name) || ">emp", null (), code);
call lox $open (loobp, 11 /* direct input */, "O", code);
call lox $seek_key (loobp, (last_name || first_name), 4 /* size (emp_struc), code */;
if code = 0 then do;
call lox $read_record (loobp, addr (emp_struc), 4 /* size (emp_struc), unused_rec_len, code */;
   if access_allowed (1) then soo_sec_no = 0;
   if access_allowed (2) then manager = 0;
   if access_allowed (3) then mail_station = "";
   if access_allowed (4) then salary = 0;
end;
else do;
call lox $close (loobp, dummy_code);
call lox $detach_loob (loobp, dummy_code);
goto return_point; /* To reset validation level */
end;
else code = 1; /* This special code indicates that no fields were accessible */
end;
else do;
if access_allowed (1) & access_allowed (2) & access_allowed (3) & access_allowed (4) then do; /* If all of
*others are write-able */
call lox $find_loob ("emp_sw", loobp, code);
call lox $attach_ptr (loobp, "vfile " || rtrim (dir_name) || ">emp -extend", null (), code);
call lox $open (loobp, 12 /* direct output */, "D", code);
call lox $seek_key (loobp, (last_name || first_name), 4 /* size (emp_struc), code */;
if code = error_table $no_record then call lox $write_record (loobp, addr (emp_struc), 4*size (emp_struc), code);
call lox $close (loobp, dummy_code);
call lox $detach_loob (loobp, dummy_code);
goto return_point;
end;
else code = 1; /* This special code indicates that some fields were inaccessible */
end;
return point:
call cu$_level_set$(old_level);
end find;
<table>
<thead>
<tr>
<th>LINE</th>
<th>NUMBER</th>
<th>DATE MODIFIED</th>
<th>NAME</th>
<th>PATHNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>07/19/79</td>
<td>pfgu.pl1</td>
<td>/user_dir_dir/F15D&gt;Student_01&gt;pfgu.pl1</td>
</tr>
<tr>
<td>37</td>
<td>1</td>
<td>02/13/79</td>
<td>dol_iox_entries.inol.pl1</td>
<td>/user_dir_dir/F15D&gt;Student_01&gt;dol_iox_entries.inol.pl1</td>
</tr>
</tbody>
</table>
### NAMES DECLARED IN THIS COMPILATION.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>access_allowed</td>
<td>000165</td>
<td>automatic</td>
<td>char(4)</td>
<td>unaligned dol 20 set ref 45 50* array unaligned dol 8 set ref 81 82 89 89 96 97 98 99 112 112 112 112 undleared dol 9 set ref 89 initial array unaligned dol 6 set ref 80* dol 29 ref 65 95 95 117 117 level 1 dol 14 dol 27 set ref 1 49 50 63 64 69 80 90 91 92* 93* 98 95 107 113 114 115 116 117 117 128* external dol 39 ref 53 external dol 40 ref 57 129 unaligned dol 10 set ref 61 63 80 91 91 dol 27 set ref 102* 103* 118* 119* level 1 unaligned dol 22 set ref 1 49 93 95 95 116 117 117 117 dol 30 ref 69 dol 31 ref 117 unaligned dol 21 set ref 46 51* 66 66 81 87 unaligned dol 28 set ref 1 49 93 116 external dol 43 ref 62 external dol 41 ref 56 external dol 42 ref 63 80 80 81 82* dol 32 set ref 78* 80 81 82* initial dol 33 set ref 31 90 90 91 92* 93* 95* 102* 103* 113* 114* 115* 116* 117* 118* 119* external dol 1-1 ref 91 114 external dol 1-1 ref 102 118 external dol 1-1 ref 103 119 external dol 1-1 ref 92 115 external dol 1-1 ref 95 external dol 1-1 ref 93 116 external dol 1-1 ref 117 unaligned dol 28 set ref 69 93 95 116 level 2 packed unaligned dol 22 set ref 98* level 2 packed unaligned dol 22 set ref 97* dol 13 set ref 63* 65 80* initial dol 33 set ref 33* 65* 66 66 66 81 82 dol 29 ref 33 33 91 91 114 114 dol 35 set ref 53* 129* level 2 packed unaligned dol 14 ref 66 66 81 dol 12 set ref 56* 57* 63* 80* dol 29 ref 91 114 level 2 dol 22 set ref 99* dol 29 ref 93 93 95 95 116 level 2 packed unaligned dol 22 set ref 96* dol 34 set ref 95* unaligned dol 11 set ref 62* 63* 80* level 2 packed unaligned dol 14 ref 66 82</td>
</tr>
</tbody>
</table>
NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.

<table>
<thead>
<tr>
<th>Name</th>
<th>Start</th>
<th>Length</th>
<th>Object</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>lox_attach_name</td>
<td>000000</td>
<td></td>
<td>0</td>
<td>1-1</td>
</tr>
<tr>
<td>lox_control</td>
<td>000000</td>
<td></td>
<td>0</td>
<td>1-1</td>
</tr>
<tr>
<td>lox_delete_record</td>
<td>000000</td>
<td></td>
<td>0</td>
<td>1-1</td>
</tr>
<tr>
<td>lox_destroy_locb</td>
<td>000000</td>
<td></td>
<td>0</td>
<td>1-1</td>
</tr>
<tr>
<td>lox_get_chars</td>
<td>000000</td>
<td></td>
<td>0</td>
<td>1-1</td>
</tr>
<tr>
<td>lox_get_line</td>
<td>000000</td>
<td></td>
<td>0</td>
<td>1-1</td>
</tr>
<tr>
<td>lox_smode</td>
<td>000000</td>
<td></td>
<td>0</td>
<td>1-1</td>
</tr>
<tr>
<td>lox_move_attach</td>
<td>000000</td>
<td></td>
<td>0</td>
<td>1-1</td>
</tr>
<tr>
<td>lox_position</td>
<td>000000</td>
<td></td>
<td>0</td>
<td>1-1</td>
</tr>
<tr>
<td>lox_put_chars</td>
<td>000000</td>
<td></td>
<td>0</td>
<td>1-1</td>
</tr>
<tr>
<td>lox_read_key</td>
<td>000000</td>
<td></td>
<td>0</td>
<td>1-1</td>
</tr>
<tr>
<td>lox_read_length</td>
<td>000000</td>
<td></td>
<td>0</td>
<td>1-1</td>
</tr>
<tr>
<td>lox_rewrite_record</td>
<td>000000</td>
<td></td>
<td>0</td>
<td>1-1</td>
</tr>
</tbody>
</table>

NAMES DECLARED BY EXPLICIT CONTEXT.

<table>
<thead>
<tr>
<th>Name</th>
<th>Start</th>
<th>Length</th>
<th>Object</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMON_CODE</td>
<td>000145</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>find</td>
<td>000117</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>put</td>
<td>000134</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>return_point</td>
<td>001200</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

THERE WERE NO NAMES DECLARED BY CONTEXT OR IMPLICATION.

STORAGE REQUIREMENTS FOR THIS PROGRAM.

<table>
<thead>
<tr>
<th>Name</th>
<th>Start</th>
<th>Length</th>
<th>Object</th>
<th>Text</th>
<th>Link</th>
<th>Symbol</th>
<th>Defs</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>find</td>
<td>1700</td>
<td>1214</td>
<td>0</td>
<td>0</td>
<td>1414</td>
<td>1462</td>
<td>1214</td>
<td>1424</td>
</tr>
</tbody>
</table>

BLOCK NAME          STACK SIZE TYPE               WHY NONQUICK/WHO SHARES STACK FRAME
find                233 external procedure is an external procedure.

STORAGE FOR AUTOMATIC VARIABLES.

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC IDENTIFIER</th>
<th>BLOCK NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>find</td>
<td></td>
<td>find</td>
</tr>
<tr>
<td>000100</td>
<td>access_allowed</td>
<td>find</td>
</tr>
<tr>
<td>000153</td>
<td>dir_name</td>
<td>find</td>
</tr>
<tr>
<td>000153</td>
<td>user_id</td>
<td>find</td>
</tr>
<tr>
<td>000163</td>
<td>ring</td>
<td>find</td>
</tr>
<tr>
<td>000164</td>
<td>mode</td>
<td>find</td>
</tr>
<tr>
<td>000165</td>
<td>ME</td>
<td>find</td>
</tr>
<tr>
<td>000166</td>
<td>find</td>
<td>find</td>
</tr>
<tr>
<td>000167</td>
<td>dummy_code</td>
<td>find</td>
</tr>
<tr>
<td>000170</td>
<td>1</td>
<td>find</td>
</tr>
<tr>
<td>000172</td>
<td>locl'bp</td>
<td>find</td>
</tr>
<tr>
<td>000174</td>
<td>mode_ptr</td>
<td>find</td>
</tr>
<tr>
<td>000176</td>
<td>unused_rec_len</td>
<td>find</td>
</tr>
<tr>
<td>000177</td>
<td>old_level</td>
<td>find</td>
</tr>
</tbody>
</table>

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC IDENTIFIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>alloc_os</td>
<td></td>
</tr>
<tr>
<td>ext_entry</td>
<td></td>
</tr>
<tr>
<td>ext realloc_os</td>
<td></td>
</tr>
<tr>
<td>call_ext_out_des</td>
<td></td>
</tr>
<tr>
<td>call_ext_out</td>
<td></td>
</tr>
<tr>
<td>return</td>
<td></td>
</tr>
<tr>
<td>shorten_stack</td>
<td></td>
</tr>
</tbody>
</table>
THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

cu $level_get
he5 $get_user_effmode
iox $find_iocb
iox $write_record
get $level_set
iox $attach_ptr
iox $open
iox $close
iox $read_record
get_group_id $tag Star
get $ring
iox $detach_iocb
iox $seek_key

