COMPUTER PROGRAM SPECIFICATION
FOR SECURITY KERNEL FOR PDP-11/45

APRIL 1978

Prepared for

DEPUTY FOR TECHNICAL OPERATIONS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
Hanscom Air Force Base, Bedford, Massachusetts

Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract No. F19628-75-C-0001

Approved for public release; distribution unlimited.
When U.S. Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Do not return this copy. Retain or destroy.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.

BRIAN W. WOODRUFF, Captain, USAF
Project Engineer

DONALD P. ERIKSEN
Project Engineer

FOR THE COMMANDER

JOHN T. HOLLAND, Lt Col, USAF
Chief, Techniques Engineering Div

STANLEY F. DERESKA, Colonel, USAF
Director, Computer Systems Engineering
Deputy for Technical Operations
This report presents the Type C5, Computer Program Product Specification for the Security Kernel for the PDP-11/45. It specifies the configuration information that fully describes the Security Kernel as an established program product. A detailed description of each individual function of the program is given. Also included are the requirements which provide the basis for development of verification procedures.
20. Abstract (continued)

and a section of informal notes for the potential user. Flow charts are included in an addenda to this report for the convenience of the reader. A complete listing of the Security Kernel program is presented in Section 10, Volume II.
ACKNOWLEDGMENT

This report has been prepared by The MITRE Corporation under Project No. 7070. The contract is sponsored by the Electronic Systems Division, Air Force Systems Command, Hanscom Air Force Base, Massachusetts.

This report is a compilation of contributions from the Project 7070 Staff Members, notably: Grace H. Nibaldi, John A. Larkins, Daniel F. Stock, John C. C. White and Alan R. Chen. Special acknowledgements are extended to John L. Mack for his preparation of Sections IV and V and to W. Lee Schiller for his "Design and Specification of a Security Kernel for the PDP-11/45" on which much of the material in this report is based.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Paragraph</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td></td>
</tr>
<tr>
<td>SCOPE</td>
<td>6</td>
</tr>
<tr>
<td>2.</td>
<td></td>
</tr>
<tr>
<td>REFERENCED DOCUMENTS</td>
<td>7</td>
</tr>
<tr>
<td>2.1</td>
<td></td>
</tr>
<tr>
<td>Government Documents</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>Non-Government Documents</td>
<td>7</td>
</tr>
<tr>
<td>3.</td>
<td></td>
</tr>
<tr>
<td>REQUIREMENTS</td>
<td>9</td>
</tr>
<tr>
<td>3.1</td>
<td></td>
</tr>
<tr>
<td>Functional Allocation Description</td>
<td>12</td>
</tr>
<tr>
<td>3.1.1</td>
<td></td>
</tr>
<tr>
<td>Entering and Exiting Kernel Domain</td>
<td>16</td>
</tr>
<tr>
<td>3.1.2</td>
<td></td>
</tr>
<tr>
<td>Directory Manipulation Functions</td>
<td>16</td>
</tr>
<tr>
<td>3.1.3</td>
<td></td>
</tr>
<tr>
<td>Segment Accessing</td>
<td>18</td>
</tr>
<tr>
<td>3.1.4</td>
<td></td>
</tr>
<tr>
<td>Process Cooperation</td>
<td>20</td>
</tr>
<tr>
<td>3.1.5</td>
<td></td>
</tr>
<tr>
<td>Process Control</td>
<td>21</td>
</tr>
<tr>
<td>3.1.6</td>
<td></td>
</tr>
<tr>
<td>Privileged Functions</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td></td>
</tr>
<tr>
<td>Functional Description</td>
<td>22</td>
</tr>
<tr>
<td>3.2.1</td>
<td></td>
</tr>
<tr>
<td>Gate (GATE)</td>
<td>26</td>
</tr>
<tr>
<td>3.2.2</td>
<td></td>
</tr>
<tr>
<td>Parameter Checker (PCHECK)</td>
<td>30</td>
</tr>
<tr>
<td>3.2.3</td>
<td></td>
</tr>
<tr>
<td>Create Segment (CREATE)</td>
<td>38</td>
</tr>
<tr>
<td>3.2.4</td>
<td></td>
</tr>
<tr>
<td>Delete (DELETE)</td>
<td>41</td>
</tr>
<tr>
<td>3.2.5</td>
<td></td>
</tr>
<tr>
<td>Give Access (GIVE)</td>
<td>45</td>
</tr>
<tr>
<td>3.2.6</td>
<td></td>
</tr>
<tr>
<td>Rescind Access (RESCIND)</td>
<td>51</td>
</tr>
<tr>
<td>3.2.7</td>
<td></td>
</tr>
<tr>
<td>Get Write Access (GETW)</td>
<td>54</td>
</tr>
<tr>
<td>3.2.8</td>
<td></td>
</tr>
<tr>
<td>Get Read Access (GETR)</td>
<td>57</td>
</tr>
<tr>
<td>3.2.9</td>
<td></td>
</tr>
<tr>
<td>Release Segment (DCONNECT)</td>
<td>59</td>
</tr>
<tr>
<td>3.2.10</td>
<td></td>
</tr>
<tr>
<td>Enable (ENABLE)</td>
<td>63</td>
</tr>
<tr>
<td>3.2.11</td>
<td></td>
</tr>
<tr>
<td>Disable (DISABLE)</td>
<td>67</td>
</tr>
<tr>
<td>3.2.12</td>
<td></td>
</tr>
<tr>
<td>Outer P (OUTERP)</td>
<td>71</td>
</tr>
<tr>
<td>3.2.13</td>
<td></td>
</tr>
<tr>
<td>P (P)</td>
<td>73</td>
</tr>
<tr>
<td>3.2.14</td>
<td></td>
</tr>
<tr>
<td>Outer V (OUTERV)</td>
<td>76</td>
</tr>
<tr>
<td>3.2.15</td>
<td></td>
</tr>
<tr>
<td>V (V)</td>
<td>78</td>
</tr>
<tr>
<td>3.2.16</td>
<td></td>
</tr>
<tr>
<td>Send Interprocess Communication (IPCSEND)</td>
<td>81</td>
</tr>
<tr>
<td>3.2.17</td>
<td></td>
</tr>
<tr>
<td>Receive Interprocess Communication (IPCRCV)</td>
<td>85</td>
</tr>
<tr>
<td>3.2.18</td>
<td></td>
</tr>
<tr>
<td>Stop Process (STOPP)</td>
<td>88</td>
</tr>
<tr>
<td>3.2.19</td>
<td></td>
</tr>
<tr>
<td>Read Directory (READIR)</td>
<td>91</td>
</tr>
<tr>
<td>3.2.20</td>
<td></td>
</tr>
<tr>
<td>Start Process (STARTP)</td>
<td>94</td>
</tr>
<tr>
<td>3.2.21</td>
<td></td>
</tr>
<tr>
<td>Change Object (CHANGEO)</td>
<td>100</td>
</tr>
<tr>
<td>3.2.22</td>
<td></td>
</tr>
<tr>
<td>Initialize Hierarchy (INITI)</td>
<td>104</td>
</tr>
<tr>
<td>3.2.23</td>
<td></td>
</tr>
<tr>
<td>Get Directory (GETDIR)</td>
<td>106</td>
</tr>
<tr>
<td>3.2.24</td>
<td></td>
</tr>
<tr>
<td>Write Directory (WRITEDIR)</td>
<td>108</td>
</tr>
<tr>
<td>3.2.25</td>
<td></td>
</tr>
<tr>
<td>Delete Segment (DELETSEG)</td>
<td>110</td>
</tr>
<tr>
<td>3.2.26</td>
<td></td>
</tr>
<tr>
<td>Connect (CONNECT)</td>
<td>114</td>
</tr>
<tr>
<td>3.2.27</td>
<td></td>
</tr>
<tr>
<td>Search Out and Destroy Descriptors (SOADD)</td>
<td>117</td>
</tr>
<tr>
<td>3.2.28</td>
<td></td>
</tr>
<tr>
<td>Directory Search (DSFARCH)</td>
<td>121</td>
</tr>
<tr>
<td>Paragraph</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------</td>
<td>------</td>
</tr>
<tr>
<td>3.2.29</td>
<td></td>
</tr>
<tr>
<td>Activate Segment (ACT)</td>
<td>124</td>
</tr>
<tr>
<td>3.2.30</td>
<td></td>
</tr>
<tr>
<td>Deactivate Segment (DEACT)</td>
<td>129</td>
</tr>
<tr>
<td>3.2.31</td>
<td></td>
</tr>
<tr>
<td>Swap Segment In (SWAPIN)</td>
<td>132</td>
</tr>
<tr>
<td>3.2.32</td>
<td></td>
</tr>
<tr>
<td>Swap Segment Out (SWAPOUT)</td>
<td>136</td>
</tr>
<tr>
<td>3.2.33</td>
<td></td>
</tr>
<tr>
<td>Initialize Segment (INITSEG)</td>
<td>139</td>
</tr>
<tr>
<td>3.2.34</td>
<td></td>
</tr>
<tr>
<td>Prehash (PREHASH)</td>
<td>142</td>
</tr>
<tr>
<td>3.2.35</td>
<td></td>
</tr>
<tr>
<td>Hash (HASH)</td>
<td>145</td>
</tr>
<tr>
<td>3.2.36</td>
<td></td>
</tr>
<tr>
<td>Load Segment Descriptors (LSD)</td>
<td>148</td>
</tr>
<tr>
<td>3.2.37</td>
<td></td>
</tr>
<tr>
<td>Disk I/O (DISKIO)</td>
<td>151</td>
</tr>
<tr>
<td>3.2.38</td>
<td></td>
</tr>
<tr>
<td>Disk Allocation (DALLOC)</td>
<td>155</td>
</tr>
<tr>
<td>3.2.39</td>
<td></td>
</tr>
<tr>
<td>Free Disk (DFREE)</td>
<td>160</td>
</tr>
<tr>
<td>3.2.40</td>
<td></td>
</tr>
<tr>
<td>Sleep (SLEEP)</td>
<td>163</td>
</tr>
<tr>
<td>3.2.41</td>
<td></td>
</tr>
<tr>
<td>Run (RUN)</td>
<td>165</td>
</tr>
<tr>
<td>3.2.42</td>
<td></td>
</tr>
<tr>
<td>Swap (SWAP)</td>
<td>168</td>
</tr>
<tr>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td>Storage Allocation</td>
<td>177</td>
</tr>
<tr>
<td>3.3.1</td>
<td></td>
</tr>
<tr>
<td>Data Base Characteristics</td>
<td>177</td>
</tr>
<tr>
<td>3.3.2</td>
<td></td>
</tr>
<tr>
<td>Constants and Macros</td>
<td>198</td>
</tr>
<tr>
<td>3.4</td>
<td></td>
</tr>
<tr>
<td>Security Kernel Function Call Matrix</td>
<td>207</td>
</tr>
<tr>
<td>3.4.1</td>
<td></td>
</tr>
<tr>
<td>Program Interrupts</td>
<td>209</td>
</tr>
<tr>
<td>3.4.2</td>
<td></td>
</tr>
<tr>
<td>Subprogram Referencing</td>
<td>209</td>
</tr>
<tr>
<td>3.4.3</td>
<td></td>
</tr>
<tr>
<td>Special Control Features</td>
<td>221</td>
</tr>
<tr>
<td>4.</td>
<td></td>
</tr>
<tr>
<td>QUALITY ASSURANCE</td>
<td>222</td>
</tr>
<tr>
<td>4.1</td>
<td></td>
</tr>
<tr>
<td>Validation Criteria</td>
<td>222</td>
</tr>
<tr>
<td>4.1.1</td>
<td></td>
</tr>
<tr>
<td>Information Security Model</td>
<td>222</td>
</tr>
<tr>
<td>4.1.2</td>
<td></td>
</tr>
<tr>
<td>Validation Tests and Demonstrations</td>
<td>223</td>
</tr>
<tr>
<td>5.</td>
<td></td>
</tr>
<tr>
<td>PREPARATION FOR DELIVERY</td>
<td>226</td>
</tr>
<tr>
<td>5.1</td>
<td></td>
</tr>
<tr>
<td>Preparation of Useable Machine Language SKCPP</td>
<td>226</td>
</tr>
<tr>
<td>5.1.1</td>
<td></td>
</tr>
<tr>
<td>Protection of SKCPP Integrity</td>
<td>226</td>
</tr>
<tr>
<td>6.</td>
<td></td>
</tr>
<tr>
<td>NOTES</td>
<td>228</td>
</tr>
<tr>
<td>ADDENDA</td>
<td>271</td>
</tr>
</tbody>
</table>
LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dynamic Address Translation</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>Spaces</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>Intended Interpretations</td>
<td>24</td>
</tr>
<tr>
<td>3a</td>
<td>CPC Write-Up Form</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>Memory Map</td>
<td>178</td>
</tr>
<tr>
<td>5</td>
<td>Data Base Reference Matrix</td>
<td>180</td>
</tr>
<tr>
<td>6</td>
<td>Kernel Function Call Matrix</td>
<td>208</td>
</tr>
<tr>
<td>7</td>
<td>GATE Call Diagram</td>
<td>210</td>
</tr>
<tr>
<td>8</td>
<td>CREATE Call Diagram</td>
<td>211</td>
</tr>
<tr>
<td>9</td>
<td>DELETE Call Diagram</td>
<td>212</td>
</tr>
<tr>
<td>10</td>
<td>GIVE and RESCIND Call Diagram</td>
<td>213</td>
</tr>
<tr>
<td>11</td>
<td>GETW and GETR Call Diagram</td>
<td>214</td>
</tr>
<tr>
<td>12</td>
<td>DCONNECT Call Diagram</td>
<td>215</td>
</tr>
<tr>
<td>13</td>
<td>ENABLE Call Diagram</td>
<td>215</td>
</tr>
<tr>
<td>14</td>
<td>OUTER Call Diagram</td>
<td>216</td>
</tr>
<tr>
<td>15</td>
<td>OUTERV Call Diagram</td>
<td>216</td>
</tr>
<tr>
<td>16</td>
<td>IPCRCV Call Diagram</td>
<td>217</td>
</tr>
<tr>
<td>17</td>
<td>STARTP Call Diagram</td>
<td>217</td>
</tr>
<tr>
<td>18</td>
<td>STOPP Call Diagram</td>
<td>218</td>
</tr>
<tr>
<td>19</td>
<td>CHANGEO Call Diagram</td>
<td>218</td>
</tr>
<tr>
<td>20</td>
<td>INIT Call Diagram</td>
<td>219</td>
</tr>
<tr>
<td>21</td>
<td>READIR Call Diagram</td>
<td>219</td>
</tr>
<tr>
<td>22</td>
<td>SWAPIN Call Diagram</td>
<td>220</td>
</tr>
<tr>
<td>23</td>
<td>The Validation Chain</td>
<td>224</td>
</tr>
</tbody>
</table>

LIST OF TABLES

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>List of Constants</td>
<td>198</td>
</tr>
<tr>
<td>II</td>
<td>List of Macros</td>
<td>205</td>
</tr>
</tbody>
</table>
1. SCOPE

This specification establishes the requirements for complete identification of the ESD/MITRE Security Kernel Computer Program Product (SKCPP) for the PDP-11/45, henceforth to be referred to as the Security Kernel. The purpose of the specification is to present the information necessary to understand the Security Kernel for use, either for further investigation, or for implementation in a similar hardware environment.

The Security Kernel is the software, when installed and running properly on the PDP-11/45 hardware, that provides the facilities to control access of active system elements (subjects) to units of information (objects) within the computer system, thus providing the basis for secure computer systems on the PDP-11/45.
2. REFERENCED DOCUMENTS

The following documents, of exact issue shown, form a part of this specification to the extent specified herein. In the event of conflict between the documents referenced herein and the contents of this specification, the contents of this specification shall be considered a superseding requirement.

2.1 Government Documents


2.2 Non-Government Documents


3. REQUIREMENTS

The Security Kernel as described in this specification operates on a Digital Equipment Corporation PDP-11/45 with memory management unit with 128K bytes of core (reference 2.2b, e, and f) and a mass storage device in the form of an RF11 disk with 512K bytes available. The other peripherals associated with this PDP-11/45 are two Delta Data Systems Delta 5000 scopes (reference 2.2g), an ASR-33 teletype and a Digital Equipment Corporation DEWriter (reference 2.2e). Only minor modifications to the Security Kernel's code would be required to add, delete, or change the peripherals. The aforementioned storage, however, is considered to be the minimal requirement for effective execution.