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  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  |
|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 33   | 000105 | 45   | 000125 | 46   | 000127 | 47   | 000131 | 49   | 000132 | 50   | 000142 |
| 51   | 000144 | 53   | 000145 | 56   | 000154 | 57   | 000165 | 59   | 000174 | 70   | 000306 | 78   | 000307 |
| 63   | 000210 | 66   | 000254 | 66   | 000256 | 69   | 000303 | 89   | 000373 | 90   | 000376 |
| 81   | 000214 | 82   | 000362 | 83   | 000367 | 87   | 000371 | 96   | 000617 | 97   | 000630 |
| 91   | 000314 | 92   | 000507 | 93   | 000532 | 94   | 000570 | 95   | 000574 | 107  | 000704 | 109  | 000707 |
| 98   | 000641 | 99   | 000651 | 102  | 000661 | 103  | 000672 | 104  | 000703 | 117  | 001124 | 118  | 001152 |
| 112  | 000710 | 113  | 000732 | 114  | 000756 | 115  | 001043 | 116  | 001066 | 120  | 001174 |
| .119 | 001163 | 124  | 001175 | 129  | 001200 | 131  | 001207 |
/* Command interface to the put_find_gate_gate_entrypoints */

find: proc;

/* DECLARATIONS */

dcl ME char (4) varying;
dcl nargs fixed bin;
dcl argl fixed bin (21);
dcl (first name, last name) char (15);
dcl code fixed bin (35);
dcl arg char (argl) based (argp);
dcl argp ptr;
dcl 1 emp_struc,
  2 soc_sec_no pic "(9)9",
  2 manager pic "(9)9",
  2 mail_station char (3),
  2 salary fixed dec (8, 2);
dcl (error_table $bigarg, error_table $wrong_no_of_args) ext static fixed bin (35);
dcl (fixed overflow, conversion, size) condition;
dcl cu_%arg_ptr entry (fixed bin, ptr, fixed bin (21), fixed bin (35));
dcl cu_%arg_count entry (fixed bin);
dcl (com_err, ioa) entry options (variable);
dcl (put_find_gate $find, put_find_gate $put) entry
  (char (15), char (15), 1 structure, 2 pic "(9)9" member, 2 pic "(9)9" member,
  2 char (3) member, 2 fixed dec (8, 2) member, fixed bin (35));
dcl cv_deo_check_entry (char(*), fixed bin(35)) returns (fixed bin(35));

 ME = "find";
call cu_%arg_count (nargs);
  if nargs = 2 then do;
    call com_err_ (error_table $wrong_no_of_args, ME,
      "Incorrect invocation: find First_name Last_name");
    return;
  end;
call cu_%arg_ptr (1, argp, argl, code);
  if argl > 15 then do;
    call com_err_ (error_table $bigarg, ME, "a exceeds 15 characters.", arg);
    return;
  end;
else first_name = arg;
call ou $arg_ptr (2, argp, arg1, code);
if arg1 > 15 then do;
call com_err_ (error_table_$bigarg, ME, "a exceeds 15 characters.", arg);
return;
end;
else last_name = arg;
51 /* Initialize emp_struc prior to calling the gate entrypoint */
52 soc_sec_no, manager, salary = 0;
mail_station = ";
56 call put_find gate $find (last_name, first_name, emp_struc, code);
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.");
else if code = 0 then call com_err_ (code, ME);
60 else call loca (""["s"]SOCIAL SECURITY NUMBER = "1
61 "]][["s"],MANAGER'S SOCIAL SECURITY NUMBER = "1
62 ][["s"]MAIL STATION = "a
63 ][["s"]SALARY = ";1
64 "]", (soc_sec_no = 0), soc_sec_no,
(manager = 0), manager,
(mail_station = ";"), mail_station,
(salary = 0), salary);
68 return; /* END OF find PROCEDURE */
69
70 put: entry ();
71 ME = "put";
72 call ou $arg_count (nargs);
if nargs = 0 then do;
call com_err_ (error_table_$wrong_no_of_args, ME,
"Correct invocation: put First_name Last_name Employee's_ssn No Manager's_ssn No Mail_Station Salary")
76
77 return;
78 end;
79 /* OBTAIN SIX REQUIRED ARGUMENTS */
80 call ou $arg_ptr (1, argp, arg1, code);
83 if arg1 > 15 then do;
call com_err_ (error_table_$bigarg, ME, "a", arg);
85 return;
86 end;
87 else first_name = arg;
88 call ou $arg_ptr (2, argp, arg1, code);
89 if arg1 > 15 then do;
call com_err_ (error_table_$bigarg, ME, "a", arg);
91 return;
92 end;
94 else last_name = arg;
95 call ou $arg_ptr (3, argp, arg1, code);
97 if arg1 < 9 then verify (arg, "0123456789") = 0 then do;
call com_err_ (0, ME, "a is not a suitable social security number for "a.",
arg, last_name);
return;
end;
else soc_sec_no = ov_dec_check_(arg, code);
call cu_ $arg_ptr ($, argp, argl, code);  
if argl" 9 then do;
        call com_err_ (0, ME, "a is not a suitable social security number for "a's manager.", arg, last_name);
        return;
end;
else manager = ov_dec_check_(arg, code);
call cu_ $arg_ptr (5, argp, argl, code);  
if argl" 3 then do;
        call com.err_ (0, ME, "a is not a suitable 3-character mail drop.", arg);
        return;
end;
else mail_station = arg;
call cu_ $arg_ptr (6, argp, argl, code);
on fixedoverflow, size begin;
call com.err_ (0, ME, "a is not a proper salary. It must resemble 999999.99", arg);
goto bottom;  
end;
salary = ov_dec_check_ (arg, code);
if code " 0 then do;
        call com.err_ (0, ME, "a is not a proper salary. It must resemble 999999.99", arg);
goto bottom;  
end;
call put_find gate $put (last_name, first_name, emp_struc, code);  
if code " 1 then call com.err_ (0, ME, "You lack access on one or more of the fields in the database.");
else if code " 0 then call com.err_ (code, ME);
else call lioa_ ("Record written.");
bottom:
return;
end find;
## SOURCE FILES USED IN THIS COMPILATION.

<table>
<thead>
<tr>
<th>LINE</th>
<th>NUMBER</th>
<th>DATE MODIFIED</th>
<th>NAME</th>
<th>PATHNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>07/19/79</td>
<td>1109.5</td>
<td>put.pl1</td>
<td>&gt;udd&gt;F15D&gt;Student_01&gt;put.pl1</td>
</tr>
</tbody>
</table>
### NAMES DECLARED IN THIS-compilation.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ME</td>
<td>000100</td>
<td>automatic</td>
<td>varying char(4)</td>
</tr>
<tr>
<td>arg</td>
<td></td>
<td>based</td>
<td>char</td>
</tr>
<tr>
<td>argl</td>
<td>000103</td>
<td>automatic</td>
<td>fixed bin(21,0)</td>
</tr>
<tr>
<td>argp</td>
<td>000116</td>
<td>automatic</td>
<td>pointer</td>
</tr>
<tr>
<td>code</td>
<td>000114</td>
<td>automatic</td>
<td>fixed bin(35,0)</td>
</tr>
<tr>
<td>com_err</td>
<td>000020</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>cu_arg_count</td>
<td>000016</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>cu_arg_ptr</td>
<td>000014</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>cv__dec_check</td>
<td>000030</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>em__ptr</td>
<td>000120</td>
<td>stack reference</td>
<td></td>
</tr>
<tr>
<td>error_table__bigarg</td>
<td>000010</td>
<td>external static</td>
<td>fixed bin(35,0)</td>
</tr>
<tr>
<td>error_table__wrong_no_of_args</td>
<td>000012</td>
<td>external static</td>
<td>fixed bin(35,0)</td>
</tr>
<tr>
<td>first_name</td>
<td>000104</td>
<td>automatic</td>
<td>char(15)</td>
</tr>
<tr>
<td>fixedoverflow</td>
<td>000132</td>
<td>stack reference</td>
<td>condition</td>
</tr>
<tr>
<td>ioe</td>
<td>000022</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>last_name</td>
<td>000110</td>
<td>automatic</td>
<td>char(15)</td>
</tr>
<tr>
<td>mail_station</td>
<td>4(18)</td>
<td>automatic</td>
<td>char(3)</td>
</tr>
<tr>
<td>manager</td>
<td>2(09)</td>
<td>automatic</td>
<td>picture(9)</td>
</tr>
<tr>
<td>narga</td>
<td>000102</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td>put__find__gate__$find</td>
<td>000024</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>put__find__gate__$put</td>
<td>000026</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>salary</td>
<td>6</td>
<td>automatic</td>
<td>fixed dec(8,2)</td>
</tr>
<tr>
<td>size</td>
<td>000140</td>
<td>stack reference</td>
<td>condition</td>
</tr>
<tr>
<td>soo__seq_no</td>
<td>000120</td>
<td>automatic</td>
<td>picture(9)</td>
</tr>
</tbody>
</table>

### NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.

<table>
<thead>
<tr>
<th>Offset</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES (* indicates a set context)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dol 8</td>
<td>set ref 31* 34* 40* 46* 57* 59* 72* 75* 84* 91* 98* 106* 113* 121* 126* 131* 132*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 10</td>
<td>set ref 38* 39 40 43 44* 45 46 49 82* 83 84 87 89* 90 91 94 97 98 99 102 104* 105 106 109 113 114 124* 126*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 14</td>
<td>set ref 38* 40 43 44* 45 46 49 82* 84 87 89* 91 94 96* 97 98 99 102 104* 105 106 109 111* 112 113 115 116 118* 121 124 126*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 12</td>
<td>set ref 38* 44* 56* 57 59 59* 82* 89* 91 94 96* 102* 104* 109* 111* 118* 124* 130* 131 132*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 25</td>
<td>ref 34 40 46 57 59 75 84 91 98 106 113 121 126 131 132*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 23</td>
<td>ref 38 44* 82 89 96 104 111 118*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 29</td>
<td>ref 102 109 124*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 20</td>
<td>set ref 40* 46* 49* 84* 91*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 20</td>
<td>set ref 34* 75*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 11</td>
<td>set ref 43* 56* 87* 130*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 12</td>
<td>ref 120*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 25</td>
<td>ref 60 133*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 25</td>
<td>ref 60 133*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 15</td>
<td>set ref 49* 56* 94* 98* 106* 130*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 15</td>
<td>set ref 54* 60 60* 116*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 21</td>
<td>ref 120*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 21</td>
<td>ref 120*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 15</td>
<td>set ref 53* 60 60* 109*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 9</td>
<td>set ref 32* 33 73* 74*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 26</td>
<td>ref 56*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 26</td>
<td>ref 130*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 15</td>
<td>set ref 53* 60 60* 124*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dol 15</td>
<td>set ref 53* 60 60* 102*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### NAME DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.