The output medium used for transfer of the load module from the IBM 360 environment, where the Project SUE System and the PDP-11 Cross Assembler both execute, to the PDP-11/45 is a 9 track magnetic tape. The Digital Equipment Corporation TM11 magnetic tape system (reference 2.2e) is used to load the 9 track magnetic tape into the PDP-11/45.

A reference monitor is fundamental to a secure computer system. The reference monitor is that portion of a computer's hardware and software which enforces the authorized access relationships between subjects and objects. Subjects are entities such as a user or a process that seek to gain access to objects, while objects are entities such as data, programs, and peripheral devices to which access must be gained in the course of the system's use. The RF11 disk is the only DMA (direct memory access) device the Security Kernel handles. It has complete control over what is written and what is read since it has sole write access to the disk status register, which contains disk address, memory address, mode, and word count.

The memory management unit (MMU) option of the PDP-11/45 provides the hardware facilities that make it a suitable base for a secure system. The MMU checks all references to memory and supports enforcement of three access modes -- read/write, read, and no access protection. It also provides a hierarchy of three domains (or modes) of execution -- kernel, supervisor, and user. The hardware affects the hierarchical ordering of domains by permitting the execution of certain machine instructions in the kernel domain only, and restricting the manner in which the instructions which pass control from domain to domain operate. The MMU performs dynamic address translation; each time an effective address is generated during instruction execution, it is treated as a 16-bit virtual address and translated into an 18-bit physical address before
reference to main memory is made. Figure 1 portrays the dynamic address translation performed by the MMU. The translation is controlled by the contents of a set of eight segmentation registers, one set for each domain of execution.

The Security Kernel is the software portion of the reference monitor, the control and certification of which is essential to achieve a basic security module. It protects itself by insuring that its own procedures are the only ones that execute in kernel mode, and by executing interpretively all attempts to access the Security Kernel data base that originate with processes executing outside the kernel.

Interpretive execution, within the kernel, of access attempts permits objects accessed in the kernel domain to be portions of segments, whereas a directly accessed object outside the kernel must be coextensive with a single segment (reference 2.14); a segment is described as a collection of from 64 to 8194 bytes of contiguous virtual memory, which is uniquely identified, and which may be inserted into an appropriately sized and protected region of main memory.

The identification of a subject recognized by the Security Kernel is an ordered pair, whose components are a process and a domain. A process, in turn, is identified by the kernel as operating on behalf of a user/project pair. Information that describes processes is contained in data structures that are accessed in the kernel domain.

The Security Kernel software, operating in the PDP-11/45 with MMU, provides a non-random access, segmented virtual memory. The segmented virtual memory is organized into a tree-like directory hierarchy. The set of all segments in the hierarchy is the system space (SS). By virtue of security attributes and a security policy, most users of the system will not be able to access all of the segments in SS. The subset that a particular user may access is called a virtual space (VS). When a user process starts execution in a system that incorporates the Security Kernel, it has a virtual machine executing on its behalf. This process will have an address space of segments constrained in size by certain design and implementation parameters. The process's address space is called the working space (WS), and is always a subset of the user's VS. Finally, to permit a process direct access to all segments in its WS would severely constrain the size of the WS (to the number of descriptors available). For that reason another space, access space (AS), is added. AS, a subset of WS, represents the segments that a process can directly address because it has descriptors.
16 bit virtual address

\[ \begin{array}{cccc}
15 & 13 & 12 & 0 \\
\text{ASF} & \text{Displacement Field} & & \\
\end{array} \]

16 bit virtual address

\[ \begin{array}{cccc}
15 & 13 & 12 & 6 \ 5 \\
\text{ASF} & \text{Block Number} & \text{Word Number} & \\
\end{array} \]

active segment field selects a segmentation register

Segment Address Field

Full Adder

\[ \begin{array}{cccc}
17 & 6 & 5 & 0 \\
\text{Physical Block Number} & \text{Word Number} & \\
\end{array} \]

\[ \begin{array}{cccc}
17 & 0 \\
\text{Physical Address} & \\
\end{array} \]

Figure 1. Dynamic Address Translation
available. A process, therefore, may frequently remove a segment from AS to make room for another, without necessarily removing the segment from WS. Figure 2 shows these space relationships.

3.1 Functional Allocation Description

a. Control Entry and Exit functions accept and check parameters furnished by the user.

b. Directory Manipulation functions change the security state of the system by creating or deleting segments, and by adding or deleting elements in a segment’s control list.

c. Segment Accessing functions move segments into and out of a process’s working space.

d. Process Cooperation functions allow the sequential processes that coexist in the physical computer system to cooperate in performing computations.

e. Process Control functions select a particular process to which the CPU will be allocated, while saving information pertaining to other processes.

f. Privileged functions establish certain system conditions, and may only be invoked by trusted subjects.

Of the forty-two CPC's, eighteen are externally callable, that is, they are called by a process operating outside the perimeter of the Security Kernel (external to the kernel domain). These components are accessed by way of kernel commands, which include the symbolic name of the function along with the appropriate arguments.

The externally callable functions are listed below according to their functional uses.

Directory Manipulation

- create
- delete
- give
- rescind
- read directory

Segment Accessing

- get write
- get read
Figure 2. Spaces
. release
. enable
. disable
. outer P
. outer V
. send interprocess communication
. receive interprocess communication

Process Control
. stop process

Privileged
. start process
. change object
. initialize hierarchy

The remaining twenty-four CPC's are called once the kernel domain has been entered (internal to the kernel domain). They are called either directly or indirectly by the externally callable components and are invisible outside the Security Kernel.

The internally callable functions are listed below according to their functional uses.

Control Entry and Exit
. gate
. parameter checker

Directory Manipulation
. delete segment
. disk allocation
. free disk
. search out and destroy descriptors
. get directory
. write directory

Segments Accessing
. directory search
. connect
. activate
. deactivate
. prehash
. hash
. load segment descriptors
. swapin
. swapout
The kernel function gate (GATE) is the sole entry and exit point into and out of the kernel domain. When a user process requests an externally callable user level function, a trap is generated (reference 2.2f) and as a result of this trap GATE is entered. Upon entry, the supervisor's stack is accessed for parameter passing. The parameter checker (PCHECK) is then invoked by GATE and the parameters associated with the user requested function are checked to assure that they are within the acceptable ranges. If all parameters are within bounds PCHECK invokes the user requested function. Once the user requested function has performed its task, the return code (RC) is passed through PCHECK to GATE. GATE accesses the supervisor's stack, places the RC on the supervisor's stack and the kernel domain is exited. The only access route through the security perimeter into kernel domain is through GATE, which is also the only exit route.

The Security Kernel supports five processes which operate on behalf of the user (see Section 3.). The interrupt handlers for these I/O devices are handled within GATE. It saves the contents of the current process's registers; the PC (program counter) and PSW (process status word) (reference 2.2f) of the interrupt vector become the new process PC and PSW. The new current process is made available for servicing by a call to V which increments the semaphore on this process's I/O segment. The general purpose registers, the PC, and the PSW are restored to what they contained before the interrupt. For a more detailed description of GATE refer to paragraph 3.2.1.

The following subparagraphs provide more detail on the Security Kernel computer program components, the functional areas in which they fall, their symbolic code, and the major function they perform.
To avoid confusion an explanation of the release function is in order here. The symbolic code of the release function changes once the kernel domain has been entered. Externally (outside kernel domain) the function is known as RELEASE to match the mathematical model (reference 2.1a). Internally (inside kernel domain) the function is known as DCONNECT, as it is the inverse of the function CONNECT.

3.1.1 Entering and Exiting Kernel Domain

Two functions are provided to accept and check parameters furnished by the user. These functions are always invoked when an external call is made to the Security Kernel or upon exiting from kernel domain.

3.1.1.1 Accepting and Checking Parameters

3.1.1.1.1 Internal Functions

Gate GATE the entry and exit point into and out of the kernel.

Parameter Checker PCHECK assures that all user input parameters are within bounds.

3.1.2 Directory Manipulation Functions

A set of functions is provided for manipulating the attributes of segments. These functions change the security state of the system by creating and deleting segments, and by adding and deleting elements to/from a segment's access control list (ACL). The common security requirement for all functions that modify segment attributes is that the modifying process currently have write access to the segment's parent directory.

3.1.2.1 Creating and Deleting Segments

3.1.2.1.1 External Functions

Create CREATE creates a segment inferior to a specified directory segment.

Delete DELETE deletes an existing segment and any segments subordinate to it.
3.1.2.1.2 **Internal Functions**

- Delete Segment: **DELETSEG**
  - Deletes an empty directory or a data segment.

- Disk Allocation: **DALLOC**
  - Allocates space on the disk as segments are created.

- Free Disk: **DFREE**
  - Frees space on the disk as segments are deleted.

3.1.2.2 **Giving and Rescinding Access**

3.1.2.2.1 **External Functions**

- Give: **GIVE**
  - Adds an ACL element to a segment's ACL chain.

- Rescind: **RESCIND**
  - Removes an ACL element from a segment's ACL chain.

3.1.2.3 **Directory Support Functions**

3.1.2.3.1 **Internal Functions**

- Search Out and Destroy Descriptors: **SOADD**
  - Removes a segment from the address space of all processes that currently have access to that segment.

3.1.2.4 **Reading Directories**

3.1.2.4.1 **External Functions**

- Read Directory: **READIR**
  - Provides interpretive read access to an object that is in a process' address space.

3.1.2.5 **Directory Checking**

3.1.2.5.1 **Internal Functions**

- Get Directory: **GETDIR**
  - Assures that a directory segment to be written is in main memory and accessible.
Write Directory WRITEDIR checks that a segment is a directory and if so, that the current process has write access to it.

3.1.3 Segment Accessing

There are functions provided for moving segments into and out of a process's working space (WS) and functions that support the implementation of access space (AS). The function that changes a process's WS changes the state of the system with respect to security, whereas the functions that change AS are only changing the representation of the current security state. A process can directly address only those segments in its WS that are also in its AS because of hardware segmentation register constraints. WS is defined, for security purposes, to be the address space of a process.

3.1.3.1 Getting and Releasing Access

3.1.3.1.1 External Functions

- Get Write GETW moves a segment into a process's WS in write mode if all requirements are satisfied.
- Get Read GETR moves a segment into a process's WS in read mode if all requirements are satisfied.
- Release DCONNECT removes a segment from a process's WS (known externally as RELEASE, internally as DCONNECT).

3.1.3.1.2 Internal Functions

- Directory Search DSEARCH searches an ACL chain looking for an ACL element that applies to the invoking process.

3.1.3.2 WS Support Functions

3.1.3.2.1 Internal Functions

- Connect CONNECT connects a process to a segment in its WS.
Activate ACT copies the directory entry into an ASTE (active segment table entry) and initializes other fields in the ASTE.

Deactivate DEACT removes a segment from the AST (Active Segment Table).

Prehash PREHASH computes an index into the hash table.

Hash HASH converts a disk address (which uniquely identifies a segment) to an ASTE#, using a hash table.

### 3.1.3.3 Enabling and Disabling Access

#### 3.1.3.3.1 External Functions

*Enable ENABLE moves a segment in a process's WS into its AS.*

*Disable DISABLE removes a segment from a process's AS and frees a segmentation register.*

#### 3.1.3.3.2 Internal Functions

*Load Segment Descriptor LSD constructs segment descriptors and inserts them into descriptor registers.*

### 3.1.3.4 AS Support Functions

#### 3.1.3.4.1 Internal Functions

*Swap In SWAPIN swaps a segment into main memory.*

*Swap Out SWAPOUT removes a segment from main memory.*

*Initialize Segment INITSEG initializes a segment in memory.*

*Disk Input/Output DISKIO performs a disk I/O operation.*
3.1.4 Process Cooperation

Mechanisms are provided to allow the sequential processes that coexist in the physical computer system to cooperate in performing computations. These mechanisms are used within the Security Kernel to insure its correct operation. The Security Kernel provides functions that allow these mechanisms to be used in an arbitrary manner subject only to security constraints. These functions do not change the security states of the system. They provide interpretive access to objects as permitted by the current state, and since they can cause the execution state of a process to change, they modify the representation of the current state.

3.1.4.1 Synchronization Primitives

3.1.4.1.1 External Functions

| Outer P       | OUTERP |
| Outer V       | OUTERV |

when a P or V is performed externally, OUTERP and OUTERV perform implementation and security checks.

3.1.4.1.2 Internal Functions

| P and V | P and V |

allows users to coordinate the modification of shared segments and to synchronize with their terminal I/O.

P and V are the standard synchronization primitives.

3.1.4.2 Interprocess Communication

3.1.4.2.1 External Functions

| Send Interprocess Communication | IPCSEND |

allocates an IPC element, fills it in and adds it to the queue of elements waiting to be received.

| Receive Interprocess Communication | IPCRCV |

removes the data from an IPC element and puts the element on the free chain.
3.1.5 **Process Control**

A set of functions are provided for selecting a particular process to run, and for preparing and saving information pertaining to the process that currently has the CPU allocated to it and the next ready process.

3.1.5.1 **Creating and Deleting Processes**

3.1.5.1.1 **External Functions**

Stop Process  STOPP  relinquishes a user's ownership of a process.

3.1.5.1.2 **Internal Functions**

Sleep  SLEEP  schedules another process, in a round robin fashion.

Run  RUN  saves information on the current process and prepares to run the next process.

Swap  SWAP  establishes the next process's address space.

3.1.6 **Privileged Functions**

Three kernel functions are invoked by the executive process only.

3.1.6.1 **Trustworthy Process Functions**

3.1.6.1.1 **External Functions**

Change Object  CHANGEO  performs a change in classification of a non-directory segment.

Initialize Hierarchy  INITH  sets up the initial directory structure at initialization time.
Start Process     STARTP     initializes a new process.

3.2 Functional Description

This paragraph contains the detailed technical descriptions of the computer program components identified in paragraph 3.1 of this specification. The instruction listing contained in Section 10 specifies the exact configuration of the Security Kernel.

Most of the computer program components of the Security Kernel are written in Project SUE System Language (reference 2.2a). DALLOC, DFREE, DISKIO, LSD, and SWAP, which deal exclusively with disk space and hardware registers, are written in PAL-11, the PDP-11/45 assembler language (reference 2.2c).

The SUE language facilitates highly readable structured programs and data. One of its features is a macro facility which permits text substitution with parameters, but no compile-time computation. That is, the macro name is textually replaced by its macro body at compile time, thus eliminating repetitive coding and branching. One of the ways the Security Kernel makes use of this feature is in producing a more efficient and more exact calling sequence.

Each externally callable function has associated with it a macro whose name begins with 'K'; e.g., KCREATE is the macro associated with the CREATE function. These macros are located in the Context Block (reference 2.2a) NOFORN (Section 10, pages 2 through 10) which contains definitions relevant to all kernel users but is external to the security perimeter of the Security Kernel. The effect of these macros is to generate code which will place the parameters of the requested function on the supervisor's stack and force an interrupt. As a result of the interrupt a branch into the kernel domain is effected. An example of the calling sequence is as follows.

A user process requests the externally callable function CREATE by using the Kernel Command CREATE along with its appropriate parameters. The macro KCREATE generates code which places the input parameters on the supervisor's segmentation register 0 stack and forces an interrupt. As a result of this interrupt a branch into GATE is effected. GATE accesses the supervisor's stack through kernel segmentation register 3 (KSR3). The parameters, which are
now residing in KSR3, are passed to PCHECK for checking. If at this point the parameters are within the acceptable ranges, PCHECK invokes the CREATE function.

Many of the kernel functions set the value of a per process return code (RC). The security attributes of the RC object are equal to those of the process. In general, Security Kernel functions set RC to indicate whether or not they were called correctly. A few functions use RC to return additional information to the user. Each process can always observe its own and only its own RC object. Once the called function has performed its task, it passes the RC through PCHECK to GATE. GATE then performs the inverse of its entry sequence, that is, it gains write access to the supervisor's stack through KSR3 and places the RC on it.

Another feature of the Project SUE System Language (reference 2.2a) is the use of Inline. Inline inserts arbitrary machine code inline at compile time. The parameters are quantities that cause code to be placed in the machine instruction format.

Figure 3 shows the intended interpretation of the external kernel function parameters and internal kernel variables. TCP (the current process) is an internal Security Kernel variable that indicates which process is currently bound to the CPU. It is part of the mechanism for implementing a distributed kernel and prevents users of the Security Kernel from forging their identity.