- conversion 000000 stack reference condition

### NAMES DECLARED BY EXPLICIT CONTEXT.

- bottom 002025 constant label
- find 000315 constant entry
- put 000753 constant entry

### NAME DECLARED BY CONTEXT OR IMPLICATION.

- verify builtin function

- ref 97 105
STORAGE REQUIREMENTS FOR THIS PROGRAM.

<table>
<thead>
<tr>
<th>Object</th>
<th>Text</th>
<th>Link</th>
<th>Symbol</th>
<th>Defs</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
<td>2364</td>
<td>2416</td>
<td>2237</td>
</tr>
<tr>
<td>Length</td>
<td>2610</td>
<td>2237</td>
<td>32</td>
<td>155</td>
<td>125</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BLOCK NAME</th>
<th>STACK SIZE</th>
<th>TYPE</th>
<th>WHY NONQUICK/WHO SHARES STACK FRAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>find</td>
<td>226</td>
<td>external procedure</td>
<td>is an external procedure.</td>
</tr>
<tr>
<td>on unit</td>
<td>on line 120</td>
<td>98 on unit</td>
<td></td>
</tr>
</tbody>
</table>

STORAGE FOR AUTOMATIC VARIABLES.

<table>
<thead>
<tr>
<th>STACK FRAME</th>
<th>LOC IDENTIFIER</th>
<th>BLOCK NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>find</td>
<td>000100</td>
<td>ME</td>
</tr>
<tr>
<td></td>
<td>000102</td>
<td>nargs</td>
</tr>
<tr>
<td></td>
<td>000103</td>
<td>arg1</td>
</tr>
<tr>
<td></td>
<td>000104</td>
<td>first_name</td>
</tr>
<tr>
<td></td>
<td>000110</td>
<td>last_name</td>
</tr>
<tr>
<td></td>
<td>000114</td>
<td>code</td>
</tr>
<tr>
<td></td>
<td>000116</td>
<td>argp</td>
</tr>
<tr>
<td></td>
<td>000120</td>
<td>emp_struc</td>
</tr>
</tbody>
</table>

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.

<table>
<thead>
<tr>
<th>r_e_as</th>
<th>r_ne_as</th>
<th>call_ext_out</th>
<th>call_ext_out</th>
<th>return</th>
<th>tra_ext</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>ext_entry</td>
<td>int_entry</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

<table>
<thead>
<tr>
<th>com_err</th>
<th>ou $arg_count</th>
<th>ou $arg_ptr</th>
<th>cv_deo_check</th>
</tr>
</thead>
<tbody>
<tr>
<td>iou</td>
<td>put_find_gate_get_find</td>
<td>put_find_gate_get_put</td>
<td></td>
</tr>
</tbody>
</table>

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

<table>
<thead>
<tr>
<th>error_table_$bigarg</th>
<th>error_table_$wrong_no_of_args</th>
</tr>
</thead>
</table>