The individual CPC's are described in the following 3.2 sub paragraphs. Figure 3a identifies the form in which each CPC is presented.
External Security Kernel function parameters

- **seg#**: segment number of a segment in a process's address space (WS)
- **offset**: identification of an entry within a directory
- **class**: a classification
- **cat**: a category set
- **type**: DATA or DIRECTORY
- **size**: size of a segment in blocks
- **mode**: WRITE, READ, or NO
- **user_id**: user identification
- **project_id**: project identification
- **reg#**: identification of a segmentation register
- **process#**: identification of a process
- **block#**: main memory address of a segment

Internal Security Kernel "variables"

- **TCP**: the current process
- **aste#**: pointer to an AST entry
- **daste#**: aste# for a segment known to be a directory
- **acle#**: pointer to an ACL element
- **smfr#**: pointer to a semaphore
- **ipce#**: pointer to an IPC element

Figure 3. Intended Interpretations
3.2.n CPC Title (symbolic code)

Identification of the level of the SKCPP function, the level of functions that call it, the level of functions that it calls and the language in which it is written.

3.2.n.1 Description

A description of the internal requirements of the CPC. A formal specification is also presented.

3.2.n.2 Flow Charts

Flow charts are presented for the PAL-11 CPCs exclusively as it is felt that the highly readable structure of the SUE language makes flow charting of those CPCs redundant.

3.2.n.3 Interfaces

The relationships of the CPCs to each other are presented here by listing those CPCs that call and those CPCs called by this specific CPC.

3.2.n.4 Data Organization

The relationships of the CPC to the data base structures are presented here. Global references, function parameters, local references, and constants are listed.

3.2.n.5 Limitations

Any known limitations of the CPC are presented here.

3.2.n.6 Listing

A complete listing of the Security Kernel is provided in Section 10; however, this subparagraph includes the DATA BLOCK and PROGRAM BLOCK of the specific CPC.

Figure 3a. CPC Write-Up Form
3.2.1 Gate (GATE)

The Gate CPC, GATE, is a kernel level internal SKCPP function that is invoked as the result of a user level program requesting an externally callable Security Kernel function. This CPC is unique as it is the sole entry and exit point into and out of the domain of the Security Kernel, that is, an externally callable function can only be reached through GATE and all return codes (if applicable) are passed to the user through GATE. GATE directly calls the kernel level internal function PCHECK, which in turn, directly calls the user requested function. The other function that GATE calls is the kernel level internal function V. It is written in Project SUE Language.

3.2.1.1 Description

Gate provides the only entry and exit point to and from the Security Kernel. It performs the KERNEL_ENTRY macro which pushes the supervisor's stack and registers onto the kernel's stack. If the logical and of PSW and PREV_MODE_MASK does not equal PREV_MODE_SUPERV, the call was not made from supervisor mode and GATE ignores it. Otherwise, the kernel must access the supervisor's stack through KSR3 to get its parameters, so it assigns to KSDR3 and KSAR3 the values of SDR0 and SAR0. Gate then calls PCHECK to check the parameters and to call the function requested, and sets RC to the value PCHECK returns. It then assigns to KSR3 and KRC the values of SR0 and RC, regaining access to the supervisor's stack and inserting the return code. Gate then invokes the KERNEL_EXIT macro to restore supervisor registers R0 to R6 from its stack and return. Gate handles interrupts by invoking KERNEL ENTRY to save the contents of the current process's registers; the PS and PSW of the interrupt vector become the new process's PC and PSW. Gate calls V to increment the semaphore on the new current process, so it can be serviced. KERNEL EXIT is then invoked, restoring the general purpose registers, the PC, and the PSW with what they contained before the interrupt.

Function: GATE
Parameters: GATE(function_code,seg#,offset,class,cat,type,
size,mode,user_id,project_id,reg#,process#,block#)
Effect:
IF PSW & PREV_MODE_MASK = PREV_MODE_SUPERV;
THEN: PCHECK(function_code,seg#,offset,class,cat,type,size,
mode,user_id,project_id,reg#,process#,block#);
END;
3.2.1.2 N/A

3.2.1.3 Interfaces

As a direct result of a user level external program requesting an externally callable Security Kernel function, the macro associated with the requested function is invoked. The macro generates code which places the user entered parameters on the supervisor's stack and causes an interrupt. As a direct result of this interrupt, entry into the Security Kernel domain is reached.

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>see above paragraph</td>
<td>PCHECK</td>
</tr>
<tr>
<td></td>
<td>V</td>
</tr>
</tbody>
</table>

3.2.1.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function GATE. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

**Data Base References**

<table>
<thead>
<tr>
<th>Global References</th>
<th>Functional Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDR0</td>
<td>FUNCTION_CODE</td>
<td>KSDR3</td>
</tr>
<tr>
<td>SAR0</td>
<td>SEGA</td>
<td>KSAR3</td>
</tr>
<tr>
<td>PSW</td>
<td>OFFSET</td>
<td>RC</td>
</tr>
<tr>
<td></td>
<td>CLASS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TYPE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIZE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MODE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>USER_ID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PROJECT_ID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REG#</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PROCESS#</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BLOCK#</td>
<td></td>
</tr>
</tbody>
</table>

**Constants**

- DECW_PROCESS#
- DISK_SMFR
- PREV_MODE_MASK
A short discussion of the two Context Blocks CONTEXT NOFORN and CONTEXT KERNEL seems appropriate at this point. The Context Block NOFORN contains definitions relevant to all kernel users and the Security Kernel supportive software. The Context Block KERNEL contains definitions of the Security Kernel's virtual address space. These, of course, reside outside the security perimeter of the Security Kernel. Refer to Section 10 for listing of both Context Blocks.

DATA GATE contains the procedure declarations for all externally callable functions and all but six of the internally callable functions. Those six, DELETSEG, DALLOC, DFREE, SETPIR, INITSEG, and SWAP, are called by only one function and are declared in the Data Block of that function.

3.2.1.5 Limitations

Entry into kernel domain is only affected if the call is made from supervisor domain. The return code (RC) is that which the requested function has passed through PCHECK.

3.2.1.6 Listing

DATA GATE;

/*
* MUST CHECK INITIALIZATION OF INTERRUPT VECTORS IN STARTUP EVERY TIME A CHANGE IS MADE TO GATE (DATA OR PROGRAM)!
*/

MACRO KERNEL_ENTRY;
  INLINE(MFPIR6);
  INLINE(MOV, 0, 5, 4, 6);
  INLINE(MOV, 0, 4, 4, 6);
  INLINE(MOV, 0, 3, 4, 6);
  INLINE(MOV, 0, 2, 4, 6);
  INLINE(MOV, 0, 1, 4, 6);
  INLINE(MOV, 0, 0, 4, 6);
  INLINE(MOV, 0, 6, 0, 4);
  INLINE(MOV, 0, 5, 0, 4);
  INLINE(MOV, 0, 4, 6, "800A");
  INLINE(SUB, 2, 7, 0, B, "800A");
END MACRO;

MACRO KERNEL_EXIT;
  INLINE(ADD, 2, 7, 0, 6, 6);
  INLINE(MOV, 2, 6, 0, 0);
  INLINE(MOV, 2, 6, 0, 1);
  INLINE(MOV, 2, 6, 0, 2);
  INLINE(MOV, 2, 6, 0, 3);
  INLINE(MOV, 2, 6, 0, 4);
  INLINE(MOV, 2, 6, 0, 5);
  INLINE(MTPIR6);
  INLINE(RTI)
END MACRO;
DECLARE
    WORD (RC);

/* EXTERNAL KERNEL FUNCTIONS */

DECLARE
    PROCEDURE (STOPP),
    PROCEDURE ACCEPTS (WORD) (DISABLE),
    PROCEDURE ACCEPTS (WORD, WORD) (DCONNECT),
    PROCEDURE ACCEPTS (WORD, WORD, WORD) (IPCSEND),
    PROCEDURE RETURNS (WORD) (IPCRCV),
    PROCEDURE ACCEPTS (WORD) RETURNS (WORD) (OUTERP, OUTEBV),
    PROCEDURE ACCEPTS (WORD, WORD) RETURNS (WORD) (DELETE, GETW, GETR, ENABLE, READIR),
    PROCEDURE ACCEPTS (WORD, WORD, WORD) RETURNS (WORD) (IPCSEND, CHANGE),
    PROCEDURE ACCEPTS (WORD, WORD, WORD, WORD) RETURNS (WORD) (GPC),
    PROCEDURE ACCEPTS (WORD, WORD, WORD, WORD, WORD) RETURNS (WORD) (CREATE, STARTF);

/* INTERNAL KERNEL FUNCTIONS */

DECLARE
    PROCEDURE (SLEEP),
    PROCEDURE ACCEPTS (WORD) (DEACT, P, V, SWAPIN, SWAPOUT, RUN),
    PROCEDURE ACCEPTS (WORD, WORD) (SOADD),
    PROCEDURE ACCEPTS (WORD, WORDER, WORD) (LSD),
    PROCEDURE ACCEPTS (WORD, WORD, WORD, WORD) (DISKIO),
    PROCEDURE RETURNS (WORD) (PCHECK),
    PROCEDURE ACCEPTS (WORD) RETURNS (WORD) (WRITEDIR, HASH, PREHASH),
    PROCEDURE ACCEPTS (WORD, WORD, WORD) RETURNS (WORD) (DSEARCH, CONNECT),
    PROCEDURE ACCEPTS (WORD, WORD, WORD, WORD, WORD) RETURNS (WORD) (GIVE),
    PROCEDURE ACCEPTS (WORD, WORD, WORD, WORD, WORD, WORD) RETURNS (WORD) (CREATE, STARTF);

PROGRAM GATE;
    /* PUSH SUPERVISOR R6 AND W0-W5 ONTO KERNEL'S STACK */
    KERNEL_ENTRY;
    /* IGNORE CALL IF NOT MADE FROM SUPERVISOR MODE */
    IF (PSW & PREV_MODE_MASK) = PREV_MODE_SUPERV;
    THEN:
        /* ACCESS SUPERVISOR'S STACK THROUGH KSR3 */
        KSR3 := SR0;
        KSAR3 := SAR0;
        /* CALL PARAMETER CHECKER - WHO IN TURN CALLS THE REQUESTED FUNCTION */
        RC := PCHECK;
        /* REGAIN ACCESS TO SUPERVISOR'S SHO STACK AND INSERT RETURN CODE */
        KSR3 := SR0;
        KSAR3 := SAR0;
        KRC := RC;
    END;
    /* RESTORE SUPERVISOR'S REGISTERS AND RETURN */
    KERNEL_EXIT;
    /* INTERRUPT ENTRY POINTS */
    /* DISK */
3.2.2 Parameter Checker (PCHECK)

The Parameter Checker CPC, PCHECK, is a kernel level internal SKCPP function that is called by only one other kernel level internal function, GATE. PCHECK calls all of the user level external functions as well as other kernel level internal functions. It is written in Project SUE System Language.

3.2.2.1 Description

PCHECK calls an externally reachable function if all input parameters are within acceptable ranges and if the seg# parameter (if required) identifies a segment in the process’s WS.

It checks the function code: if FUNCTION_CODE_APARM is less than FUNCTION_CODE_MIN or greater than FUNCTION_CODE_MAX, PCHECK returns the SEVERE_FLAG. Otherwise, it sets FUNCTION_CODE equal to FUNCTION_CODE_APARM. It then performs a P on the kernel semaphore to block the kernel until after the function is performed, preventing more than one process from occupying the kernel at a time. PARM_FLAGS is assigned FUNCTION_ARRAY(FUNCTION_CODE), which indicates which parameters the function takes, and RC is set to OK_FLAG.
If the seg# parameter is required, PCHECK tests if the parameter supplied lies within the accepted range, as follows: if the logical and of PARM_FLAGS and SEG#_FLAG is not equal to zero, the specified function requires the seg# parameter. SEG#_PARM is set equal to SEG#_APARM. Then, if SEG#_PARM is less than SEG#_MIN or greater than SEG#_MAX, RC is set to SEVERE_FLAG.

PCHECK also checks that the seg# identifies a segment in the process's WS if the seg# parameter is required. If RC is not SEVERE_FLAG, ASTE#_PARM is assigned the value, PS_SEG(SEG#_PARM). Now, if the logical and of ASTE#/PARM and SEG_FLAG is not equal to zero, PS_SEG(SEG#_PARM) held a seg#. Therefore, it must have been on the free segment# chain. Hence, the segment specified was not in the WS of the process, so RC is set to zero. Otherwise, PS_SEG(SEG#_PARM) held an ast# and RC remains set to OK_FLAG.

In a similar manner, if the offset, classification, mode, reg#, and process# parameters are required, PCHECK checks that the values supplied lie within their acceptable ranges. Then, to CAT_PARM, SEG_TYPE_PARM, SIZE_PARM, USER_PARM, PROJECT_PARM, and MESSAGE_PARM, it assigns the respective values of CAT_APARM, SEG_TYPE_APARM logical and DIR_TYPE_MASK, SIZE_APARM, USER_APARM logical and ACL_USER_MASK, PROJECT_APARM, and MESSAGE_APARM.

If RC is not SEVERE_FLAG, PCHECK uses a case selector to call the function specified by FUNCTION_CODE and set RC to the value returned. To prevent compile-time parse stack overflow due to an implementation quirk, the case selection is broken up into two cases, FUNCTION_CODE between 1 and 9 and FUNCTION_CODE between 10 and 20, selected by an IF statement.

PCHECK then calls V to increment the kernel semaphore, and returns RC.

There are two short functions that are incorporated into the case selector under their FUNCTION_CODE tags. They are the T function whose FUNCTION_CODE tag is 12 and the PROCID (process identification) function whose FUNCTION_CODE tag is 18. Both have macros, in context NOFORN, associated with them.

The T function, which is user callable, is an inquiry about the semaphore count which does not require write access. If SMFR_COUNT (ASTE_PARM) is less than zero, T sets RC to ERR_FLAG.