LINE LOC 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 45 000441 44 000445 45 000461 46 000467
47 000521 49 000521 53 000526 54 000584 56 000584 57 000561 59 000612
60 000632 68 000751 70 000752 72 000760 73 000764 74 000773 75 000776
77 01022 82 01023 83 01042 84 01045 85 01076 87 01077 89 01103
90 01122 91 01125 92 01156 98 01157 96 01163 97 01202 98 01224
100 01263 102 01264 104 01315 105 01334 106 01356 107 01415 109 01416
111 01447 112 01466 113 01471 118 01524 116 01525 118 01531 120 01550
121 01568 122 01620 120 01623 124 01630 125 01661 126 01663 127 01717
130 01720 131 01735 132 01766 133 02006 135 02025
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
```

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 file procedure "put_find_gate_util_.

include gate_macros

BEGIN INCLUDE FILE ...... gate_macros.incl.alm
Last modified 6/77 by N. MorFis, B. Greenberg, & T. VanVleck
Modified 741212 by PG to inhibit while computing virtual CPU time
This file contains several macros used when generating gate segments.
The major macros that are defined are:

gate_info general setup code for normal gates
hardcore_gate_info general setup code for hardcore gates
bad_dir_handler setup and handler for bad dir_condition (goes at bottom)
bad_dir_handler_entry entrypoint for above (goes at top)
gate to define a normal gate
hgate to define a hardcore gate
fgate to define a fast hardcore gate

maclist off
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$entry.
1-30  *
1-31  *
1-32  macro  hgate
1-33       gentry  &1,&&2,&&1.t
1-34       tax2  .setup
1-35       &&&5,bad_dir_trap$[ tax0  .set_dir_trap
1-36       ]  .short_call &2&3(ap$0)
1-37       eppbp  lp&1.t
1-38       tra  .return_code
1-39       maclist  restore
1-40       use  linkage
1-41       even
1-42       maclist  on,save
1-43       &1.t:  bas  ,6
1-44       bas  .6
1-45       maclist  restore
1-46       &end
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  * macro  fgate
1-55       gentry  &1,0,0
1-56       epllp  .my lp,*
1-57       tra  &2&3
1-58       maclist  off
1-59       &end
1-60     &end
1-61
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  * macro  gate
1-69       gentry  &1,&&2,0
1-70       tax2  .setup
1-71       &&&0,short_call &2&3(ap$0)
1-72       return
1-73       maclist  restore
1-74     &end
1-75

handout  _  put_fnd_gate  "at
1-80: macro, gentry
1-81: maclist, on, save
2-82: segdef, &1
3-83: maclist, restore
4-84: use, transfer_vector
5-85: maclist, on, save
6-86: &1:
7-87: tralist
8-88: use, main
9-89: maclist, restore
10-90: zero, &2, &3
11-91: &1.e:
12-92: &1.e:
13-93: &end
1-94
1-95
1-96 "HARDCORE_GATE_INFO - general info for hardcore gates
1-97
1-98 macro hardcore_gate_info
1-99 maclist on,save
1-100 name &1
1-101 include stack_header
1-102 include stack_frame
1-103 maclist restore
eject
tempd .temp
1-109 tempd .label_variable(0)
1-110 tempd .time1, time2
1-111 tempd .unwinder_arglist(0)
1-112 tempd .on_unit(5)
1-113 tempd .pf, entryp
1-114 tempd .vfl_arglist(2)
1-115 tempd .temp:
1-116 tempd .temp
1-117 use transfer_vector
equ .tv_begin,#
1-119 tra .actor
1-120 tra .actor
1-121 use tv_end
1-122 use tv_end
1-123 use tv_end
1-124 use tv_end
1-125 use tv_end
1-126 .tv_end: vfd 14/".tv_begin")
1-127 use main
1-128 use main
even
1-129 even
1-130 even
1-131 even
1-132 .my lp: bss ,2
1-133 use linkage
1-134 use linkage
1-135 use linkage
1-136 use linkage
1-137 use linkage
1-138 use linkage
1-139 use linkage
1-140 use linkage
1-141 maclist restore
eject
1-142 epl: .my lp,#
1-143 .actor: eplp .my lp,#
1-144 maclist restore
gate_actor
1-145 maclist restore
1-146 gate_actor
1-147 gate_actor
1-148
I
51 1-149 .setup: push
52 1-150 eplp .my lp,#
53 1-151 srplp sp#stack_frame.lp_ptr
54 1-152 maolist restore
gcheck
55 1-153 inhibit on <+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+><+<
MACRO  BAD_DIR_HANDLER - code to setup and handle bad_dir condition

  MACRO  bad_dir_handler
  .maclist  on,save
  include  on_unit
  use  transfer_vector
  .handler_entry:
  .transfer
  .handler
  .handler_restart_entry:
  .transfer
  .handler_restart_point
  use  main

  .set_dir_trap:
  .entryp  save  for  restart
  stx0
  mlr  
  .desc9a  0,0
  .desc9a  .on_unit,10x4
  .eppbp  .bad_dir_name
  .prbp  .on_unit+on_unit.name
  .eppbp  .handler_entry
  .prbp  .on_unit+on_unit.body
  .lxl  .bad_dir_deSc
  .sx1l  .on_unit+on_unit.size
  .eaa  .on_unit
  .on_unit  set  up  on-unit  for  bad_dir_
  .sb1a  .sp[0,du]
  .sta  .sp[0,du]
  .lxl  .stack_frame.on_unit.rel_ptr
  .lxl  .stack_frame.condition_bit,31
  .lxl  .stack_frame.flag_word
  .tr  0,0
  .tr  0,0
  .string  bad_dir
.handler: epaq
  splo
  verify that call came from ring 0
  1-236
  epaq
  splo
  verify that call came from ring 0
  1-236
  epaq
  splo
  verify that call came from ring 0
  1-236

cmp
  handle epaq apl0 verify that call came from ring 0
  1-236
  epaq
  check ring number in AL
  1-237
cmp
  handle epaq apl0 verify that call came from ring 0
  1-237
  epaq
  check ring number in AL
  1-237

tze
  go way kid you bother me
  1-238
tze
  go way kid you bother me
  1-238

push
  " ok, we like the call
  1-239
push
  " ok, we like the call
  1-239

epplp
  .my lp,#
  get display
  1-240
epplp
  .my lp,#
  get display
  1-240

lda
  bp|stack_frame.prev_sp
  1-241
lda
  bp|stack_frame.prev_sp
  1-241

r1a
  =07000000,d1
  from another ring?
  1-242
r1a
  =07000000,d1
  from another ring?
  1-242

tze
  .continue_signal if not, back to signal_
  1-243
tze
  .continue_signal if not, back to signal_
  1-243

epapp
  .ap12,#
  Get mptr
  1-244
epapp
  .ap12,#
  Get mptr
  1-244

spripb
  bp|mptr
  .. save in gate frame
  1-245
spripb
  bp|mptr
  .. save in gate frame
  1-245

epppp
  .label_variable+2
  1-246
epppp
  .label_variable+2
  1-246

spripb
  .label_variable
  1-247
spripb
  .label_variable
  1-247

epppp
  .label_variable
  1-248
epppp
  .label_variable
  1-248

fld
  =1b24,d1
  1-249
fld
  =1b24,d1
  1-249

stp
  .unwinder_arglist
  1-250
stp
  .unwinder_arglist
  1-250

call
  unwinder_unwinder_(.unwinder_arglist)
  1-251
call
  unwinder_unwinder_(.unwinder_arglist)
  1-251

.epncontinue_signal:
  1-252
eppap
  .ap12,#
  Get mptr
  1-253
eppap
  .ap12,#
  Get mptr
  1-253

lda
  =400000,du
  "!"b
  1-254
lda
  =400000,du
  "!"b
  1-254

sta
  ap|10,#
  set continue bit
  1-255
sta
  ap|10,#
  set continue bit
  1-255

return
  1-256
return
  1-256

.epnhandler_restart_point:
  1-257
epaq
  splo
  check that call came from ring 0
  1-258
epaq
  splo
  check that call came from ring 0
  1-258

cmp
  handle epaq apl0 verify that call came from ring 0
  1-259
cmp
  handle epaq apl0 verify that call came from ring 0
  1-259

tze
  go way kid you bother me
  1-260
tze
  go way kid you bother me
  1-260

two
  .continue_signal: if not, back to signal_
  1-261
two
  .continue_signal: if not, back to signal_
  1-261

epplp
  .my lp,#
  get display
  1-262
epplp
  .my lp,#
  get display
  1-262

lda
  =400000,du
  "!"b
  1-263
lda
  =400000,du
  "!"b
  1-263

sta
  ap|10,#
  set continue bit
  1-264
sta
  ap|10,#
  set continue bit
  1-264

return
  1-265
return
  1-265

.epnhandler_restart_point:
  1-266
epaq
  splo
  check that call came from ring 0
  1-267
epaq
  splo
  check that call came from ring 0
  1-267

cmp
  handle epaq apl0 verify that call came from ring 0
  1-268
cmp
  handle epaq apl0 verify that call came from ring 0
  1-268

tze
  go way kid you bother me
  1-269
tze
  go way kid you bother me
  1-269

two
  .continue_signal: if not, back to signal_
  1-270
two
  .continue_signal: if not, back to signal_
  1-270

epplp
  .my lp,#
  get display
  1-271
epplp
  .my lp,#
  get display
  1-271

lda
  .stack_frame.condition_bit+1,d1 Vanish on-unit
  1-272
lda
  .stack_frame.condition_bit+1,d1 Vanish on-unit
  1-272

r1a
  .ap|stack_frame.flag_word
  1-273
r1a
  .ap|stack_frame.flag_word
  1-273

epppp
  .moptr
  1-274
epppp
  .moptr
  1-274

spripb
  .vfl_arglist+2
  1-275
spripb
  .vfl_arglist+2
  1-275

fld
  =1b24,d1
  1-276
fld
  =1b24,d1
  1-276

staq
  .vfl_arglist
  1-277
staq
  .vfl_arglist
  1-277

short_call verify_lock verify_lock_bad_dir(.vfl_arglist)
  1-278
short_call verify_lock verify_lock_bad_dir(.vfl_arglist)
  1-278

lda
  .entryp
  1-279
lda
  .entryp
  1-279

epapp
  .ap|stack_frame.arg_ptr,#
  1-280
epapp
  .ap|stack_frame.arg_ptr,#
  1-280

tra
  0,0
  retry the call
  1-281
tra
  0,0
  retry the call
  1-281

maclist restore
  1-282
maclist restore
  1-282
eject
  1-283
eject
  1-283

maclist restore
  1-284
maclist restore
  1-284

&end
  1-285
&end
  1-285

1-286
1-286
1-287
1-287
GATE_INFO - general info for non-hardcore gates

macro gate_info
  macro list on, save
  use transfer_vector
  tra .actor_
  use main
  join /text/transfer_vector, main
  macro list restore
  eject

.actor: getlp
  macro list restore
  gate_actor

.actor: getlp
  macro list restore
  goheck
  tra 0,2

.macolist restore
  macro list restore
  eject

.setup: push
  getlp
  macro list restore

.macolist restore
  goheck
  tra 0,2

.macolist restore
  macro list restore
  eject

.macolist restore
  macro list restore

end
" Macro to generate gate actor.

  macro gate_actor
  maolist on,save
  eppbp sp|2,*
  lda bpl-1
  tze .return_name
  lda 1,dl
  stz sp|4,*
  tsx0 .search_defs
  cmpo (pr,rl),(pr,rl) compare name
  deso9a bpl-1(3),al
  deso9a bb0,al
  tnz .next_def
  lda ab11,2
  arl 18
  sta sp|4,*

short_return

  .return_name:
  1x13 sp|4,*
  tsx0 .search_defs
  cmpx3 ab11,2
  tnz .next_def
  lda bb10
  tnz .next_def
  lda bb|0
  sta bpl-1
  mlr (pr,rl),(pr,rl) return string
  deso9a bb|0(1),al
  deso9a bpl0,al