The PROCID function merely assigns RC the value of PS_CURRENT_PROCESS. The process can use this to index the process directory directory and find its process directory or to index the I/O segment.
Function: PCHECK
Parameters: PCHECK(function_code, seg#, offset, class, cat, type, size, mode, user_id, project_id, reg#, process#, block#)

Effect:
IF (FUNCTION_CODE_MIN <= function_code <= FUNCTION_CODE_MAX &
(not SEG#_PARM(function_code) |
((SEG#_MIN <= seg# <= SEG#_MAX) &
PS_SEG_INUSE(TCP, seg#))) &
(not OFFSET_PARM(function_code) |
(OFFSET_MIN <= offset <= OFFSET_MAX) &
(not CLASS_PARM(function_code) |
(CLASS_MIN <= class <= CLASS_MAX) &
(not CAT_PARM(function_code) |
(cat ∈ CATEGORY_SET)) &
(not TYPE_PARM(function_code) |
((type = DIRECTORY) | (type = DATA)) &
(not SIZE_PARM(function_code) |
(SIZE_MIN <= size <= SIZE_MAX) &
(not MODE_PARM(function_code) |
((mode = WRITE) | (mode = READ) | (mode = NO)) &
(not USER_ID_PARM(function_code) |
(USER_ID_MIN <= user_id <= USER_ID_MAX)) &
(not PROJECT_ID_PARM(function_code) |
(PROJECT_ID_MIN <= project_id <= PROJECT_ID_MAX)) &
(not REG#_PARM(function_code) |
(REG#_MIN <= reg# <= REG#_MAX)) &
(not PROCESS#_PARM(function_code) |
(PROCESS#_MIN <= process# <= PROCESS#_MAX)) &
(not BLOCK#_PARM(function_code) |
(BLOCK#_MIN <= block # <= BLOCK#_MAX));
THEN: Let aste# = PS_SEG(TCP, seg#);
CASE of function_code:
  1. CREATE(TCP, aste#, offset, class, cat, type, size);
  2. DELETE(TCP, aste#, offset);
  3. GIVE(TCP, aste#, offset, mode, user_id, project_id);
  4. RESCIND(TCP, aste#, offset, user_id, project_id);
  5. GETW(TCP, aste#, offset);
  6. GETR(TCP, aste#, offset);
  7. DCONNECT(seg#, aste#);
  8. ENABLE(TCP, aste#, reg#);
  9. DISABLE(reg#);
10. OUTERP(aste#);
11. OUTERV(aste#);
12. T(TCP, aste#);
13. IPCSEND(process#, message, USER_DOMAIN);
14. IPCRCV;
15. STARTP(TCP, user_id, project_id, class, cat, process#, offset);
16. STOPP;
17. CHANGEO(TCP, aste#, offset, class, cat);
18. KPROCID(TCP);
19. INITH(TCP, aste#, offset, cat);
20. READIR(TCP, aste#, offset);

ELSE:  RC(TCP) = NO
END;

3.2.2.2 N/A

3.2.2.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>GATE</td>
<td>CREATE</td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
</tr>
<tr>
<td></td>
<td>GIVE</td>
</tr>
<tr>
<td></td>
<td>RESCIND</td>
</tr>
<tr>
<td></td>
<td>OUTERP</td>
</tr>
<tr>
<td></td>
<td>OUTERV</td>
</tr>
<tr>
<td></td>
<td>STARTP</td>
</tr>
<tr>
<td></td>
<td>STOPP</td>
</tr>
<tr>
<td></td>
<td>CHANGEO</td>
</tr>
<tr>
<td></td>
<td>INITH</td>
</tr>
<tr>
<td></td>
<td>READIR</td>
</tr>
<tr>
<td></td>
<td>IPCRCV</td>
</tr>
<tr>
<td></td>
<td>IPCSEND</td>
</tr>
<tr>
<td></td>
<td>GETW</td>
</tr>
<tr>
<td></td>
<td>GETR</td>
</tr>
<tr>
<td></td>
<td>ENABLE</td>
</tr>
<tr>
<td></td>
<td>DISABLE</td>
</tr>
<tr>
<td></td>
<td>DCONNECT</td>
</tr>
<tr>
<td></td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>V</td>
</tr>
</tbody>
</table>

3.2.2.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function PCHECK. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1.
For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

**Data Base References**

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS_SEG</td>
<td>FUNCTION_CODE</td>
<td>SEG#_PARAM</td>
</tr>
<tr>
<td>PS_CURRENT_PROCESS</td>
<td></td>
<td>OFFSET_PARM</td>
</tr>
<tr>
<td>SEG#_APARM</td>
<td>SEG#</td>
<td>CLASS_PARM</td>
</tr>
<tr>
<td>OFFSET_APARM</td>
<td>OFFSET</td>
<td>CAT_PARM</td>
</tr>
<tr>
<td>CLASS_APARM</td>
<td>CLASS</td>
<td>TYPE_PARM</td>
</tr>
<tr>
<td>CAT_APARM</td>
<td>CAT</td>
<td>SEG_TYPE_PARM</td>
</tr>
<tr>
<td>SEG_TYPE_APARM</td>
<td>SIZE</td>
<td>SIZE_PARM</td>
</tr>
<tr>
<td>SIZE_APARM</td>
<td>MODE</td>
<td>MODE_PARM</td>
</tr>
<tr>
<td>MODE_APARM</td>
<td>USER_ID</td>
<td>USER_PARM</td>
</tr>
<tr>
<td>USER_APARM</td>
<td>PROJECT_ID</td>
<td>PROJECT_PARM</td>
</tr>
<tr>
<td>PROJECT_APARM</td>
<td>REG#</td>
<td>REG#_PARAM</td>
</tr>
<tr>
<td>REG#_APARM</td>
<td>PROCESS#</td>
<td>PROCESS#_PARAM</td>
</tr>
<tr>
<td>PROCESS#_APARM</td>
<td>BLOCK#</td>
<td>MESSAGE_PARM</td>
</tr>
<tr>
<td>MESSAGE_APARM</td>
<td></td>
<td>ASTE#_PARAM</td>
</tr>
<tr>
<td>FUNCTION_CODE_APARM</td>
<td></td>
<td>PARM_FLAGS</td>
</tr>
<tr>
<td>FUNCTION_CODE</td>
<td></td>
<td>FUNCTION_CODE_TYPE</td>
</tr>
<tr>
<td>FUNCTION_ARRAY</td>
<td></td>
<td>RC</td>
</tr>
</tbody>
</table>

**Constants**

- ACL_USER_MASK
- CLASS_FLAG
- CLASS_MAX
- CLASS_MIN
- DIR_TYPE_MASK
- ERR_FLAG
- FUNCTION_CODE_MAX
- FUNCTION_CODE_MIN
- KERNEL_SMFR
- MODE_FLAG
- NO_ACCESS
- OFFSET_FLAG
- OFFSET_MAX
- OFFSET_MIN
- OK_FLAG
- PROCESS#_FLAG
- PROCESS#_MAX
- PROCESS#_MIN
- READ$EXECUTE_ACCESS
- REG_FLAG
- REG#_MAX
- REG#_MIN
- SEG#_FLAG
- SEG#_MAX
- SEG#_MIN
- SEVERE_FLAG
- WRITE$READ$EXECUTE_ACCESS
3.2.2.5 Limitations

PCHECK returns SEVERE_FLAG if the parameters passed to it do not fall within the acceptable ranges or if the seg# (if required) is not in the process's WS.

3.2.2.6 Listing

DATA PCHECK RETURNS (RC):

PROGRAM PCHECK;

TYPE FUNCTION_CODE_TYPE = (0 TO FUNCTION_CODE_MAX);
DECLARE FUNCTION_CODE_TYPE (FUNCTION_CODE);
DECLARE
  WORD (SEG#_PARM, OFFSET_PARM, CLASS_PARM, CAT_PARM, SEG_TYPE_PARM, SIZE_PARM,
        MODE_PARM, USER_PARM, PROJECT_PARM, REG#_PARM, PROCESS#_PARM,
        MESSAGE_PARM, ASTE#_PARM, PARM_FLAGS);

  /* CHECK FUNCTION CODE */
  IF FUNCTION_CODE_APARM < FUNCTION_CODE_MIN;
    THEN:
      RETURN WITH SEVERE_FLAG;
    END;

  IF FUNCTION_CODE_APARM > FUNCTION_CODE_MAX;
    THEN:
      RETURN WITH SEVERE_FLAG;
    END;

  FUNCTION_CODE := FUNCTION_CODE_APARM;
  P(KERNEL_SMFR);
  PARM_FLAGS := FUNCTION_ARRAY(FUNCTION_CODE);
  RC := OK_FLAG;

  /* CHECK SEG# PARAMETER IF REQUIRED */
  IF (PARM_FLAGS & SEG#_FLAG) ^= 0;
    THEN: SEG#_PARM := SEG#_APARM;
      IF SEG#_PARM < SEG#_MIN;
        THEN: RC := SEVERE_FLAG;
      END;

      IF SEG#_PARM > SEG#_MAX;
        THEN: RC := SEVERE_FLAG;
      END;

    IF RC ^= SEVERE_FLAG;
      THEN: ASTE#_PARM := PS_SEG(SEG#_PARM);
        IF (ASTE#_PARM & SEG_FLAG) ^= 0;
          THEN: RC := SEVERE_FLAG;
        END;

      END;

END;
/* CHECK OFFSET PARAM IF REQUIRED */

if (parm_flags & offset_flag) ^= 0;
then: offset_parm := offset_aparm;

if offset_parm < offset_min;
  then: rc := severe_flag;
  end;

if offset_parm > offset_max;
  then: rc := severe_flag;
  end;
end;

/* CHECK CLASSIFICATION PARAMETER IF REQUIRED */

if (parm_flags & class_flag) ^= 0;
then: class_parm := class_aparm;

if class_parm < class_min;
  then: rc := severe_flag;
  end;

if class_parm > class_max;
  then: rc := severe_flag;
  end;
end;

/* NO SYNTACTIC ERROR CHECKING FOR CATEGORY PARAMETER */

cat_parm := cat_aparm;

/* NO CHECKING OF SEG_TYPE PARAMETER */

seg_type_parm := (seg_type_aparm & disp_type_mask);

/* NO CHECKING OF SIZE PARAMETER */

size_parm := size_aparm;

/* CHECK MODE PARAM IF REQUIRED */

if (parm_flags & mode_flag) ^= 0;
then: mode_parm := mode_aparm;

if (mode_parm ^= write$read$execute_access) & (mode_parm ^= read$execute_access)
  & (mode_parm ^= no_access);
  then: rc := severe_flag;
  end;
end;

/* NO CHECKING OF USER PARAMETER */

user_parm := (user_aparm & acl_user_mask);

/* NO CHECKING OF PROJECT PARAMETER */
PROJECT_PARM := PROJECT_APARM;

/* CHECK REG# IF REQUIRED */
IF (PARAM_FLAGS & REG_FLAG) == 0;
THEN: REG#_PARm := REG#_APARM;

IF REG#_PARm < REG#_MIN;
THEN: RC := SEVERE_FLAG;
END;

IF REG#_PARm > REG#_MAX;
THEN: RC := SEVERE_FLAG;
END;
END;

/* CHECK PROCESS# PARAMETER IF REQUIRED */
IF (PARAM_FLAGS & PROCESS#_FLAG) == 0;
THEN: PROCESS#_PARAM := PROCESS#_APARM;

IF PROCESS#_PARm < PROCESS#_MIN;
THEN: PC := SEVERE_FLAG;
END;

IF PROCESS#_PARm > PROCESS#_MAX;
THEN: RC := SEVERE_FLAG;
END;
END;

/* NO CHECKING OF MESSAGE PARAMETER */
MESSAGE_PARM := MESSAGE_APARM;

IF RC == SEVERE_FLAG;
THEN:

IF FUNCTION_CODE < 10;
THEN:

CASE FUNCTION_CODE:
  1: RC := CREATE(ASTM#_PARm, OFFSET_PARM, CLASS_PARn, CAT_PARM,
                     SEG_TYPE_PARM, SIZE_PARm);
  2: RC := DELETE(ASTM#_PARm, OFFSET_PARM);
  3: RC := GIVE(ASTM#_PARm, OFFSET_PARM, Mode_PARM, USER_PARM,
                  PROJECT_PARM);
  4: RC := RESCIND(ASTM#_PARm, OFFSET_PARM, USER_PARM, PROJECT_PARM);
  5: RC := GETW(ASTM#_PARm, OFFSET_PARM);
  6: RC := GETR(ASTM#_PARm, OFFSET_PARM);
  7: DCONNECT(SEG#_PARm, ASTM#_PARm);
  8: RC := ENABLE(ASTM#_PARm, REG#_PARm);
  9: DISABLE(REG#_PARm);
END;
3.2.3 Create Segment (CREATE)

The Create Segment CPC, CREATE, is a user level external SKCPP function that is called by user level external programs with the parameters seg#, offset, class, cat, type, and size. CREATE calls only kernel level internal functions. It is written in Project SUE System Language.

3.2.3.1 Description

CREATE creates a segment if security and implementation requirements are met. It calls WRITEDIR which sets RC to OK_FLAG if the intended parent segment is a directory to which the process has write access; otherwise, WRITEDIR sets ERR_FLAG. CREATE then sets RC to ERR_FLAG if the classification of the new segment is less than that of the parent segment, if the category set of the new segment does not include that of the parent segment, if the size of the new segment is not allowable, or if the offset does not identify a free directory entry. If RC is not set to ERR_FLAG, DALLOC allocates disk space for the segment with address DISK_ADR. DIR_TYPE is assigned SEG-type or DIR_UNINITIALIZED or CLASS, DIR_CAT is assigned CAT, DIR_DISK is assigned DISK_ADR, and DIR_SIZE is filled in with SIZE.
Function: CREATE
Parameters: CREATE(process#, aste#, offset, class, cat, type, size)

Effect:
IF not AST_WAL(aste#, process#) |
(class < AST_CLASS(aste#)) |
(aste # AST_CAT(aste#)) |
(aste # DIRECTORY) |
('DIR_SIZE'(aste#, offset) # 0) |
(size # SIZE_SET) |
((type = DIRECTORY) & (size # DIRECTORY_SIZE))
THEN: RC(process#) = NO;
ELSE: DIR_TYPE(aste#, offset) = type;
   DIR_STATUS(aste#, offset) = UNINITIALIZED;
   DIR_CLASS(aste#, offset) = class;
   DIR_CAT(aste#, offset) = cat;
   DIR_SIZE(aste#, offset) = size;
   DISK_ALLOC(size);
   DIR_DISK(aste#, offset) = NEXT_DISK_ADDRESS;
   DIR_ACL_HEAD(aste#, offset) = 0;
   RC(process#) = YES;
END;

3.2.3.2 N/A

3.2.3.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

Called By          Calls

PCHECK            WRITEDIR
                 DALLOC

3.2.3.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function CREATE. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1 For constants refer to Table I, List of Constants, in subparagraph 3.3.2.
3.2.3.5 Limitations

CREATE returns with ERR_FLAG if the intended parent segment is a directory to which the process does not have write access, if the classification of the new segment is less than that of the parent segment, if the category set of the new segment does not include that of the parent segment, if the size of the new segment is not allowable, or if the offset does not identify a free directory entry.

3.2.3.6 Listing

DATA CREATF(ASTE#, OFFSET, CLASS, CAT, SEG_TYPE, SIZE) RETURNS (RC);
DECLARE
PROCEDURE ACCEPTS (WORD) RETURNS (WORD) (DALLOC);

PROGRAM CREATF;
DECLARE
WORD (DISK_ADR);
/* SECURITY CHECKS FIRST * /
/* CREATF IS INTERPRETATIVE DIRECTORY WRITE */
RC := WRITEDIR (ASTE#);
/* CHECK COMPATIBILITY REQUIREMENTS */
IF CLASS < (AST_CLASS(ASTE#) & AST_CLASS_MASK);
THEN: RC := ERR_FLAG;
END;
3.2.4 Delete (DELETE)

The Delete CPC, DELETE, is a user level external SKCPP function that is called by user level external programs with the parameters seg# and offset. DELETE calls only kernel level internal functions. It is written in Project SUE System Language.

3.2.4.1 Description

DELETE removes all ACL elements from the parent directory entry identified by ASTE#, OFFSET, deactivates the specified segment, frees its disk space, and marks the parent directory entry available. If WRITEDIR returns ERR_FLAG indicating that the parent segment is not a directory to which the calling process has write access or if the entry to be deleted is of size zero, DELETE returns with ERR_FLAG. If not, DELETE calculates the logical and of DIR_TYPE and DIR_TYPE_MASK to separate type information from the status and class. If the result is not DIR_TYPE_DIRECTORY, the entry is a data segment; DELETSEG is called to remove all ACL elements, deactivate the segment, free its disk space, and set DIR_SIZE = Ø which marks the
entry available. If the result equals DIR_TYPE_DIRECTORY, DELETSEG cannot be called until the directory is empty.

DELETE then begins the directory deletion cycle: SPASTE# and SOFFSET are assigned the values of the ASTE# and OFFSET of the original directory to be deleted.

The process to find and delete the lowest directory in the hierarchy starting with the specified directory then follows: DELETE finds the disk address of the directory specified by SPASTE#, SOFFSET. It then calls HASH, which returns the directory's own ASTE# if it is active, or Ø if it is not. If it is Ø, ACT is called to activate the directory and SASTE# is set to the ASTE# of the directory to be deleted. Its segment descriptors are then loaded by GETDIR. DELETE then scans the entries of the directory to be deleted starting with TOFFSET equal to OFFSET_MIN. If the size of the entry identified by TOFFSET is zero, the scan of directory entries is continued. Otherwise, DELETE tests if the DIR_TYPE of the segment and DIR_TYPE_MASK equals DIR_TYPE_DIRECTORY. If so, this entry is itself another directory whose entries must be deleted before it can be deleted. In this case, DELETE resets SPASTE# and SOFFSET to SASTE# and TOFFSET and repeats the directory deletion process. If not, the entry is a data segment which DELETSEG deletes. The entry scan continues until TOFFSET equals OFFSET_MAX. Then, the directory is empty and DELETE can call DELETSEG to remove it. If SPASTE# is not equal to ASTE#, the last directory deleted was part of the sub-hierarchy of the directory originally specified to be deleted. DELETE loads the segment descriptors of the original directory and continues the directory deletion cycle. If SPASTE# = ASTE#, the last directory deleted was the original segment; DELETE sets RC to OK_FLAG and returns control to the calling program.

Function: DELETE
Parameters: DELETE(process#, aste#, offset)
Effect:
IF not AST_WAL(aste#, process#) |
   (AST_TYPE(aste#) ≠ DIRECTORY) |
   ('DIR_SIZE'(aste#, offset) = 0);
THEN: RC(process#) = NO;
ELSE:
   IF 'DIR_ACL_HEAD'(aste#, offset) ≠ 0;
   THEN: Let acle# = FINDEND(aste#, 'DIR_ACL_HEAD'(aste#, offset));
         ACL_CHAIN(aste#, acle#) = 'ACL_CHAIN'(aste#, 0);
         ACL_CHAIN(aste#, 0) = 'DIR_ACL_HEAD'(aste#, offset);
   END;
DIR_SIZE(aste#, offset) = 0;
DELETSEG(aste#, offset)
IF DIR_TYPE(aste#, offset) = DIRECTORY;
THEN: DELETSEG(aste#, offset)
END;
RC(process#) = YES;
END;

3.2.4.2 N/A

3.2.4.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
</table>
| PCHECK    | GETDIR
|           | DELETSEG
|           | WRITEDIR
|           | HASH
|           | ACT

3.2.4.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function DELETE. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIR_SIZE</td>
<td>ASTE#</td>
<td>SPASTE#</td>
</tr>
<tr>
<td>DIR_TYPE</td>
<td>OFFSET</td>
<td>SASTE#</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SOFFSET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TOFFSET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RC</td>
</tr>
</tbody>
</table>

Constants

| DIR_TYPE
| DIR_TYPE_DIRECTORY
| DIR_TYPE_MASK
| OK_FLAG
3.2.4.5 Limitations

DELETE returns ERR_FLAG if the parent segment is not a directory to which the calling process has access or if the entry to be deleted is of size zero. Otherwise, RC = OK_FLAG.

3.2.4.6 Listing

```
DATA DELETE(ASTE#, OFFSET) RETURNS (RC);
DECLARE
  PROCEDURE ACCEPTS (WORD) (GETDIR);
  PROCEDURE ACCEPTS (WORD, WORD) (DELETSEG);

PROGRAM DELETE;
DECLARE
  WORD (SPASTE#, SASTE#, SOFFSET, TOFFSET);
  /* SECURITY CHECKS FIRST */
  /* DELETE IS AN INTERPRETATIVE DIRECTORY WRITE */
  IF WRITEDIR(ASTE#) = ERR_FLAG;
  THEN:
    RETURN WITH ERR_FLAG;
  END;
  /* IMPLEMENTATION CHECKS */
  IF DIR_SIZE(OFFSET) = 0;
  THEN:
    RETURN WITH ERR_FLAG;
  END;
  /* ELIMINATE CASE OF DATA SEGMENT */
  IF (DIR_TYPE(OFFSET) & DIR_TYPE_MASK) != DIR_TYPE_DIRECTORY;
  THEN: DELETSEG(ASTE#, OFFSET);
  ELSE: <OUTER_CYCLE>
    CYCLE
    /* START AT THE TOP OF THE CHAIN AND WORK DOWN */
    SPASTE# := ASTE#;
    SOFFSET := OFFSET;
    <INNER_CYCLE>
    CYCLE
    /* ASSURE DIRECTORY TO BE DELETED IS ACTIVE */
    SASTE# := HASH(DIR_DISK(SOFFSET));
    IF SASTE# = 0;
      THEN: SASTE# := ACT(SPASTE#, SOFFSET);
    END;
    /* LOAD SEGMENT DESCRIPTORS */
    GETDIR(SASTE#);
  END;
```
3.2.5 Give Access (GIVE)

The Give Access CPC, GIVE, is a user level external SKCPP function that is called by user level external programs with the parameters seg#, offset, mode, user, and project. GIVE calls only kernel level internal functions. It is written in Project SUE System Language.

3.2.5.1 Description

GIVE adds an ACL element to the directory entry of a segment if all constraints are satisfied. It calls WRITEDIR which returns ERR_FLAG if the process does not have write access to the parent directory of the segment. If WRITEDIR returns ERR_FLAG or if DIR_DISK for the segment is zero indicating the segment is non-existent, GIVE returns with ERR_FLAG. GIVE then scans the ACL beginning at the element specified by DIR_ACL_HEAD until ACL_CHAIN equals zero.
If, for an existing element, the logical and of ACL_USER and ACL_USER_MASK matches the USER specified in the function call, and ACL_PROJECT matches the PROJECT specified in the function call, GIVE returns with ERR_FLAG. Otherwise, it allocates an ACL element ACLE# equal to ACL_CHAIN ( ), which holds the head of the free element chain. If ACLE# equals zero, no more elements are free, and GIVE returns with ERR_FLAG; if not, GIVE resets ACL_CHAIN to the number of the next element held in ACL_CHAIN(ACLE#). GIVE then finds the proper position for the new element. POSITION is set to zero. If DIR_ACL_HEAD is zero, the ACL is empty and the element may be inserted at POSITION zero. If the ACL is not empty, and USER is ALL_USERS and PROJECT is ALL_PROJECTS the new element belongs at the end of the list: GIVE finds this by setting POSITION to DIR_ACL_HEAD(OFFSET) and resetting it to ACL_CHAIN(POSITION) repeatedly until ACL_CHAIN(POSITION) equals zero. If USER is ALL_USERS or PROJECT is ALL_PROJECTS but not both, GIVE tests if the first entry in the ACL, ACL_USER(DIR_ACL_HEAD(OFFSET)) and ACL_USER_MASK is ALL_USERS. If so, the element can be placed first in the list at POSITION zero. If not, POSITION is set to DIR_ACL_HEAD(OFFSET), and until ACL_CHAIN(POSITION) is zero marking the end of the chain or ACL_USER(ACL_CHAIN(POSITION)) and ACL_USER_MASK is ALL_USERS, POSITION is reset to ACL_CHAIN(POSITION). GIVE now fills in the new element, ACL_USER(ACLE#) assigned USER or MODE and ACL_PROJECT(ACLE#) assigned PROJECT. If POSITION is zero, the element is entered at the beginning of the list by setting ACL_CHAIN(ACLE#) to DIR_ACL_HEAD(OFFSET) and DIR_ACL_HEAD(OFFSET) to ACLE#. Otherwise, the element is inserted after the ACL element identified by POSITION: ACL_CHAIN(ACLE#) is assigned ACL_CHAIN(POSITION) and ACL_CHAIN(POSITION) is assigned ACLE#. Finally, if MODE is less than WRITE$READ$EXECUTE_ACCESS, SOADD is called to remove the segment from the work spaces of processes whose access rights have been rescinded. RC is then set to OK_FLAG.

Function: GIVE
Parameters: GIVE(process#,aste#,offset,mode,user_id,project_id)
Effect:
If not AST_WAL(aste#,process#) | (AST_TYPE(aste#) # DIRECTORY) | (DIR_SIZE(aste#,offset) = 0) | DUPACL(aste#, 'DIR_ACL_HEAD'(aste#,offset), user_id, project_id) | ('ACL_CHAIN'(aste#, 0) = 0);
THEN: RC(process#) = NO;
ELSE: Let acle# = 'ACL_CHAIN'(aste#,0);
ACL_CHAIN(aste#,0) = 'ACL_CHAIN'(aste#,acle#);
Let position = FACLPOS(aste#, 'DIR_ACL_HEAD'(aste#,offset), user_id, project_id);
IF position = 0;
THEN:  ACL_CHAIN(aste#,acle#) =
    'DIR_ACL_HEAD'(aste#,offset);
    DIR_ACL_HEAD(aste#,offset) = acle#;
ELSE:  ACL_CHAIN(aste#,acle#) = 'ACL_CHAIN'(aste#,position);
    ACL_CHAIN(aste#,position) = acle#;
END;
ACL_USER(aste#,acle#) = user_id;
ACL_PROJECT(aste#,acle#) = project_id;
ACL_MODE(aste#,acle#) = mode;
SOADD(aste#,offset);
RC(process#) = YES;
END;

Function: DUPACL
Parameters: DUPACL(aste#,acle#,user_id,project_id);
Value:
IF acle# = 0;
THEN:  FALSE;
ELSE:
    IF (ACL_USER(aste#,acle#) = user_id) &
        (ACL_PROJECT(aste#,acle#) = project_id);
    THEN:  TRUE;
    ELSE:  DUPACL(aste#,ACL_CHAIN(aste#,acle#),user_id,
                                project_id);
    END;
END;

Function: FACLPOS
Parameters: FACLPOS(aste#,acle#,user_id,project_id)
Value:
IF acle# = 0;
THEN:  0;
ELSE:
    IF (user_id = ALL_USERS) &
        (project_id = ALL_PROJECTS);
    THEN:  FINDEND(aste#,acle#);
    ELSE:
        IF (user_id = ALL_USERS)
            (project_id = ALL_PROJECTS);
        THEN:
IF ACL_USER(aste#, acle#) = ALL_USERS;
THEN:  0;
ELSE:  FINDUSER(aste#, acle#);
END;
ELSE:  0;
END;
END;
END;

Function:  FINDEND
Parameters:  FINDEND(aste#, acle#)
Value:
IF ACL_CHAIN(aste#, acle#) ≠ 0;
THEN:  FINDEND(aste#, ACL_CHAIN(aste#, acle#));
ELSE:  acle#;
END;

Function:  FINDUSER
possible values:  acle#
Parameters:  FINDUSER(aste#, acle#);  
Value:
IF (ACL_CHAIN(aste#, acle#) = 0) | (ACL_USER(aste#, acle#) = ALL_USERS
THEN:  acle#;
ELSE:  FINDUSER(aste#, ACL_CHAIN(aste#, acle#));
END;

3.2.5.2 N/A

3.2.5.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCHECK</td>
<td>WRITEDIR</td>
</tr>
<tr>
<td></td>
<td>SOADD</td>
</tr>
</tbody>
</table>

3.2.5.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function GIVE. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.
3.2.5.5 Limitations

GIVE returns ERR_FLAG if DIR_DISK for the segment is zero, if ACL_USER and ACL_USER_MASK matches the USER specified and ACL_PROJECT matches the PROJECT specified, or if no elements are free. Otherwise, RC = OK_FLAG.

3.2.5.6 Listing

DATA GIVE(ASTE#, OFFSET, MODE, USER, PROJECT) RETURNS (RC);

PROGRAM GIVE;
DECLARE WORD (ACLE#, INDEX, POSITION);
/
SECURITY CHECKS FIRST
/
GIVE WRITES INTO DIRECTORY (INTERPRETATIVE WRITE)
IF WRITEDIR(ASTE#) = ERR_FLAG;
THEN:
.... RETURN WITH ERR_FLAG;
END;
/
IMPLEMENTATION CHECKS
/
SEGMENT MUST EXIST
IF DIR_DISK(OFFSET) = 0;
THEN:
.... RETURN WITH ERR_FLAG;
END;
/* SEARCH FOR DUPLICATE ACL ELEMENT */
INDEX := DIR_ACL_HEAD(OFFSET);

CYCLE
.... EXIT WHEN INDEX = 0;
    IF ((USER = (ACL_USER(INDEX) & ACL_USER_MASK)) & (PROJECT = ACL_PROJECT(INDEX))
    THEN:
        RETURN WITH ERR_FLAG;
    END;
    INDEX := ACL_CHAIN(INDEX);
END;
/* ALLOCATE AN ACL ELEMENT */
ACLE# := ACL_CHAIN(0);
IF ACLE# = 0;
THEN:
    RETURN WITH ERR_FLAG;
END;
/* CHECKING COMPLETE - PERFORM STATE CHANGE */
ACL_CHAIN(0) := ACL_CHAIN(ACLE#);
/* DETERMINE CORRECT POSITION FOR NEW ACL ELEMENT */
POSITION := 0;
IF DIP_ACL_HEAD(OFFSET) = 0;
THEN:
    IF ((USER = ALL_USERS) & (PROJECT = ALL_PROJECTS)):
        THEN: POSITION := DIP_ACL_HEAD(OFFSET);
        CYCLE
            EXIT WHEN ACL_CHAIN(POSITION) = 0;
            POSITION := ACL_CHAIN(POSITION);
        END;
    ELSE:
        IF ((USER = ALL_USERS) | (PROJECT = ALL_PROJECTS)):
            THEN: IF (ACL_USER(DIR_ACL_HEAD(OFFSET)) & ACL_USER_MASK) = ALL_USERS;
                    THEN: POSITION := DIR_ACL_HEAD(OFFSET);
                    CYCLE
                        EXIT WHEN (ACL_CHAIN(POSITION) = 0) | ((ACL_USER(ACL_CHAIN(POSITION)) & ACL_USER_MASK) = ALL_USERS);
                        POSITION := ACL_CHAIN(POSITION);
                    END;
            END;
        END;
    END;
END;
END;
3.2.6 Rescind Access (RESCIND)

The Rescind Access CPC, RESCIND, is a user level SKCPP function that is called by user level external programs with the parameters seg#, offset, user, and project. RESCIND calls only kernel level internal functions. It is written in Project SUE System Language.

3.2.6.1 Description

RESCIND moves an element from an entry's ACL to the directory's free element chain. If WRITEDIR shows that the ASTE# supplied does not identify a directory to which the process has write access, RESCIND returns ERR_FLAG. Otherwise, RESCIND searches the ACL for an element such that ACL_USER(INDEX) and ACL_USER_MASK equals USER and ACL_PROJECT equals PROJECT, starting with INDEX set to DIR_ACL_HEAD(OFFSET). If INDEX is 0, the end of the list has been reached without finding the specified element, so ERR_FLAG is returned. If the element identified by INDEX is the one specified, RESCIND stops its search; otherwise SAVE_LAST is set to INDEX, INDEX is reset to ACL_CHAIN(INDEX), and the search is continued.

After the element is found, if INDEX equals DIR_ACL_HEAD(OFFSET) DIR_ACL_HEAD(OFFSET) is reset to ACL_CHAIN(INDEX). If not, ACL_CHAIN(SAVE_LAST) is set to ACL_CHAIN(INDEX). The element thus removed is then put at the head of the free chain by assigning ACL_CHAIN(INDEX) the value of ACL_CHAIN(0) and ACL_CHAIN(0) the value of INDEX. SOADD is called to remove the entry from the work spaces of processes whose access rights have been limited by RESCIND. RESCIND then returns with RC set to OK_FLAG.
Function: RESCIND
Parameters: RESCIND(process#, aste#, offset, user_id, project_id)
Effect:
IF not AST.VAL(aste#, process#):
  (AST_TYPE(aste#) # DIRECTORY)
  (DIR_SIZE(aste#, offset) = 0)
  not DUPACL(aste#, 'DIR_ACL_HEAD' (aste#, offset), user_id, project_id);
THEN: RC(process#) = NO;
ELSE: Let acle# = FINDACLE(aste#, 'DIR_ACL_HEAD'(aste#, offset), user_id, project_id);
  IF acle# = 'DIR_ACL_HEAD'(aste#, offset);
  THEN: DIR_ACL_HEAD(aste#, offset)
       ACL_CHAIN(aste#, acle#);
  ELSE: Let pacle# = FINDPLACE(acle#);
        'DIR_ACL_HEAD'(aste#, offset), acle#);
        ACL_CHAIN(aste#, pacle#) = 'ACL_CHAIN'(aste#, acle#);
END;

Function: FINDACLE
Parameters: FINDACLE(aste#, acle#, user_id, project_id)
Value:
IF (ACL_USER(aste#, acle#) = user_id)&
   (ACL_PROJECT(aste, acle#) = project_id);
THEN: acle#;
ELSE: FINDACLE(aste#, ACL_CHAIN(aste#, acle#), user_id, project_id);
END:

Function: FINDPACLE
Parameters: FINDPACLE(aste#, vacle#, acle#)
Value:
IF ACL_CHAIN(aste, vacle#) = acle#;
THEN: vacle#;
ELSE: FINDPACLE(aste#, ACL_CHAIN(aste#, vacle#), acle#);
END:

3.2.6.2 N/A

3.2.6.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCHECK</td>
<td>WRITEDIR</td>
</tr>
<tr>
<td></td>
<td>SOADD</td>
</tr>
<tr>
<td></td>
<td>52</td>
</tr>
</tbody>
</table>
3.2.6.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function RESCIND. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

<table>
<thead>
<tr>
<th>Data Base References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global References</td>
</tr>
<tr>
<td>DIR_ACL_HEAD</td>
</tr>
<tr>
<td>ACL_USER</td>
</tr>
<tr>
<td>ACL_PROJECT</td>
</tr>
<tr>
<td>ACL_CHAIN</td>
</tr>
</tbody>
</table>

Constants

ALL_USER_MASK
ERR_FLAG
OF_FLAG

3.2.6.5 Limitations

RESCIND returns ERR_FLAG if the ASTE# supplied does not identify a directory to which the process has write access or if ACL_USER and ACL_USER MASK does not match the USER specified and ACL_PROJECT does not match the PROJECT specified. Otherwise, RC = OK_FLAG.