short_return

.search_defs:
  eax2 0
  eppab lp0,*
  .defs_loop:
  1x11 ab|1,2
  cmpx1 =0400000,du
  tnz .next_def
  1dx7 ab|2,2
  eppbb ab|0,7
  tra 0,0
  .next_def:
  1dx2 ab|0,2
  tnz .defs_loop
short_return

52  1-375
53  1-376
54  1-377 &end
1-378
1-379
"Miscellaneous macros.

1-380 1-383 macro gocheck maolist on,save
1-384 1-386 ldx1 -2,2 get number of args expected
1-385 1-387 tze .no_gate_error if zero, none or doesn't matter
1-388 1-389 cmpx1 sp0 compare against number given
1-390 1-392 tze .no_gate_error args match, call procedure
1-393 1-395 call signal $signal_(signal_arglist)
1-396 1-398 oct 0
1-399 1-401 even
1-402 1-404 signal_arglist:
1-405 1-407 zero 2,4
1-408 1-409 zero 2,0
1-410 1-411 arg .gate_errordesc
1-412 1-413 arg .gate_errorname
1-414 1-415 arg .gate_error
do
1-416 1-417 maolst on,save
3 1-418 &end
1-419 1-420 " END INCLUDE FILE ...... gate_macros.incl.slm
1-421 6

gate_info
use transfer_vector
tra .actor
use main
join /text/transfer_vector.main
000004  aa  7 00046 2721 20  .actor:  getlp  eppbp  api2,  get length of string
000005  aa  0 00002 3521 20  lda  bpi-1,  zero length => get name
000006  aa  2 77777 2351 00  tze  .return_name  include length of acc
000007  0a  000023 6000 00  
000010  aa  000001 0350 07  ad1  1,dl  
000011  aa  0 00004 4501 20  stz  api4,  
000012  0a  000036 7000 00  tso  .search_defs  
000013  aa  0 00140 1065 40  cmpco  (pr,rl),(pr,rl)  compare name
000014  aa  277777 600005  des09a  bpi-1(3),al  
000015  aa  300000 000005  des09a  bb0,al  
000016  0a  000046 6010 00  tnz  .next_def  
000017  aa  1 00001 2351 12  lda  ab1,2  return location
000020  aa  000022 7710 00  arl  18  
000021  aa  0 00004 7551 20  sta  api4,  
000022  aa  7 00044 7101 20  short_return  
000023  0a  0 00004 7231 20  
000024  0a  000036 7000 00  
000025  aa  1 00001 1031 12  cmpx3  ab1,2  compare location
000026  0a  000046 6010 00  tnz  .next_def  
000027  aa  3 00000 2351 00  lda  bb0  get length of name
000030  aa  000033 7710 00  arl  27  set length of varying string
000031  aa  2 77777 7551 00  sta  bb-1  return string
000032  aa  0 00140 1005 40  mfr  (pr,rl),(pr,rl)  
000033  aa  300000 200005  des09a  bb0(1),al  
000034  aa  200000 000005  des09a  bb0,al  
000035  aa  7 00044 7101 20  short_return  
000036  0a  0 00004 6220 00  
000037  aa  4 00000 3515 20  eax2  0  
000038  0a  00004 3515 20  eppab  lp0,  ab -> defs
000040  aa  1 00001 7211 12  .defs_loop:  lxl1  ab1,2  get class and flags
000041  aa  400000 1010 03  omph1  =400000,du  must be class 0
000042  0a  000046 6010 00  tnz  .next_def  
000043  aa  1 00002 2271 12  ldx7  ab1,2  
000044  aa  1 00000 3535 17  eppbb  ab0,7  bb -> name
000045  aa  000000 7100 10  tra  0,0  test definition
000046  aa  1 00000 2221 12  .next_def:  ldx2  ab0,2  chain to next def
000047  0a  000040 6010 00  tnz  .defs_loop  
000050  aa  7 00044 7101 20  short_return
.setup: push
getlp
ldx1
-t2,2
get number of args expected
if zero, none or doesn't matter
tze .no_gate_error
compare against number given
tzep .no_gate_error
args match, call procedure
call signal_$signal_(signal_arglist)

000010 signal_arglist:

zero 2,4
tzep .no_gate_error
zero 2,0

arg .gate_errormame
arg .gate_errordesc
arg

string .gate_errormame:

.aol "gate_error"

.vfd o9/525,027/10

.trn 0,2
7  gate  find, put_find_gate_util, find, 4
segdef find

find:  
tra  find.e
zero  4*2,0

find.e:

7  tax2  .setup
short_call put_find_gate_util $find(ap|0)

return

8  gate  put, put_find_gate_util, put, 4
segdef put

put:  
tra  put.e
zero  4*2,0

put.e:

8  tax2  .setup
short_call put_find_gate_util $put(ap|0)

return

9  end

NO LITERALS
### NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

<table>
<thead>
<tr>
<th>Address</th>
<th>Segment Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>000120</td>
<td>5a 000003 000000</td>
</tr>
<tr>
<td>000121</td>
<td>5a 000032 000000</td>
</tr>
<tr>
<td>000122</td>
<td>aa 000000 000000</td>
</tr>
<tr>
<td>000123</td>
<td>5a 000012 000002</td>
</tr>
<tr>
<td>000124</td>
<td>5a 000002 000003</td>
</tr>
<tr>
<td>000125</td>
<td>5a 000006 000012</td>
</tr>
<tr>
<td>000126</td>
<td>aa 016 160 165 164</td>
</tr>
<tr>
<td>000127</td>
<td>aa 137 146 151 156</td>
</tr>
<tr>
<td>000130</td>
<td>aa 144 137 147 141</td>
</tr>
<tr>
<td>000131</td>
<td>aa 164 145 137 000</td>
</tr>
<tr>
<td>000132</td>
<td>5a 000016 000003</td>
</tr>
<tr>
<td>000133</td>
<td>0a 000002 000000</td>
</tr>
<tr>
<td>000134</td>
<td>5a 000015 000003</td>
</tr>
<tr>
<td>000135</td>
<td>aa 003 160 165 164</td>
</tr>
<tr>
<td>000136</td>
<td>5a 000023 000012</td>
</tr>
<tr>
<td>000137</td>
<td>0a 000001 000000</td>
</tr>
<tr>
<td>000140</td>
<td>5a 000021 000003</td>
</tr>
<tr>
<td>000141</td>
<td>aa 004 146 151 156</td>
</tr>
<tr>
<td>000142</td>
<td>aa 144 000 000 000</td>
</tr>
<tr>
<td>000143</td>
<td>5a 000002 000016</td>
</tr>
<tr>
<td>000144</td>
<td>5a 000000 000002</td>
</tr>
<tr>
<td>000145</td>
<td>5a 000026 000003</td>
</tr>
<tr>
<td>000146</td>
<td>aa 014 163 171 155</td>
</tr>
<tr>
<td>000147</td>
<td>aa 142 157 154 137</td>
</tr>
<tr>
<td>000150</td>
<td>aa 164 141 142 154</td>
</tr>
<tr>
<td>000151</td>
<td>aa 145 000 000 000</td>
</tr>
</tbody>
</table>

### DEFINITIONS HASH TABLE

<table>
<thead>
<tr>
<th>Address</th>
<th>Segment Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>000152</td>
<td>aa 000000 000015</td>
</tr>
<tr>
<td>000153</td>
<td>aa 000000 000000</td>
</tr>
<tr>
<td>000154</td>
<td>aa 000000 000000</td>
</tr>
<tr>
<td>000155</td>
<td>5a 000012 000000</td>
</tr>
<tr>
<td>000156</td>
<td>aa 000000 000000</td>
</tr>
<tr>
<td>000157</td>
<td>aa 000000 000000</td>
</tr>
<tr>
<td>000158</td>
<td>aa 000000 000000</td>
</tr>
<tr>
<td>000159</td>
<td>aa 000000 000000</td>
</tr>
<tr>
<td>000160</td>
<td>aa 000000 000000</td>
</tr>
<tr>
<td>000161</td>
<td>5a 000016 000000</td>
</tr>
<tr>
<td>000162</td>
<td>5a 000023 000000</td>
</tr>
<tr>
<td>000163</td>
<td>aa 000000 000000</td>
</tr>
<tr>
<td>000164</td>
<td>aa 000000 000000</td>
</tr>
<tr>
<td>000165</td>
<td>aa 000000 000000</td>
</tr>
<tr>
<td>000166</td>
<td>aa 000000 000000</td>
</tr>
<tr>
<td>000167</td>
<td>aa 000000 000000</td>
</tr>
</tbody>
</table>

### EXTERNAL NAMES

<table>
<thead>
<tr>
<th>Address</th>
<th>Segment Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>000170</td>
<td>023 160 165 164</td>
</tr>
<tr>
<td>000171</td>
<td>aa 137 146 151 156</td>
</tr>
<tr>
<td>000172</td>
<td>aa 144 137 147 141</td>
</tr>
<tr>
<td>000173</td>
<td>aa 164 145 137 165</td>
</tr>
<tr>
<td>000174</td>
<td>aa 164 151 154 137</td>
</tr>
<tr>
<td>000175</td>
<td>aa 007 163 151 147</td>
</tr>
<tr>
<td>000176</td>
<td>aa 156 141 154 137</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Segment Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>000177</td>
<td>aa 008 160 154 137</td>
</tr>
<tr>
<td>000178</td>
<td>aa 137 146 151 156</td>
</tr>
<tr>
<td>000179</td>
<td>aa 144 137 147 141</td>
</tr>
<tr>
<td>000180</td>
<td>aa 164 145 137 165</td>
</tr>
<tr>
<td>000181</td>
<td>aa 164 151 154 137</td>
</tr>
<tr>
<td>000182</td>
<td>aa 007 163 151 147</td>
</tr>
<tr>
<td>000183</td>
<td>aa 156 141 154 137</td>
</tr>
</tbody>
</table>
NO TRAP POINTER WORDS

**TYPE PAIR BLOCKS**

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Value1</th>
<th>Value2</th>
</tr>
</thead>
<tbody>
<tr>
<td>000177</td>
<td>aa</td>
<td>000004</td>
<td>000000</td>
</tr>
<tr>
<td>000200</td>
<td>55</td>
<td>000050</td>
<td>000015</td>
</tr>
<tr>
<td>000201</td>
<td>aa</td>
<td>000004</td>
<td>000000</td>
</tr>
<tr>
<td>000202</td>
<td>55</td>
<td>000050</td>
<td>000021</td>
</tr>
<tr>
<td>000203</td>
<td>aa</td>
<td>000004</td>
<td>000000</td>
</tr>
<tr>
<td>000204</td>
<td>55</td>
<td>000055</td>
<td>000055</td>
</tr>
<tr>
<td>000205</td>
<td>aa</td>
<td>000001</td>
<td>000000</td>
</tr>
<tr>
<td>000206</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
</tr>
</tbody>
</table>

**INTERNAL EXPRESSION WORDS**

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Value1</th>
<th>Value2</th>
</tr>
</thead>
<tbody>
<tr>
<td>000207</td>
<td>5a</td>
<td>000057</td>
<td>000000</td>
</tr>
<tr>
<td>000210</td>
<td>5a</td>
<td>000061</td>
<td>000000</td>
</tr>
<tr>
<td>000211</td>
<td>5a</td>
<td>000063</td>
<td>000000</td>
</tr>
</tbody>
</table>
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
000011  5a  000071  0000  00
000012  9a  777766  0000  46
000013  5a  000070  0000  00
000014  9a  777764  0000  46
000015  5a  000067  0000  00

signal

put_find_gate_util

put_find_gate_util
### SYMBOL INFORMATION

#### SYMBOL TABLE HEADER

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value 1</th>
<th>Value 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>aa</td>
<td>000000</td>
</tr>
<tr>
<td>000001</td>
<td>aa</td>
<td>163171</td>
</tr>
<tr>
<td>000002</td>
<td>aa</td>
<td>164162</td>
</tr>
<tr>
<td>000003</td>
<td>aa</td>
<td>000000</td>
</tr>
<tr>
<td>000004</td>
<td>aa</td>
<td>000000</td>
</tr>
<tr>
<td>000005</td>
<td>aa</td>
<td>124344</td>
</tr>
<tr>
<td>000006</td>
<td>aa</td>
<td>000000</td>
</tr>
<tr>
<td>000007</td>
<td>aa</td>
<td>564756</td>
</tr>
<tr>
<td>000010</td>
<td>aa</td>
<td>141154</td>
</tr>
<tr>
<td>000011</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000012</td>
<td>aa</td>
<td>000024</td>
</tr>
<tr>
<td>000013</td>
<td>aa</td>
<td>000034</td>
</tr>
<tr>
<td>000014</td>
<td>aa</td>
<td>000044</td>
</tr>
<tr>
<td>000015</td>
<td>aa</td>
<td>000002</td>
</tr>
<tr>
<td>000016</td>
<td>aa</td>
<td>000064</td>
</tr>
<tr>
<td>000017</td>
<td>aa</td>
<td>000000</td>
</tr>
<tr>
<td>000020</td>
<td>aa</td>
<td>000000</td>
</tr>
<tr>