3.2.6.6 Listing

DATA RESCIND (ASTE#, OFFSET, USER, PROJECT) RETURNS (RC):

PROGRAM RESCIND:
DECLARE
   WORD (INDEX, SAVE_LAST):
   /* SECURITY CHECKS FIRST */
   /* RESCIND IS INTERPRETATIVE DIRECTORY WRITE */
   IF WRITEDIR (ASTE#) = ERR_FLAG:
      THEN:
         RETURN WITH ERR_FLAG:
      END:
/* IMPLEMENTATION CHECKS */
/* SEARCH FOR SPECIFIED ACL ELEMENT */

INDEX := DIR_ACL_HEAD(OFFSET); CYCLE

IF INDEX = 0;
THEN:
    RETURN WITH ERR_FLAG;
END;

IF ((USER = (ACL_USER(INDEX) & ACL_USER_MASK)) & (PROJECT = ACL_PROJECT(INDEX)));
THEN:
    EXIT;
END;

SAVE_LAST := INDEX;
INDEX := ACL_CHAIN(INDEX);
END;

/* CHECKING COMPLETE - PERFORM STATE CHANGE */
/* REMOVE FOUND ACL ELEMENT FROM CHAIN */

IF INDEX = DIR_ACL_HEAD(OFFSET);
THEN: DIR_ACL_HEAD(OFFSET) := ACL_CHAIN(INDEX);
ELSE: ACL_CHAIN(SAVE LAST) := ACL_CHAIN(INDEX);
END;

END;

3.2.7 Get Write Access (GETW)

The Get Write Access CPC, GETW, is a user level SKCPP function that is called by one other user level external function and by user level external programs with the parameters seg# and offset. GETW calls only kernel level internal functions. It is written in Project SUE System Language.

3.2.7.1 Description

GETW gets write access to a segment if security requirements are met. It calls DSEARCH which returns ERR_FLAG if the user does not have WRITE$READ$EXECUTE_ACCESS according to the ACL. If DSEARCH returns ERR_FLAG, GETW returns with ERR_FLAG.

If PS_CURRENT_PROCESS does not equal EXEC_PROCESS#, the process is not trusted and the *-property must be enforced. The *-property requires that all objects to which a subject has write access have
the same security level and that all objects to which it has read access have a security level less than or equal to the write security level. Thus, if PS_CUR_CLASS is not identical to DIR_CLASS(OFFSET) and DIR_CLASS_MASK or if PS_CUR_CAT is not identical to DIR_CAT(OFFSET) GETW returns with ERR_FLAG.

If PS_CURRENT_PROCESS does not equal EXEC_PROCESS#, the process is trusted and only the preservation of security need be enforced. If PS_CUR_CLASS is less than DIR_CLASS(OFFSET) and DIR_CLASS_MASK or if PS_CUR_CAT is not equal to DIR_CAT(OFFSET) or PS_CUR__CAT, GETW returns with ERR_FLAG.

Security checking complete, GETW calls CONNECT. If the process is not already connected to the seg and has a free seg#, the segment is activated if necessary and the process is added to the CPL and the WAL of the ASTE. GETW returns the seg# with which the process can identify the segment.

Function: GETW
Parameters: GETW(process#, aste#, offset)
Effect:
IF (ASTE_TYPE(aste# ≠ DIRECTORY)|
(DIR_SIZE(aste#, offset) = 0)|
not DSEARCH(process#, aste#, DIR_ACL_HEAD(aste#, offset), WRITE);
THEN: RC(process#) = NO;
ELSE:
IF PS_TYPE(process#) = TRUSTED;
THEN:
IF (PS_CUR_CLASS(process#) < DIR_CLASS(aste#, offset))|
(PS_CUR_CAT(process#) < DIR_CAT(aste#, offset));
THEN: RC(process#) = NO;
ELSE: CONNECT(process#, aste#, offset, WRITE);
END;
ELSE:
IF (PS_CUR_CLASS(process#) ≠ DIR_CLASS(aste#, offset))|
(PS_CUR_CAT(process#) ≠ DIR_CAT(aste#, offset));
THEN: RC(process#) = NO;
ELSE: CONNECT(process#, aste#, offset, WRITE);
END;
END;

3.2.7.2 N/A
3.2.7.3 Interfaces

Called By
STARTP
PCHECK

Calls
DSEARCH
CONNECT

3.2.7.4 Data Organization

Listed below are Security Kernel data base references and constants used by function GETW. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraphs 3.3.1. For constants refer to Table I, List of constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS_CURRENT_PROCESS</td>
<td>ASTE#</td>
<td>RC</td>
</tr>
<tr>
<td>PS_CUR_CLASS</td>
<td>OFFSET</td>
<td></td>
</tr>
<tr>
<td>PS_CUR_CAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR_CLASS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR_CAT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constants

DIR_CLASS_MASK
ERR_FLAG
EXEC_PROCESS#
WRITE$READ$EXECUTE_ACCESS

3.2.7.5 Limitations

GETW returns ERR_FLAG if the user does not have WRITE$READ$EXECUTE_ACCESS to the segment, if PS_CURRENT_PROCESS does not equal EXEC_PROCESS# or if PS_CUR_CLASS and PS_CUR_CAT does not match DIR_CLASS and DIR_CAT. Otherwise, RC = seg#.

3.2.7.6 Listing

DATA GETW(ASTE#, OFFSET) RETURNS (RC);
PROGRAM GETW:

/* SECURITY CHECKS FIRST */
/* SEARCH DIRECTORY ACL */

IF DSEARCH(ASTE#, OFFSET, WRITE$READ$EXECUTE_ACCESS) = ERR_FLAG;
   THEN:
      RETURN WITH ERR_FLAG;
      END;

IF PS_CURRENT_PROCESS = EXEC_PROCESS#;
   THEN:
      /* CHECK FOR PRESERVATION OF SECURITY AND -PROPERTY */
      IF PS_CUR CLASS = (DIR_CLASS(OFFSET) & DIR_CLASS_MASK):
         THEN:
            RETURN WITH ERR_FLAG;
            END;

      IF PS_CUR_CAT = DIR_CAT(OFFSET):
         THEN:
            RETURN WITH ERR_FLAG;
            END;

      ELSE:
      IF PS_CUR_CLASS < (DIR_CLASS(OFFSET) & DIR_CLASS_MASK):
         THEN:
            RETURN WITH ERR_FLAG;
            END;

      IF PS_CUR_CAT = (DIR_CAT(OFFSET) & PS_CUR_CAT):
         THEN:
            RETURN WITH ERR_FLAG;
            END;

      END;

/* IMPLEMENTATION CHECKS */
/* CONNECT PROCESS TO AST ENTRY FOR THIS SEGMENT */

RC := CONNECT(ASTE#, OFFSET, WRITE$READ$EXECUTE_ACCESS);

3.2.8 Get Read Access (GETR)

The Get Read Access CPC, GETR, is a user level external SKCPP function that is called by one other user level external function and by user level external programs with the parameters seg# and offset. GETR calls only kernel level internal functions. It is written in Project SUE System Language.

3.2.8.1 Description

GETR gets access to a segment identified by aste#, offset for a process, if security requirements are satisfied. It calls DSEARCH which scans the ACL of the segment and returns ERR_FLAG if the process lacks READ$EXECUTE_ACCESS. If DSEARCH returns
ERR_FLAG, GETR returns with ERR_FLAG. GETR then checks the preservation of security. If PS_CUR_CLASS is less than DIR_CLASS(OFFSET) and DIR_CLASS_MASK or or if PS_CUR_CAT does not equal DIR_CAT(OFFSET) or PS_CUR_CAT, ERR_FLAG is returned.

Otherwise, security checking is complete, and CONNECT is called. It adds the process to the CPL of the segment's ASTE and returns its seg# if the process is not already connected to the segment and has a free seg#. GETR then returns with the seg# which the process can subsequently use to refer to the segment.

Function: GETR
Parameters: GETR(process#, aste#, offset)
Effect:
IF (AST_TYPE(aste#) = DIRECTORY) |
(DIR_SIZE(aste#, offset) = 0) |
not DSEARCH(process#, aste#, DIR_ACL_HEAD(aste#, offset), READ) |
(PS_CUR_CLASS(process#) < DIR_CLASS(aste#, offset)) |
(PS_CUR_CAT(process#) > DIR_CAT(aste#, offset));
THEN: RC(process#) = NO;
ELSE: CONNECT(process#, aste, offset, READ);
END;

3.2.8.2 N/A

3.2.8.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>STARTP</td>
<td>DSEARCH</td>
</tr>
<tr>
<td>PCHECK</td>
<td>CONNECT</td>
</tr>
</tbody>
</table>

3.2.8.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function GETR. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS_CUR_CLASS</td>
<td>ASTE#</td>
<td>RC</td>
</tr>
<tr>
<td>PS_CUR_CAT</td>
<td>OFFSET</td>
<td></td>
</tr>
</tbody>
</table>
3.2.8.5 Limitations

GETR returns ERR_FLAG if the process does not have READ$EXECUTE_ACCESS to the segment or if PS_CUR_CLASS and PS_CUR_CAT do not match DIR_CLASS and DIR_CAT. Otherwise, RC = seg#.

3.2.8.6 Listing

DATA GETR(AS#*, OFFSET) RETURNS (RC):

PROGRAM GETE:
    /* SECURITY CHECKS FIRST */ /* SEARCH DIRECTORY ACL */
    IF DSFARCH(AS#*, OFFSET, READ$EXECUTE_ACCESS) = ERR_FLAG:
        END;
    /* CHECK FOR PRESERVATION OF SECURITY AND -PROPERTY */
    IF PS_CUR_CLASS < (DIR_CLASS(OFFSET) & DIR_CLASS_MASK):
        END;
    IF PS_CUR_CAT -= (DIR_CAT(OFFSET) | PS_CUR_CAT):
        END;
    /* IMPLEMENTATION CHECKS */
    /* CONNECT PROCESS TO AST ENTRY FOR THIS SEGMENT */
    RC := CONNECT(AS#*, OFFSET, READ$EXECUTE_ACCESS);

3.2.9 Release Segment (DCONNECT)

The Release Segment CPC, DCONNECT, is a user level external SKCPP function that is called by user level external functions, one kernel level internal function, and by user external programs with
the parameter seg#. DCONNECT calls only one kernel level internal function. This function is unique in that it is known by two names: external to the Security Kernel by RELEASE and internal to the Security Kernel by DCONNECT. The rationale being the nomenclature RELEASE matches a function in the mathematical model (reference 2.1a) where as the procedure, while being true to the mathematical model, is the logical inverse of the internal procedure CONNECT. DCONNECT is written in Project SUE Language.

3.2.9.1 Description

DCONNECT releases a segment from the process's WS as long as the seg# is valid. It sets BLOCK# to AST_ADR(ASTE#), which holds the main memory address of the segment, if any, or zero. If BLOCK# is not zero and AST_UNLOCK(ASTE#) and AST_UNLOCK_MASK does not equal AST_UNLOCK_FLAG, the segment is in the AS of the process and must be disabled. In this case, starting with REG# set to zero until REG# equals REG#_MAX, DCONNECT tests if PS_SAR(REG#) equals BLOCK#. If so, DISABLE can be called to remove the segment identified by REG# from the AS.

The process can now be disconnected from the segment. AST_CPL (ASTE#) is assigned the logical and of PS_PROCESS_NOTMASK, which consists of all one's except for the bit corresponding to the process#, and AST_CPL(ASTE#). Similarly, AST_WAL(ASTE#) is assigned PS_PROCESS_NOTMASK and AST_WAL(ASTE#).

If the WAL is empty DCONNECT must free any processes that are blocked on the segment semaphore, since P and V require write access to the segment. Thus, if AST_WAL(ASTE#) equals zero and AST_CPL(ASTE#) and WIRED_DOWN_MASK does not equal WIRED_DOWN, DCONNECT calls V repeatedly while SMFR_COUNT is negative. On exiting this loop, SMFR_COUNT(ASTE#) is set to 1 and SMFR_POINTER (ASTE#) is set to \(\emptyset\).

If \(\text{AST\_CPL}(\text{ASTE#}) = \emptyset\), the segment is not in the working space of any process. Therefore, DCONNECT sets AST_AGE_CHAIN(ASTE#) to AST_AGE_CHAIN(\(\emptyset\)) and AST_AGE_CHAIN(\(\emptyset\)) to ASTE#.

Finally, DCONNECT puts SEG# on the chain of free segment numbers; PS_SEG(SEG#) is set to PS_SEG(\(\emptyset\)) and PS_SEG(\(\emptyset\)) is reset to SEG# or SEG_FLAG.
Function: RELEASE
Parameters: RELEASE(process#, aste#, seg#)
Effect:
Let block# = 'AST_ADR'(aste#);
IF (block# ≠ 0);
THEN:
   (Vreg#) (REG_MIN ≤ reg# ≤ REG_MAX) &
   IF ('PS_SAR'(process#, reg#) = block#);
   THEN: DISABLE(process#, reg#);
END;
END;
END;

AST_CPL(aste#, process#) = FALSE;
AST_WAL(aste#, process#) = FALSE;
IF not (i)(PROCESS_MIX ≤ i ≤ PROCESS_MAX) &
   (AST_CPL(aste#, i) = TRUE));
THEN: AST(aste#);
END;

PS_SEG(process#, seg#) = 'PS_SEG'(process#, 0);
PS_SEG(process#, 0) = seg#;
PS_SEG_INUSE(process#, seg#) = FALSE;

3.2.9.2 N/A

3.2.9.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCHECK</td>
<td>DISABLE</td>
</tr>
<tr>
<td>STARTP</td>
<td></td>
</tr>
<tr>
<td>STOFP</td>
<td></td>
</tr>
<tr>
<td>SOADD</td>
<td></td>
</tr>
</tbody>
</table>

3.2.9.4. Data Organization

Listed below are Security Kernel data base references and constants used by the function DCONNECT. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.
3.2.9.5 Limitations

None

3.2.9.6 Listing

DATA DCONNECT(SEC#, ASTE#);

PROGRAM DCONNECT;
   DECLARE
      WORD (BLOCK#, REG#);
      /* PROCESS MUST NOT HAVE ANY DESCRIPTORS ON SEGMENT LOCKED IN BLOCK# := AST_ADR(ASTE#);
         IF (BLOCK# = 0) & ((AST_UNLOCK(ASTE#) & AST_UNLOCK_MASK) = AST_UNLOCK_FLAG): Then:
         /* MY BLOCKS TO THEIRS IXLIVE(ASL, BLOCK#);
            INLINF(ASL, BLOCK#);
            DO REG# := 0 TO REG#_MAX;
               IF PS_SAR(REG#) = BLOCK#:
                  THEN: DISABLE(REG#):
               END;
            END;
         END;
      END;
   END;
END;
62
3.2.10 Enable (ENABLE)

The Enable CPC, ENABLE, is a user level external SKCPP function that is called by other user level external functions and by user level external programs with the parameters seg# and reg#. ENABLE calls only kernel level internal functions. It is written in Project SUE System Language, including the Inline feature.

3.2.10.1 Description

ENABLE moves a segment from a process's WS to its AS if implementation constraints are satisfied. If REG# is greater than CROSS_REG#, it sets P_REG# to REG# + REG_CONSTANT; otherwise it sets P_REG# to REG#.

If PS_SAR(REG#) is not ∅, the register specified is not free and ENABLE returns with ERR_FLAG. It next tests whether sufficient space in the user's memory is available. If the logical and of AST_CPL(ASTE#) and WIRED_DOWN_MASK does not equal WIRED_DOWN and AST_SIZE(ASTE#) is greater than PS_MEM_QUOTA, ENABLE returns with ERR_FLAG. If not, implementation checks are complete.

ENABLE then insure that the segment is in main memory by calling SWAPIN if AST_ADR(ASTE#) is zero. If AST_UNLOCK(ASTE#) and AST_UNLOCK_MASK equals AST_UNLOCK_FLAG, the segment is currently eligible to be swapped out of main memory and must be removed from
the swap chain. Starting with \( \text{INDWX} \) set to 0, ENABLE repeatedly
sets NEXT to \( \text{AST\_SWAP\_CHAIN(INDEX)} \) and INDEX to NEXT, leaving the
loop upon setting NEXT to \( \text{ASTE#} \). It then assigns to \( \text{AST\_SWAP\_CHAIN(INDEX)} \),
to \( \text{AST\_DES\_COUNT(ASTE#)} \) the value zero, and to \( \text{AST\_UNLOCK(ASTE#)} \) the logical and of
\( \text{AST\_UNLOCK(ASTE#)} \) and \( \text{AST\_LOCK\_MASK} \).

To determine the mode of access, ENABLE calculates the logical and
of \( \text{AST\_TYPE(ASTE#)} \) and \( \text{AST\_TYPE\_MASK} \). If this equals
\( \text{AST\_TYPE\_DIRECTORY} \), MODE is set to zero. Otherwise, if \( \text{AST\_WAL\_ (ASTE#)} \)
and \( \text{P\_PROCESS\_MASK} \) equals zero MODE is set to \( \text{SDR\_READ\_ACCESS} \); if not, MODE is set equal to \( \text{SDR\_WRITE\_ACCESS} \).

Sets priority level high through the Inline feature, and LSD
is called to construct the descriptors and store them in the appropriate segmentation registers. If \( \text{PS\_CURRENT\_PROCESS} \) is \( \text{THE\_CURRENT\_PROCESS} \), ENABLE loads the hardware registers as well: \( \text{SDR(P\_REG#)} \)
is assigned \( \text{PS\_SDR(REG#)} \) and \( \text{SAR(P\_REG#)} \) is assigned \( \text{PS\_SAR(REG#)} \).

To complete the operation, sets priority level low through the
Inline feature, and \( \text{AST\_DES\_COUNT(ASTE#)} \) is incremented if
\( \text{AST\_DES\_COUNT(ASTE#)} \) is incremented. If \( \text{AST\_CPL(ASTE#)} \) and
\( \text{WIRED\_DOWN\_MASK} \) does not equal WIRED\_DOWN, the user's memory space,
\( \text{PS\_MEM\_QUOTA} \) is diminished by \( \text{AST\_SIZE(ASTE#)} \). ENABLE then
returns RC set equal to \( \text{OK\_FLAG} \).

Function: ENABLE
Parameters: ENABLE(process#, reg#)
Effect:
Let \( \text{size} = \text{AST\_SIZE(aste#)} \);
IF (PS\_SAR(process#, reg#) \neq 0)
      ((\text{AST\_WIRED\_DOWN(aste#)} = \text{OFF}) \&
       (\text{size} > \text{\text{PS\_MEM\_QUOTE} (process#)}));
THEN: \( \text{RE(process#)} = \text{NO}; \)
ELSE:
      IF \( \text{\text{AST\_ADR}(ASTE#)} = 0; \)
         THEN: \( \text{SWAPIN(aste#)}; \)
      END;
      IF \( \text{\text{AST\_LOCK}(aste#)} = \text{UNLOCKED}; \)
         THEN: \( \text{LOCK(aste#)}; \)
      END;

64
IF AST_TYPE(aste#) = DIRECTORY
THEN: Let mode = NO;
ELSE:
    IF AST_WAL(aste#, process#) = TRUE;
    THEN: Let mode = WRITE;
    ELSE: Let mode = READ;
    END:
END;
LSD(AST_ADR(aste#), reg#, mode);
IF AST_WIRED_DOWN(aste#) = OFF;
THEN: PS_MEM_QUOTA(process#) = 'PS_MEM_QUOTA'(process#) - size;
END;
AST_DES_COUNT(aste#) = 'AST_DES_COUNT(aste#) + 1;
RC(process#) = YES

3.2.10.2 N/A

3.2.10/3 Interfaces

Refers to Figure 6, Function Call Matrix

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCHECK</td>
<td>SWAPIN</td>
</tr>
<tr>
<td>STARTP</td>
<td>LSD</td>
</tr>
</tbody>
</table>

3.2.10.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function ENABLE. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS_SAR</td>
<td>ASTE#</td>
<td>P_REG#</td>
</tr>
<tr>
<td>PS_SDR</td>
<td>REG#</td>
<td>MODE</td>
</tr>
<tr>
<td>PS_MEM_QUOTA</td>
<td></td>
<td>REG_ADR</td>
</tr>
<tr>
<td>PS_PROCESS_MASK</td>
<td></td>
<td>INDEX</td>
</tr>
<tr>
<td>PS_CURRENT_PROCESS</td>
<td></td>
<td>NEXT</td>
</tr>
<tr>
<td>AST_ADR</td>
<td></td>
<td>RC</td>
</tr>
<tr>
<td>AST_UNLOCK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_SWAP_CHAIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_DES_COUNT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

65
3.2.10.5 Limitations

ENABLE returns ERR_FLAG if PS_SAR(REG#) is not 0, if AST_CPL(ASTE#) and WIRED_DOWN_MASK does not equal WIRED_DOWN and if AST_SIZE(ASTE) is greater than PS_MEM_QUOTA. Otherwise, RC=OK_FLAG.

3.2.10.6 Listing

DATA ENABLE(AST#, REG#) RETURNS (RC);

PROGRAM ENABLE;
DECLARE
  WORD (P_REG#, MODE, REG_ADR, INDEX, NEXT);
  IF REG# > CROSS_REG#
    THEN: P_REG# := REG# * REG_CONSTANT;
           ELSE: P_REG# := REG#;
       END;
  /* REGISTER MUST BE FREE
  IF PS_SAR(REG#) => 0;
  THEN:
      . . . RETURN WITH ERR_FLAG;
  END;
  /* SPACE IN USER'S MEMORY MUST BE AVAILABLE
  IF (((AST_CPL(ASTE#) & WIRED_DOWN_MASK) => WIRED_DOWN) & (AST_SIZE(ASTE#) => PS_MEM_QUOTA);
  THEN:
     . . . RETURN WITH ERR_FLAG;
  END;
SWAPIN IF NECESSARY

If AST_ADR(ASTE#) = 0;
Then SWAPIN(ASTE#);
End;

REMOVE FROM SWAP CHAIN IF NECESSARY

If (AST_UNLOCK(ASTE#) & AST_UNLOCK_MASK) = AST_UNLOCK_FLAG;
Then: INDEX := 0;
    CYCLE
      NEXT := AST_SWAP_CHAIN(INDEX);
      .... Exit WHEN NEXT = ASTE#;
      INDEX := NEXT;
    END;
  AST_SWAP_CHAIN(INDEX) := AST_SWAP_CHAIN(ASTE#);
  AST_DESC_COUNT(ASTE#) := 0;
  AST_UNLOCK(ASTE#) := (AST_UNLOCK(ASTE#) & AST_LOCK_MASK);
End;

DETERMINE TYPE OF ACCESS PERMITTED

If (AST_TYPE(ASTE#) & AST_TYPE_MASK) = AST_TYPE_DIRECTORY;
Then: MODE := 0; /* DIRECTORY ACCESSES MUST BE INTERPRETIVE */
Else:
  If (AST_WAL(ASTE#) & PS_PROCESS_MASK) = 0;
  Then: MODE := SDR_READ_ACCESS;
  Else: MODE := SDR_WRITE_ACCESS;
End;

LOAD SEGMENT DESCRIPTOR

INLINE(SPLHIGH);
LSD(ASTE#, PS_SDR_ADDR + REG# * REG#, MODE);
/* IF THIS IS THE CURRENT PROCESS LOAD HARDWARE PEGS ALSO */
If PS_CURRENTPROCESS = THE_CURRENT_PROCESS;
Then: SDR(P_REG#) := PS_SDR(P_REG#);
      SAR(P_REG#) := PS_SAR(P_REG#);
End;
INLINE(SPLLOW);
/* INCREMENT DESCRIPTOR COUNT */
AST_DESC_COUNT(ASTE#) := AST_DESC_COUNT(ASTE#) + 1;
/* ADJUST USER'S QUOTA */
If (AST_CPL(ASTE#) & WIRED_DOWN_MASK) = WIRED_DOWN;
Then: PS_REP_QUOTA := PS_REP_QUOTA - AST_SIZE(ASTE#);
End;
RC := OK_FLAG;

3.2.11 Disable (DISABLE)

The Disable CPC, DISABLE, is a user level external SKCPP function that is called by other user level external functions and by user level external programs with the parameter reg#. It is written in Project SUE System Language, including the Inline feature.
3.2.11.1 Description

DISABLE removes a segment from AS. It sets BLOCK# to PS_SAR(REG#). If BLOCK# equals zero, the register REG# contains no descriptor and DISABLE has no effect. If not, BLOCK# contains the storage address of the segment with the 6 least significant bits omitted. Since the MBT omits the 8 least significant bits of the address, two ASR (arithmetic shift right) commands are executed (using Inline code) on BLOCK#. DISABLE then finds the ASTE# of the segment. If BLOCK# is less than END BLOCK#, ASTE# is assigned MBT (BLOCK#). Otherwise, each value from ASTE#_MIN to ASTE#_MAX is tested until one is found such that AST_ADR(ASTE#) equals BLOCK#.

The priority level is then set high (using Inline code) and the descriptor destroyed by setting PS_SDR(REG#) and PS_SAR(REG#) to 0. If PS_CURRENT_PROCESS is THE_CURRENT_PROCESS, the hardware segmentation registers are also cleared, as follows. If REG# is greater than CROSS_REG#, REG_CONSTANT is added to REG#. Next, if SDR(REG#) and SDR_CHANGE_MASK equals SDR_CHANGED, the change bit is set, and AST_CHANGE(ASTE#) is reset to AST_CHANGE(ASTE#) or AST_CHANGED. SDR(REG#) and SAR(REG#) are then zeroed and the priority level is set to low using Inline code.

AST_DES_COUNT(ASTE#) is then decremented, and if the segment is wired down, DISABLING is complete. If, however, AST_CPL(ASTE#) and WIRED_DOWN_MASK equals 0, the segment may be eligible for swapping out. If AST_DES_COUNT(ASTE#) is 0, the segment is added to the head of the swap chain by setting AST_SWAP_CHAIN(ASTE#) to AST_SWAP_CHAIN(0) and AST_SWAP_CHAIN(0) to ASTE#. The segment is also unlocked in this case by resetting AST_UNLOCK(ASTE#) to the logical or of AST_UNLOCK(ASTE#) and AST_UNLOCK_FLAG, the DISABLE function is then completed by crediting PS_MEM_QUOTA with AST_SIZE(ASTE#).

Function: DISABLE

Parameters: DISABLE(process#,reg#)
Effect:
IF PS_SAR(process#,reg#) ≠ 0;
THEN: Let block# = 'PS_SAR'(process#,reg#);
      Let aste# = MBT_ASTE(block#);
      AST_CHANGE(block#) = 'AST_CHANGE'(block#) | 'PS_SDR_CHANGE'(process#,reg#);
      PS_SAR(process#,reg#) = 0;
      PS_SDR(process#,reg#) = 0;
AST_DES_COUNT(aste#) = 'AST_DES_COUNT'(aste#) - 1;
IF (AST_DES_COUNT(aste#) = 0) &
(AST WIRED DOWN(aste#) = OFF);
THEN: UNLOCK(aste#);
END;
IF AST WIRED DOWN(aste#) = OFF;
THEN: PS_MEM QUOTA(process#) = 'PS_MEM QUOTA'(process#) +
size;
END;
END;

3.2.11.2 N/A

3.2.11.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCHECK</td>
<td>None</td>
</tr>
<tr>
<td>DCONNECT</td>
<td></td>
</tr>
</tbody>
</table>

3.2.11.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function DISABLE. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

Global References | Function Parameters | Local References
PS_SAR | REG# | BLOCK#
PS_SDR | | ASTE#
PS_MEM QUOTA | | |
PS_CURRENT_PROCESS | | |
AST ADR | | |
AST_UNLOCK | | |
AST_SWAP_CHAIN | | |
AST_DES_COUNT | | |
AST_CPL | | |
AST_CHANGE | | |
MBT_ASTE | | |
THE_CURRENT_PROCESS | | |
Constants

- ASR
- AST_CHANGE
- AST_UNLOCK_FLAG
- ASTE#_MAX
- ASTE#_MIN
- CROSS_REG#
- END_BLOCK#

3.2.11.5 Limitations

None.

3.2.11.6 Listing

DATA DISABLE(REG#):
DECLARE WORD (BLOCK#, ASTF#);

PROGRAM DISABLE;
/* TRANSLATE REG# TO ASTE# (AND CHECK FOR DESCRIPTOR IN REGISTER) */
BLOCK# := PS_SAR(REG#);
IF BLOCK# != 0;
THEN: INLINE(ASR, BLOCK#); /* THEIR BLOCKS TO MINE */
INLINE(ASR, BLOCK#);
IF BLOCK# < END_BLOCK#;
THEN: ASTE# := MBT_ASTE(BLOCK#);
ELSE:
DO ASTE# := ASTE#_MIN TO ASTE#_MAX;
.... EXIT WHEN AST_ADR(ASTE#) = BLOCK#;
END;

END;
/* DESTROY DESCRIPTOR */
INLINE(SPLHIGH);
PS_SDR(REG#) := 0;
PS_SAR(REG#) := 0;
/* IF FOR CURRENT PROCESS ALSO CLEAR SEGMENTATION REGISTER */
IF PS_CURRENT_PROCESS = THE_CURRENT_PROCESS;
THEN:
IF REG# > CROSS_REG#;
THEN: REG# := REG# + REG_CONSTANT;
END;
/* CHECK FOR CHANGE BIT BEING SET */
IF (SDR(RFG#) & SDR_CHANGE_MASK) = SDR_CHANGED;
THEN: AST_CHANGE(ASTE#) := (AST_CHANGE(ASTE#) | AST_CHANGED);
END;
/* CLEAR SEGMENTATION REGISTER */
SDR(RFG#) := 0;
SAR(RFG#) := 0;
END;
INLINE(SPLLOW);
/* DECREMENT DESCRIPTOR COUNT */
AST_DES_COUNT(ASTE#) := AST_DES_COUNT(ASTE#) - 1;
/* THE FOLLOWING DOES NOT APPLY TO WIRED DOWN SEGMENTS */
IF (AST_CPL(ASTE#) & WIRED_DOWN_MASK) = 0;
THEN:
    IF NO DESCRIPTORS LEFT THEN UNLOCK AND ADD TO SWAP CHAIN
    IF (AST_DES_COUNT(ASTE#) = 0):
        THEN: AST_SWAP_CHAIN(ASTE#) := AST_SWAP_CHAIN(0);
        AST_SWAP_CHAIN(0) := ASTE#;
        AST_UNLOCK(ASTE#) := (AST_UNLOCK(ASTE#) | AST_UNLOCK_FLAG);
END;
/* ADJUST MEMORY QUOTA */
PS_MEM_QUOTA := PS_MEM_QUOTA + AST_SIZE(ASTE#);
END;

3.2.12 Outer P (OUTERP)

The Outer P CPC, OUTERP, is a user level external SKCPP function that is called by user level external programs with the parameter seg#. OUTERP calls only one kernel level internal function. It is written in Project SUE System Language, including the Inline feature.

3.2.12.1 Description

OUTERP performs security and implementation checks preliminary to calling P when P is invoked externally.

If AST_WAL(ASTE#) logical and PS_PROCESS_MASK equals zero, the process does not have write access to the segment, and OUTERP returns with ERR_FLAG.

It then performs INLINE(SPLHIGH) to set the priority level high. If SMFR_COUNT(ASTE#) equals -128, it is beyond the bounds of the implementation and ERR_FLAG is returned. Otherwise, P is called to
decrement the semaphore associated with the specified segment and block the process if the result is negative. RC is set to OK_FLAG.