<td>000021</td>
<td>aa</td>
<td>000000</td>
</tr>
<tr>
<td>000022</td>
<td>aa</td>
<td>000134</td>
</tr>
<tr>
<td>000023</td>
<td>aa</td>
<td>000064</td>
</tr>
<tr>
<td>000024</td>
<td>aa</td>
<td>101114</td>
</tr>
<tr>
<td>000025</td>
<td>aa</td>
<td>126145</td>
</tr>
<tr>
<td>000026</td>
<td>aa</td>
<td>151157</td>
</tr>
<tr>
<td>000027</td>
<td>aa</td>
<td>066056</td>
</tr>
<tr>
<td>000030</td>
<td>aa</td>
<td>040112</td>
</tr>
<tr>
<td>000031</td>
<td>aa</td>
<td>145040</td>
</tr>
<tr>
<td>000032</td>
<td>aa</td>
<td>067071</td>
</tr>
<tr>
<td>000033</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000034</td>
<td>aa</td>
<td>106162</td>
</tr>
<tr>
<td>000035</td>
<td>aa</td>
<td>155145</td>
</tr>
<tr>
<td>000036</td>
<td>aa</td>
<td>115105</td>
</tr>
<tr>
<td>000037</td>
<td>aa</td>
<td>144155</td>
</tr>
<tr>
<td>000040</td>
<td>aa</td>
<td>056141</td>
</tr>
<tr>
<td>000041</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000042</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000043</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000044</td>
<td>aa</td>
<td>154151</td>
</tr>
<tr>
<td>000045</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000046</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000047</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000050</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000051</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000052</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000053</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000054</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000055</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000056</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000057</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000060</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000061</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>000062</td>
<td>aa</td>
<td>040040</td>
</tr>
<tr>
<td>Address</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>000063</td>
<td>040040</td>
<td></td>
</tr>
<tr>
<td>000064</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>000065</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>000066</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>000067</td>
<td>063453</td>
<td></td>
</tr>
<tr>
<td>000070</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>000071</td>
<td>564153</td>
<td></td>
</tr>
<tr>
<td>000072</td>
<td>000106</td>
<td></td>
</tr>
<tr>
<td>000073</td>
<td>057660</td>
<td></td>
</tr>
<tr>
<td>000074</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>000075</td>
<td>053324</td>
<td></td>
</tr>
<tr>
<td>000076</td>
<td>076165</td>
<td></td>
</tr>
<tr>
<td>000077</td>
<td>076106</td>
<td></td>
</tr>
<tr>
<td>000100</td>
<td>104076</td>
<td></td>
</tr>
<tr>
<td>000101</td>
<td>076160</td>
<td></td>
</tr>
<tr>
<td>000102</td>
<td>137146</td>
<td></td>
</tr>
<tr>
<td>000103</td>
<td>144137</td>
<td></td>
</tr>
<tr>
<td>000104</td>
<td>161145</td>
<td></td>
</tr>
<tr>
<td>000105</td>
<td>141154</td>
<td></td>
</tr>
<tr>
<td>000106</td>
<td>076154</td>
<td></td>
</tr>
<tr>
<td>000107</td>
<td>076151</td>
<td></td>
</tr>
<tr>
<td>000110</td>
<td>154165</td>
<td></td>
</tr>
<tr>
<td>000111</td>
<td>076147</td>
<td></td>
</tr>
<tr>
<td>000112</td>
<td>145137</td>
<td></td>
</tr>
<tr>
<td>000113</td>
<td>143162</td>
<td></td>
</tr>
<tr>
<td>000114</td>
<td>056151</td>
<td></td>
</tr>
<tr>
<td>000115</td>
<td>154066</td>
<td></td>
</tr>
<tr>
<td>000116</td>
<td>155040</td>
<td></td>
</tr>
</tbody>
</table>

`>udd>F15D>s1>put_find_gate_.alm`

`>ldd>inlude>gate_macros.inol.alm`
MULTICS ASSEMBLY CROSS REFERENCE LISTING

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbol</th>
<th>Source file</th>
<th>Line number</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>.def s loop</td>
<td>put_find_gate:</td>
<td>6.</td>
</tr>
<tr>
<td>76</td>
<td>.gate_errorname</td>
<td>put_find_gate:</td>
<td>6.</td>
</tr>
<tr>
<td>46</td>
<td>.next def</td>
<td>put_find_gate:</td>
<td>6.</td>
</tr>
<tr>
<td>102</td>
<td>.no_gate_error</td>
<td>put_find_gate:</td>
<td>6.</td>
</tr>
<tr>
<td>23</td>
<td>.return name</td>
<td>put_find_gate:</td>
<td>6.</td>
</tr>
<tr>
<td>51</td>
<td>.setup</td>
<td>put_find_gate:</td>
<td>6, 7, 8</td>
</tr>
<tr>
<td>1</td>
<td>find</td>
<td>put_find_gate:</td>
<td>6, 7, 8</td>
</tr>
<tr>
<td>104</td>
<td>find.e</td>
<td>put_find_gate:</td>
<td>7.</td>
</tr>
<tr>
<td>4</td>
<td>main</td>
<td>put_find_gate:</td>
<td>6, 7, 8</td>
</tr>
<tr>
<td>2</td>
<td>put</td>
<td>put_find_gate:</td>
<td>8.</td>
</tr>
<tr>
<td>112</td>
<td>put.e</td>
<td>put_find_gate:</td>
<td>8.</td>
</tr>
<tr>
<td></td>
<td>put_find_gate_util</td>
<td>put_find_gate:</td>
<td>7, 8</td>
</tr>
<tr>
<td>70</td>
<td>signal</td>
<td>put_find_gate:</td>
<td>6.</td>
</tr>
<tr>
<td>0</td>
<td>transfer_vector</td>
<td>put_find_gate:</td>
<td>6, 7, 8</td>
</tr>
</tbody>
</table>

NO FATAL ERRORS.
trans.list

COMPILATION LISTING OF SEGMENT trans
Compiled by: Multics PL/I Compiler, Release 250, of February 18, 1980
Compiled at: Honeywell LISD Phoenix, System M
Compiled on: 03/19/80 1043.7 most Wed
Options: map

1 trans: proc;
2 /*
3 *
4 Name: trans
5 *
6 The trans command adds, counts, deletes, reads, and summarizes
7 transactions resident in a message segment whose pathname is
8 >udd>F15D>trans.ms. The amount of information it may return and the
9 ability to add "transactions" into the segment are governed by
10 extended access.
11
12 Usage
13 trans key [args]
14 where:
15
16 1. key
17 is one of the functions listed below.
18
19 add, a
20  adds a transaction to the message segment. Three
21 additional arguments must be furnished:
22  1. part name
23  2. unit price
24  3. how many sold
25 The caller must have 'append' extended access to
26 the segment trans.ms.
27
28 count, c
29 returns the number of transactions (messages) in
30 the segment. The caller must have status extended
31 access to the segment trans.ms.
32
33 delete, d
34 deletes the "current" transaction. A transaction
35
is made current by some previous read operation. If the last transaction read was not added by the caller, the caller must have delete extended access on trans.ms. Otherwise, the caller must at least have 'own' extended access.

read, r

reads one or more transactions. If an additional argument is furnished, it must be one of the following:

1. all
   If user has 'read' extended access, every transaction is dumped.
   If user has 'own', but not 'read', only 'own' transactions are dumped.

2. first
   If user has 'read' extended access, the first transaction is dumped.
   If user has 'own', but not 'read', the first 'own' transaction is dumped.

3. last
   analogous to 'trans read first'

4. next
   If user has 'read' extended access, the next transaction in trans.ms is dumped.
   If user has 'own', but not 'read', the caller's next transaction is dumped.

5. prior
   analogous to 'trans read next'

6. <argument missing>
   reads the current message.

summary, s

If the caller has 'read' extended access, the dollar amount grand total of all transactions in the segment is returned. If the caller has 'own', but not 'read', the dollars amount grand total of 'own' transactions is returned instead.

2. args

is dependent upon key as documented above.
/*
96     dol HE internal static options (constant) char (5) init ("trans");
97     dol TARGET SEGMENT int static options (constant) char (8) init ("trans.ms");
98     dol TARGET DIR int static options (constant) char (9) init ("\$addF15d");
99     dol (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     dol areap ptr;
103     dol (argp, messagep, arg_list ptr) ptr;
104     */ BEGIN INCLUDE FILE . . . msg_return_args_v3.incl.pl1 */
1
2
3
4     /* structure returned when message is read from a message segment */
5
6     dol ms_arg_ptr ptr;
7
8     dol 1 msg_return_args based (ms_arg_ptr) aligned;
9        2 ms_ptr ptr,
10        2 ms_len fixed bin (18),
11        2 sender id char (32),
12        2 level fixed bin,
13        2 ms_id bit (72),
14        2 sender_authorization bit (72),
15        2 access_class bit (72);
16
17     */ END INCLUDE FILE . . . msg_return_args_v3.incl.pl1 */
104
105     dol (code, binary_number_sold) bin (35);
106     dol index fixed bin;
107
108     dol message_segment $open entry (char (*), char (*), fixed bin, fixed bin (35));
109     dol message_segment $close entry (fixed bin, fixed bin, fixed bin (35));
110     dol message_segment $get message_count index entry (fixed bin, fixed bin, fixed bin (35));
111     dol message_segment $add index entry (fixed bin, ptr, fixed bin (24), bit (72) aligned, bin (35));
112     dol message_segment $delete_index entry (fixed bin, bit (72) aligned, fixed bin (35));
113     dol message_segment $read_index entry (fixed bin, ptr, bit (1) aligned, ptr, fixed bin (35));
114     dol message_segment $incremental_read_index entry (fixed bin, ptr, bit (2) aligned, bit (72) aligned, ptr, fixed bin (35));
115     dol message_segment $own_read_index entry (fixed bin, ptr, bit (1) aligned, ptr, fixed bin (35));
116     dol message_segment $own_incremental_read_index entry (fixed bin, ptr, bit (2) aligned, bit (72) aligned, ptr, fixed bin (35));
117
118     dol 1 trans_msg based (ms_arg_ptr -> msg_return_args.ms_ptr), /* Sometimes pointed to by message */
119     2 widget_name char (15),
120     2 unit_price pic "999999$9.9999",
121     2 how_many_sold fixed dec (5, 0),
122     2 total_cost pic "999999$9.9999";
123     dol grand total dollars internal static pic "999999$9.9999";
124     dol silent bit (1) internal static init ("0"b);
125     dol read option (2:6) char (5) internal static options (constant)
126     init ("all", "next", "prior", "first", "last");
127     dol current_message_id bit (72) internal static aligned init ("0"b);
128     dol looping_index fixed bin (17) init (1);
own bit (1) init ("0"b);
read options internal static options (constant) char (121)
init ("Your read options are:
        trans read
        trans read all
        trans read next
        trans read prior
        trans read first
        trans read last");
read options (variable);
    entry options (constant) char (121)
    ou$_arg_count entry (fixed bin);
cu$_arg ptr entry (fixed bin, ptr, fixed bin(21), fixed bin (35));
cu$_arg list ptr entry (ptr);
cu$_arg ptr rel entry (fixed bin, ptr, fixed bin(21), fixed bin (35), ptr);
nargs fixed bin, argl fixed bin(21);
    argl char (argl) based (argp);
auto_area area (2048); /* Should be large enough for any and all allocated structures */
string char (24);
date time entry (fixed bin (71), char (1));
clock entry returns (fixed bin (71));
cv_deo_check entry (char (1), fixed bin (35)) returns (fixed bin (35));
conversion condition;
binary_time fixed bin (71);
alternate_binary_time bit (72) aligned based;
(size, addr) builtin;
message_count;
iox $control entry (ptr, char(1)), ptr, fixed bin(35));
iox $user_io ext static ptr; /*
/* COMMON BEGINNING POINT */

call ou $arg_count (nargs);
if nargs = 0 then do;
call com_err_ (error_table $wrong_no_of_args, "trans",
"You have not invoked 'trans' properly.
Please type 'trans key {args}', where 'key' is either
'add', 'delete', 'read', 'summary', or 'count'.
'args' is a function of what key has been supplied.");
return;
end;
call ou $arg_ptr (1, argp, argl, code);
call ou $arg_list_ptr (arg_list_ptr); /* Sets an automatic pointer */
if arg = "add" | arg = "a" then call trans add;
else if arg = "delete" | arg = "d" then call trans delete;
else if arg = "read" | arg = "r" then call trans read;
else if arg = "summary" | arg = "s" then call trans_summary;
else if arg = "count" | arg = "c" then call trans_count;
else call com_err_ (error_table $bad_arg, HE, arg);
return; /*
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 ou$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 "s"." TARGET_DIR, TARGET_SEGMENT);
    return;
end;

areap = addr (auto_area);
allocate msg_return_arg 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" then do;
    call com_err_ (0, ME, "There is no current message.");
    return;
end;

else do;
    call message_segment$incremental_read_index (index, areap, "00", current_message_id, ms_arg_ptr, code); /* User must have 'r' extended access */
    if code ^= error_table$moderr then call message_segment$own_incremental_read_index (index, areap, "00", current_message_id, ms_arg_ptr, code);
    if code ^= 0 then do;
        call com_err_ (code, ME);
        call message_segment$close (index, code);
        return;
    end;

call print_message ();

call message_segment$close (index, code);
return;
end;
read_label (2):

    call message_segment $read_index (index, areap, "O"b /& from the first */,
    ms_arg_ptr, code);
    if code = error_table $moderr then do;
        call message_segment $own_read_index
        (index, areap, "O"b, ms_arg_ptr, code);
        if code = error_table $moderr then do;
            call com_err (code, ME);
            call message_segment $close (index, code);
            return;
        end;
    own = "1"b;
    end;
    if code = error_table $no_message then do;
        call com_err (code, ME, "[You have no] messages in "a>a.", own, TARGET_DIR, TARGET_SEGMENT);
        call message_segment $close (index, code);
        return;
    end;
    if own then do;
        do while (code ^= error_table $no_message);
            call print_message ();
            current_message_id = ms_arg_ptr -> msg_return_args.ms_id;
            call message_segment $own_incremental_read_index (index, areap, "0"b /& next */,
            current_message_id, ms_arg_ptr, code);
        end;
        end;
        else do;
            do while (code ^= error_table $no_message);
                call print_message ();
                current_message_id = ms_arg_ptr -> msg_return_args.ms_id;
                call message_segment $incremental_read_index (index, areap, "01"b /& next */,
                current_message_id, ms_arg_ptr, code);
            end;
        end;
    end;
    call message_segment $close (index, code);
    return;
read_label (3):

    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, "01"b,
current_message_id, ms_arg_ptr, code);
if code = error_table $moderr then call message_segment $own_incremental_read_index (index,
areap, "0"b, current_message_id, ms_arg_ptr, code);
if code = error_table $moderr then do;
call com_err (code, ME);
call message_segment $close (index, code);
return;
end;
if code = error_table $no_message then do;
call com_err (code, ME, "There is no next message.");
call message_segment $close (index, code);
return;
end;
call print_message ();
current_message_id = ms_arg_ptr -> ms_id; /* Update what is now the current msg */
call message_segment $close (index, code);
return;
end;
read_label (4): /* 'trans read prior' */
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, "10"b,
current_message_id, ms_arg_ptr, code);
if code = error_table $moderr then call message_segment $own_incremental_read_index (index,
areap, "0"b, current_message_id, ms_arg_ptr, code);
if code = error_table $moderr then do;
call com_err (code, ME);
call message_segment $close (index, code);
return;
end;
if code = error_table $no_message then do;
call com_err (code, ME, "There is no prior message.");
call message_segment $close (index, code);
return;
end;
call print_message ();
current_message_id = ms_arg_ptr -> ms_id; /* Update what is now the current msg */
call message_segment $close (index, code);
return;
end;
read_label (5): /* 'trans read first' */
call message_segment $read_index (index, areap, "0"b /* the first */,
ms_arg_ptr, code);
if code = error_table $moderr then do;
    call message_segment $own_read_index (index, areap, "0"b /* the first */,
        ms_arg_ptr, code);
    if code = error_table $moderr then do;
        call com_err (code, ME);
        call message_segment $close (index, code);
    return;
end;
oown = "1"b;
end;
if code = error_table $no_message then do;
    call com_err (code, ME, "You have no"; No] messages in "a"="a.", own, TARGET_DIR, TARGET_SEGMENT);
    call message_segment $close (index, code);
return;
end;
call print_message ();
current_message_id = ms_arg_ptr -> mseg_return_args.ms_id; /* For a possible 'trans-read' in the future */
call message_segment $close (index, code);
return;
end
read_label (6):
    /* 'trans-read last' */
call message_segment $read_index (index, areap, "1"b /* the last */,
        ms_arg_ptr, code);
if code = error_table $moderr then do;
    call message_segment $own_read_index (index, areap, "1"b /* the last */,
        ms_arg_ptr, code);
    oown = "1"b;
end;
if code = error_table $no_message then do;
    call com_err (code, ME, "You have no"; No] messages in "a"="a.", own, TARGET_DIR, TARGET_SEGMENT);
    call message_segment $close (index, code);
return;
end;
call print_message ();
current_message_id = ms_arg_ptr -> mseg_return_args.ms_id; /* For a possible 'trans-read' in the future */
call message_segment $close (index, code);
return;
end trans_read;
/*
trans_add: proc;
  if nargs ~= 4 then do;
    call com_err (error_table$_wrong_no_of_args, ME, "Proper invocation:"
    trans add; widget_name unit_price how_many_sold");
  return;
end;

allocate trans_msg in (auto_area) set (messagep);
  call ou $arg_ptr_rel (2, argp, arg1, code, arg_list_ptr);
if arg1 > 16 then do;
  call com_err (error_table$_bigarg, ME, "Widget_name mustn’t exceed 16 characters.");
  return;
end;
else messagep -> trans_msg.widget_name = arg;
  call ou $arg_ptr_rel (3, argp, arg1, code, arg_list_ptr);
  on conversion begin;
    call com_err (error_table$_bad_arg, ME, "Unit price specified does not look something like 9999.99");
  goto add_return_point;
end;
messagep -> trans_msg.unit_price = arg;
revert conversion;
  /* If we get here, conversion went fine */
  call ou $arg_ptr_rel (4, argp, arg1, code, arg_list_ptr);
  binary_number_sold = ov_dec_check_ (arg, code);
if code ~= 0 then do;
  call com_err (error_table$_bad_arg, ME, "Number sold could not be converted into an integer.");
  return;
end;
else if binary_number_sold > 9999 then do;
  call com_err (error_table$_bad_arg, ME, "Number sold exceeds 99999.");
  return;
end;
else messagep -> trans_msg.how_many_sold = binary_number_sold;
messagep -> trans_msg.total_cost = messagep -> trans_msg.unit_price *
messagep -> trans_msg.how_many_sold; /* trans_msg structure ready for addition to >udd>F15d>trans. */
call message_segment$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;
binary_time = clock (); /* Need a unique id for message_id, hence clock */
call message_segment$add_index (index, messagep, 36 /* size (messagep -> trans_msg)
  addr (binary_time) -> alternate_binary_time, code));
if code ~= 0 then call com_err_ (code, ME, "While attempting to add a transaction.");
else do;
current_message_id = addr (binary_time) -> alternate_binary_time;
call date_time (((current_message_id), string);
call ioa_ ("Transaction added at "a", string); 438
end;
call message_segment_close (index, code);
add_return_point:
end trans_add;
trans_delete: proc;
  if nargs = 1 then do;
    call com_err_ (error_table$wrong_no_of_args, ME, "Proper invocation: trans delete");
    return;
  end;
  if current_message_id = "O"b then do;
    call com_err_ (0, ME, "There is no current transaction to delete.
    Try some kind of read first.");
    return;
  end;
  else do;
    call message_segment$open (TARGET_DIR, TARGET_SEGMENT, index, code);
    if code = 0 then do;
      call com_err_ (code, ME, "While attempting to open "a">"a.", TARGET_DIR, TARGET_SEGMENT);
    return;
    end;
    call message_segment$delete_index (index, current_message_id, code);
    if code = 0 then do;
      current_message_id = "O"b;
      call ioa_ ("Transaction deleted.");
    end;
    else call com_err_ (code, ME, "While attempting to delete the current message.");
    call message_segment$close (index, code);
  end;
end trans_delete;
trans_summary: proc;
do1 trans entry options (variable);
    if nargs ^= 1 then do;
        call com_err (error_table$wrong_no_of_args, ME, "Proper invocation:
trans summary");
        return;
    end;  
/* We can not tolerate a quit, so ... */
    call iox$control (iox$user io, "quit_disable", null (), code);
    silent = "1"b;
    grand_total_dollars = 0;
    call trans ("read", "all");
    call iox_ ("Grand total = ", grand_total_dollars);
    silent = "0"b;
    grand_total_dollars = 0;
    /* Now safe to unmask quits */
    call iox$control (iox$user io, "quit_enable", null (), code);
end trans_summary;
/*
/*
496 trans_count: proc;
497 /* User must have status extended access on trans.ms */
498 if nargs ^= 1 then do;
499     call com_err (error_table.$wrong_no_of_args, ME, "Proper invocation: 
500         trans count");
501     return;
502 end;
503
504 call message_segment.$open (TARGET_DIR, TARGET_SEGMENT, index, code);
505 if code ^= 0 then do;
506     call com_err (code, ME, "While attempting to open "a". "a." TARGET_DIR, TARGET_SEGMENT);
507     return;
508 end;
509
510 call message_segment.$get message count index (index, message_count, code);
511 if code ^= 0 then call ioa ("There are "d messages in "a", message_count, TARGET_SEGMENT);
512 else call com_err (code, ME);
513
514 call message_segment.$close (index, code);
515
516 end trans_count;
517 */
print_message: proc();

  /* Can tally dollars OR print transaction info */
  call date_time ((ms_arg_ptr -> ms_id), string);
  if "silent" then call_log ("At "#a,""a sold "#d "a," at a total cost of "#a, (unit price = ",
                     string, ms_arg_ptr -> msg_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;
<table>
<thead>
<tr>
<th>LINE</th>
<th>NUMBER</th>
<th>DATE MODIFIED</th>
<th>NAME</th>
<th>PATHNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>03/11/80</td>
<td>trans.p11</td>
<td>&gt;user_dir_dir&gt;F15D&gt;Student_01&gt;trans.p11</td>
</tr>
<tr>
<td>104</td>
<td>1</td>
<td>08/03/77</td>
<td>mseg_return_args_v3.inol.p11</td>
<td>&gt;ldd&gt;Include&gt;mseg_return_args_v3.inol.p11</td>
</tr>
</tbody>
</table>
### NAMES DECLARED IN THIS COMPILATION.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES (* indicates a set context)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ME</td>
<td>000062</td>
<td>constant</td>
<td>char(5)</td>
<td>initial unaligned dol 96 set ref 176* 182* 194* 201*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>213* 223* 241* 248* 282* 292* 297* 312* 322* 327*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>347* 354* 375* 389* 397* 404* 412* 416* 427* 433*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>446* 451* 458* 466* 476* 501* 508* 514*</td>
</tr>
<tr>
<td>TARGET_DIR</td>
<td>000054</td>
<td>constant</td>
<td>char(9)</td>
<td>initial unaligned dol 98 set ref 199* 201* 248* 354*</td>
</tr>
<tr>
<td>TARGET_SEGMENT</td>
<td>000060</td>
<td>constant</td>
<td>char(8)</td>
<td>initial unaligned dol 97 set ref 199* 201* 248* 354*</td>
</tr>
<tr>
<td>addr</td>
<td></td>
<td>based</td>
<td></td>
<td>375* 425* 427* 456* 458* 506* 508* 513*</td>
</tr>
<tr>
<td>alternate_binary_time</td>
<td></td>
<td>based</td>
<td></td>
<td>375* 425* 427* 456* 458* 506* 508* 513*</td>
</tr>
<tr>
<td>areap</td>
<td></td>
<td>automatic</td>
<td></td>
<td>000000 stack reference condition</td>
</tr>
<tr>
<td>arg</td>
<td></td>
<td>automatic</td>
<td></td>
<td>dol 150 ref 403 408</td>
</tr>
<tr>
<td>arg_list_ptr</td>
<td>000106</td>
<td>automatic</td>
<td>pointer</td>
<td>dol 105 ref 403 408</td>
</tr>
<tr>
<td>argl</td>
<td>000120</td>
<td>automatic</td>
<td>pointer</td>
<td>dol 105 ref 403 408</td>
</tr>
<tr>
<td>argp</td>
<td>000102</td>
<td>automatic</td>
<td>pointer</td>
<td>dol 151 ref 430 431 435</td>
</tr>
<tr>
<td>auto_area</td>
<td>000122</td>
<td>automatic</td>
<td>area(248)</td>
<td>dol 114 ref 430 431 435</td>
</tr>
<tr>
<td>binary_number_sold</td>
<td>000113</td>
<td>automatic</td>
<td>area(148)</td>
<td>dol 114 ref 430 431 435</td>
</tr>
<tr>
<td>binary_time</td>
<td>000130</td>
<td>automatic</td>
<td>fixed bin(35)</td>
<td>dol 105 ref 410 415 419</td>
</tr>
<tr>
<td>clock</td>
<td>000072</td>
<td>automatic</td>
<td>fixed bin(35)</td>
<td>dol 105 ref 410 415 419</td>
</tr>
<tr>
<td>code</td>
<td>000112</td>
<td>automatic</td>
<td>fixed bin(35)</td>
<td>dol 105 ref 410 415 419</td>
</tr>
<tr>
<td>com_err_</td>
<td>000054</td>
<td>constant</td>
<td>entry</td>
<td>dol 150 ref 403 408</td>
</tr>
<tr>
<td>conversion</td>
<td>000000</td>
<td>stack reference</td>
<td>condition</td>
<td>dol 150 ref 403 408</td>
</tr>
<tr>
<td>cu $arg_count</td>
<td>000060</td>
<td>constant</td>
<td>entry</td>
<td>dol 105 ref 410 415 419</td>
</tr>
<tr>
<td>cu $arg_list_ptr</td>
<td>000064</td>
<td>constant</td>
<td>entry</td>
<td>dol 105 ref 410 415 419</td>
</tr>
<tr>
<td>cu $arg_ptr</td>
<td>000062</td>
<td>constant</td>
<td>entry</td>
<td>dol 105 ref 410 415 419</td>
</tr>
<tr>
<td>cu $arg_ptr_rel</td>
<td>000066</td>
<td>constant</td>
<td>entry</td>
<td>dol 105 ref 410 415 419</td>
</tr>
<tr>
<td>current_message_id</td>
<td>000014</td>
<td>internal static</td>
<td>bit(72)</td>
<td>dol 105 ref 410 415 419</td>
</tr>
<tr>
<td>cv _dec _check</td>
<td>000074</td>
<td>constant</td>
<td>entry</td>
<td>dol 150 ref 403 408</td>
</tr>
<tr>
<td>date_time</td>
<td>000070</td>
<td>constant</td>
<td>entry</td>
<td>dol 105 ref 410 415 419</td>
</tr>
<tr>
<td>error_tableabad_arg</td>
<td>000024</td>
<td>external static</td>
<td>fixed bin(35)</td>
<td>dol 99 set ref 176* 194* 404* 412* 416*</td>
</tr>
</tbody>
</table>
NAMES DECLARED BY EXPLICIT CONTEXT:

add_return_point 0031165 constant label
dol 440 ref 405
dol 531
constant bottom of trans 001025 constant label
internal dol 520 ref 228 257 268 302 332 359 380
print_message 004330 constant entry
array(6) dol 210 ref 208
read_label 000000 constant label
external dol 1
trans 000570 constant entry
internal dol 387 ref 171
trans_add 002644 constant entry
internal dol 496 ref 175
trans_count 004122 constant entry
internal dol 443 ref 172
trans_delete 003466 constant entry
internal dol 179 ref 173
trans_read 001026 constant entry
internal dol 471 ref 174
trans_summary 003727 constant entry

NAMES DECLARED BY CONTEXT OR IMPLICATION:
empty null

STORAGE REQUIREMENTS FOR THIS PROGRAM:

<table>
<thead>
<tr>
<th>Object</th>
<th>Text</th>
<th>Link</th>
<th>Symbol</th>
<th>Defs</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>5462</td>
<td>4501</td>
<td>104</td>
<td>315</td>
<td>322</td>
</tr>
<tr>
<td>Length</td>
<td>104</td>
<td>4501</td>
<td>315</td>
<td>322</td>
<td>6</td>
</tr>
<tr>
<td>BLOCK NAME</td>
<td>STACK SIZE</td>
<td>TYPE</td>
<td>WHY NONQUICK/WHO SHARES STACK FRAME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trans</td>
<td>2555</td>
<td>external procedure</td>
<td>is an external procedure.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trans_read</td>
<td>262</td>
<td>internal procedure</td>
<td>shares stack frame of external procedure trans.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>on unit on line 403</td>
<td>94</td>
<td>internal procedure</td>
<td>enables or reverts conditions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trans_delete</td>
<td>262</td>
<td>internal procedure</td>
<td>shares stack frame of external procedure trans.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trans_summary</td>
<td>262</td>
<td>internal procedure</td>
<td>shares stack frame of external procedure trans.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trans_count</td>
<td>262</td>
<td>internal procedure</td>
<td>shares stack frame of external procedure trans.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>print_message</td>
<td>262</td>
<td>internal procedure</td>
<td>shares stack frame of external procedure trans.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

STORAGE FOR INTERNAL STATIC VARIABLES:

<table>
<thead>
<tr>
<th>LOC IDENTIFIER</th>
<th>BLOCK NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>000010</td>
<td>grand_total_dollars</td>
</tr>
<tr>
<td>000013</td>
<td>silent</td>
</tr>
<tr>
<td>000014</td>
<td>current_message_id</td>
</tr>
</tbody>
</table>

STORAGE FOR AUTOMATIC VARIABLES:

<table>
<thead>
<tr>
<th>LOC IDENTIFIER</th>
<th>BLOCK NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>000100</td>
<td>areap</td>
</tr>
<tr>
<td>000102</td>
<td>argp</td>
</tr>
<tr>
<td>000104</td>
<td>message</td>
</tr>
<tr>
<td>000106</td>
<td>arg_list_ptr</td>
</tr>
<tr>
<td>000110</td>
<td>ms_arg_ptr</td>
</tr>
<tr>
<td>000112</td>
<td>code</td>
</tr>
<tr>
<td>000113</td>
<td>binary_number_sold</td>
</tr>
<tr>
<td>000114</td>
<td>index</td>
</tr>
<tr>
<td>000115</td>
<td>looping_index</td>
</tr>
<tr>
<td>000116</td>
<td>own</td>
</tr>
<tr>
<td>000117</td>
<td>args</td>
</tr>
<tr>
<td>000120</td>
<td>argl</td>
</tr>
<tr>
<td>000122</td>
<td>auto_area</td>
</tr>
</tbody>
</table>