Function:  OUTERP
Parameters:  OUTERP(aste#)
Effect:
IF AST_WAL(aste#,TCP);
THEN:  P(aste#);
ELSE:  RC(TCP) = NO;
END;

3.2.12.2  N/A

3.2.12.3  Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCHECK</td>
<td>P</td>
</tr>
</tbody>
</table>

3.2.12.4  Data Organization

Listed below are Security Kernel data base references and constants used by the function OUTERP. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

<table>
<thead>
<tr>
<th>Data Base References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global References</td>
</tr>
<tr>
<td>PS_PROCESS_MASK</td>
</tr>
<tr>
<td>AST_WAL</td>
</tr>
<tr>
<td>SMFR_COUNT</td>
</tr>
<tr>
<td>Function Parameters</td>
</tr>
<tr>
<td>ASTE#</td>
</tr>
<tr>
<td>Local References</td>
</tr>
<tr>
<td>RC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR_FLAG</td>
</tr>
<tr>
<td>OK_FLAG</td>
</tr>
<tr>
<td>SPL_HIGH</td>
</tr>
</tbody>
</table>

3.2.12.5  Limitations

OUTERP returns ERR_FLAG if the process does not have write access to the segment or if the SMFR_COUNT for the segment equals 72.
Otherwise, RC=OK_FLAG. The SMFR_COUNT for an active segment runs from +127 to -128.

3.2.12.6 Listing

DATA OUTERP(ASTE#) RETURNS (RC):

PROGRAM OUTERP;
  
  /* SECURITY CHECKS FIRST */
  /* PROCESS MUST HAVE WRITE$READ ACCESS TO SEMAPHORE */
  IF (AST_WAL(ASTE#) & PS_PROCESS_MASK) = 0;
  THEN;
  .... RETURN WITH ERR_FLAG;
  END;
  
  /* IMPLEMENTATION CHECKS */
  INLINE(SPLHIGH);
  IF SMFR_COUNT(ASTE#) = -128;
  THEN;
  .... RETURN WITH ERR_FLAG;
  END;
  
  /* CHECKING COMPLETE - PERFORM STATE CHANGE */
  P(ASTE#);
  RC := OK_FLAG;

3.2.13 P (P)

The P CPC, P, is a kernel level internal SKCPP function that is called by both kernel level internal functions and one external user level function. P calls only kernel level internal functions. It is written in Project SUE System Language, including the Inline feature.

3.2.13.1 Description

P decrements a specified semaphore counter, and if the result is negative, blocks the process.

It blocks interrupts by setting the priority level high using Inline code, and then decrements SMFR_COUNT(SMFR#) by 1. If SMFR_COUNT(SMFR#) is less than zero, the process is added to the queue of processes blocked on this semaphore: PT_FLAGS(THE_CURRENT_PROCESS) is set to SMFR_POINTER(SMFR#) logical or BLOCKED and SMFR_POINTER(SMFR#) is set to THE_CURRENT_PROCESS.
If the current process has been blocked it also starts a new process running, as follows. If SMFR# is less than KERNEL_SMFR, the operation was performed on a segment semaphore. A V is performed on KERNEL_SMFR to prevent the possibility of a deadlock, SLEEP is called to find the next process ready to run, and a P is performed on KERNEL_SMFR to restore its previous condition. If SMFR# is greater than or equal to KERNEL_SMFR, P simply calls SLEEP. It then resets the priority level low using Inline code and returns control to the calling program.

Function:  P
Parameters:  P(smfr#)
Effect:
SMFR_COUNT(smfr#) = 'SMFR_COUNT'(smfr#) - 1;
IF SMFR_COUNT(smfr#) < 0;
THEN:  PT_FLAGS(TCP) = BLOCKED;
       PT_LINK(TCP) = 'SMFR_POINTER'(smfr#);
       SMFR_POINTER(smfr#) = TCP;
END;
RC(TCP) = YES;

3.2.13.2 N/A

3.2.13.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCHECK</td>
<td>P</td>
</tr>
<tr>
<td>OUTERP</td>
<td>V</td>
</tr>
<tr>
<td>SWAPIN</td>
<td>SLEEP</td>
</tr>
<tr>
<td>SWAPOUT</td>
<td></td>
</tr>
<tr>
<td>IPCRCV</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P</td>
</tr>
</tbody>
</table>

3.2.13.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function P. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.
3.2.13.5 Limitations

If the SMFR_COUNT for the specified SMFR# is less than 0 the process is blocked and added to the queue of processes blocked on that semaphore. The process becomes unblocked when enough V's are performed (see 3.2.15.1).

3.2.13.6 Listing

DATA P(SMFR#):

PROGRAM P;

/* BLOCK INTERRUPTS */
INLINE(SPLHIGH);

SMFR_COUNT(SMFR#) := SMFR_COUNT(SMFR#) - 1;

/* IF SEMAPHORE COUNT IS NEGATIVE, THEN PROCESS BECOMES BLOCKED */
IF SMFR_COUNT(SMFR#) < 0;

THEN:

/* ADD CURRENT PROCESS TO QUEUE OF PROCESSES BLOCKED ON SEMAPHORE */
PT_FLAGS(THF_CURRENT_PROCESS) := (SMFR_POINTER(SMFR#) | BLOCKED);
SMFR_POINTER(SMFR#) := THE_CURRENT_PROCESS;

/* START A NEW PROCESS RUNNING */
IF SMFR# < KERNEL_SMFR;
THEN: V(KERNEL_SMFR);
SLEEP;
P(KERNEL_SMFR);
ILSF: SLEEP;
END;

END;

INLINE(SPLLOW):
3.2.14 Outer V (OUTERV)

The Outer V CPC, OUTERV, is a user level external SKCPP function that is called by user level external programs with the parameter seg#. OUTERV calls one kernel level internal function. It is written in Project SUE System Language, including the Inline feature.

3.2.14.1 Description

OUTERV performs security and implementation checks whenever V is called externally. If AST_WAL(ASTE#) logical and PS_PROCESS_MASK equals 0, the process lacks write access to the semaphore, so OUTERV returns with ERR_FLAG. To prevent interrupts, it sets the priority level high using Inline code. Next, if SMFR_COUNT(ASTE#) equals 127, it is beyond the bounds of the implementation, and OUTERV returns with ERR_FLAG. Otherwise it performs the state change, called V to increment the semaphore, and, if the result is non-positive, makes ready a process blocked on the semaphore. OUTERV returns with RC set to OK_FLAG.

Function: OUTERV
Parameters: OUTERV(aste#)
Effect:
IF AST_WAL(aste#,TCP);
THEN: V(aste#);
ELSE: RC(TCP) = NO;
END;

3.2.14.2 N/A

3.2.14.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCHECK</td>
<td>V</td>
</tr>
</tbody>
</table>

3.2.14.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function OUTERV. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.
3.2.14.5 Limitations

OUTERV returns ERR_FLAG if the process does not have write access to the semaphore or if the SMFR_COUNT for the segment equals 127.

3.2.14.6 Listing

DATA OUTERV(ASTE#) RETURNS (RC);

PROGRAM OUTERV;

/** SECURITY CHECKS FIRST */
/** PROCESS MUST HAVE WRITE/READ ACCESS TO SEMAPHORE */
IF (AST_WAL(ASTE#) & PS_PROCESS_MASK) = 0;
THEN:
    RETURN WITH ERR_FLAG;
END;

/** IMPLEMENTATION CHECKS */
INLINE(SPL_HIGH);
IF SMFR_COUNT(ASTE#) = 127;
THEN:
    RETURN WITH ERR_FLAG;
END;

/** CHECKING COMPLETE - PERFORM STATE CHANGE */
V(ASTE#);
RC := OK_FLAG;
3.2.15 \( V(V) \)

The \( V \) CPC, \( V \), is a kernel level internal SKCPP function that is called by both kernel level internal functions and one user level external function. It is written in Project SUE System Language, including the Inline feature.

3.2.15.1 Description

\( V \) is the inverse of \( P \); it increments the specified semaphore counter and, if the result is non-positive, makes a blocked process ready. First, it prevents interrupts by setting the priority level high using Inline code. Then, it resets the \( \text{SMFR\_COUNT}(\text{SMFR}\#) \) to \( \text{SMFR\_COUNT}(\text{SMFR}\#) + 1 \).

If the \( \text{SMFR\_COUNT}(\text{SMFR}\#) \) is positive, the priority level is set low using Inline code and \( V \) is exited; if \( \text{SMFR\_COUNT}(\text{SMFR}\#) \) is less than or equal to zero, a blocked process must be unqueued. \( \text{PROCESS\_A} \) is assigned the value of \( \text{SMFR\_POINTER}(\text{SMFR}\#) \). If \( \text{SMFR\_COUNT}(\text{SMFR}\#) \) does not equal \( \emptyset \), \( V \) finds the process blocked longest by setting \( \text{PROCESS\_B} \) equal to \( \text{PROCESS\_A} \) and \( \text{PROCESS\_A} \) equal to \( \text{PT\_LINK}(\text{PROCESS\_B}) \) until \( \text{PT\_LINK}(\text{PROCESS\_A}) \) equals zero. It then removes \( \text{PROCESS\_A} \) from the end of the queue by setting \( \text{PT\_LINK}(\text{PROCESS\_A}) \) equal to \( \emptyset \). Otherwise, if \( \text{SMFR\_COUNT}(\text{SMFR}\#) \) equals \( \emptyset \), only \( \text{PROCESS\_A} \) is blocked on the semaphore; \( V \) removes it by setting \( \text{SMFR\_POINTER}(\text{SMFR}\#) \) to \( \emptyset \).

\( V \) then readies \( \text{PROCESS\_A} \) by assigning \( \text{PT\_FLAGS}(\text{PROCESS\_A}) \) the value \( \text{READY} \). It then sets the priority level low using Inline code and returns.

Function: \( V \)
Parameters: \( V(\text{smfr}\#) \)
Effect: \( \text{SMFR\_COUNT}(\text{smfr}\#) = \text{SMFR\_COUNT}(\text{smfr}\#) + 1; \)
\( \text{IF SMFR\_COUNT}(\text{smfr}\#) \leq 0; \)
\( \text{THEN:} \)
\( \text{IF SMFR\_COUNT}(\text{smfr}\#) = 0; \)
\( \text{THEN:} \) Let \( \text{process\#} = \text{SMFR\_POINTER}(\text{smfr}\#); \)
\( \text{SMFR\_POINTER}(\text{smfr}\#) = 0; \)
\( \text{ELSE:} \) Let \( \text{process\#} = \text{VEND}; \)
\( \text{VUNCHAIN}(\text{SMFR\_POINTER}(\text{smfr}\#)); \)
\( \text{END}; \)
\( \text{PT\_FLAGS}(\text{process\#}) = \text{READY}; \)
\( \text{END}; \)
\( \text{RC(TCP)} = \text{YES}; \)
Function: VEND
Parameters: VEND(process#)
Value:
IF 'PT_LINK'(process#) = 0;
THEN: process#;
ELSE: VEND('PT_LINK'(process#));
END;

Function: VUNCHAIN
Parameters: VUNCHAIN(process#)
Effect:
IF 'PT_LINK'(PT_LINK'(process#)) = 0;
THEN: PT_LINK(process#) = 0;
ELSE: VUNCHAIN('PT_LINK'(process#));
END;

3.2.15.2 N/A

3.2.15.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCHECK</td>
<td>None</td>
</tr>
<tr>
<td>OUTERV</td>
<td></td>
</tr>
<tr>
<td>IPCRCV</td>
<td></td>
</tr>
<tr>
<td>STOPP</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
</tbody>
</table>

3.2.15.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function V. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT_LINK</td>
<td>SMFR#</td>
<td>PROCESS_A</td>
</tr>
<tr>
<td>PT_FLAG</td>
<td></td>
<td>PROCESS_B</td>
</tr>
<tr>
<td>SMFR_COUNT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMFR_POINTER</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

79
Constants

READY
SPLHIGH
SPLLOW

3.2.15.5 Limitations

None.

3.2.15.6 Listing

DATA V(SMFR#);
DECLARE
WORD (PROCESS_A, PROCESS_B);

PROGRAM V;
/* BLOCK INTERRUPTS */
INLINE(SPLHIGH);
SMFR_COUNT(SMFR#) := SMFR_COUNT(SMFR#) + 1;
/* IF SEMAPHORE COUNT IS NON-POSITIVE, THEN A PROCESS BLOCKED ON THE SEMAPHORE MUST BE UNQUED */
IF SMFR_COUNT(SMFR#) <= 0;
THEN:
/* THE MOST RECENT PROCESS ADDED TO THIS SEMAPHORE'S QUEUE */
PROCESS_A := SMP_Pointer(SMFR#);
/* IF THERE IS MORE THAN ONE PROCESS ON QUEUE, FOLLOW CHAIN THROUGH PROCESS TABLE */
IF SMFR_COUNT(SMFR#) <= 0;
THEN:
CYCLE
.... EXIT WHEN PT_LINK(PROCESS_A) = 0;
PROCESS_B := PROCESS_A;
PROCESS_A := PT_LINK(PROCESS_B);
END;
/* REMOVE PROCESS_A FROM END OF QUEUE */
PT_LINK(PROCESS_B) := 0;
ELSE: SMP_Pointer(SMFR#) := 0;
END;
/* PROCESS_A BECOMES READY */
PT_FLAGS(PROCESS_A) := FEADY;
END;
INLINE(SPLLOW);
3.2.16  Send Interprocess Communication (IPCSEND)

The Send Interprocess Communication CPC, IPCSEND, is a user level external SKCPP function that is called by user level external programs and one user level external function with the parameters process# and message. It is written in Project SUE System Language.

3.2.16.1  Description

IPCSEND sends a message to a specified process if security and implementation constraints are met. If PT_FLAGS(PROCESS#) logical and PT_FLAGS_MASK equals INACTIVE, IPCSEND returns. It has no return code since that might be used to compromise security.

If PS_CURRENT_PROCESS does not equal EXEC_PROCESS#, it is not trusted and the following security checks must be made to preserve the *-property. If PT_CURRENT_CLASS(PROCESS#) is less than PS_CUR_CLASS or if PT_CUR_CAT(PROCESS#) does not equal PT_CUR_CAT (PROCESS#) logical or PS_CUR_CAT, IPCSEND returns with no effect.

If PT_IPC_QUOTA(PROCESS#) is zero, IPCSEND ignores the call because the receiving process has no more free IPC elements.

Checking complete, it then allocates an element from the queue of free elements by assigning INDEX the value of IPC_LINK(0), and resetting IPC_LINK(0) to IPC_LINK(INDEX) and IPC_LINK(INDEX) to 0. It fills in the element by letting IPC_PROCESS//(INDEX) equal PS_CURRENT_PROCESS logical or DOMAIN and letting IPC_DATA(INDEX) equal MESSAGE.

If PT_IPC_QUEUE_HEAD(PROCESS#) logical and BYTE_MASK equals IPC_WAIT, the specified process is blocked because it is awaiting a message and none was available. In this case, PT_IPC_QUEUE_HEAD (PROCESS#) is set to INDEX and the process is unblocked by setting PT_FLAGS(PROCESS#) to READY. Otherwise, the new IPC element must be appended to the process's queue of messages. If PT_IPC_QUEUE_HEAD (PROCESS#) equals zero, the process's IPC queue is empty, and PT_IPC_QUEUE_HEAD(PROCESS#) is set to INDEX. If not, to find the end of the queue, INDEX2 is set equal to PT_IPC_QUEUE_HEAD(PROCESS#). Until IPC_LINK(INDEX2) equals zero, INDEX2 is reset to IPC_LINK (INDEX2). Since INDEX2 now holds the IPC element number of the last element in the queue, letting IPC_LINK(INDEX2) equal INDEX attaches the new element to the end of the queue.

Finally, the IPC element quota of the receiving process is adjusted by decrementing PT_IPC_QUOTA(PROCESS#) and IPCSEND is exited.
Function: IPCSEND
Parameters: IPCSEND(process#, message, domain)
Effect:
IF (PT_FLAGS(process#) ≠ INACTIVE) &
  (((PS_CUR_CLASS(process#) ≠ PS_CUR_CLASS(TCP)) &
    (PS_CUR_CAT(process#) ≠ PS_CUR_CAT(TCP))) &
  (PT_TYPE(TCP) = TRUSTED)) &
  ('PT_IPC_QUOTA'(process#) ≠ 0);
THEN: Let ipce# = 'IPC_LINK'(0);
  IPC_LINK(0) = 'IPC_LINK'(ipce#);
  IPC_LINK(ipce#) = 0;
  IPC_PROCESS(ipce#) = TCP;
  IPC_DOMAIN(ipce#) = domain;
  IPC_DATA(ipce#) = message;
  IF 'PT_IPC_QUEUE_HEAD'(process#) = 0;
  THEN: PT_IPC_QUEUE_HEAD(process#) = ipce#;
  ELSE: Let eipce# = FINDIPCEND('PT_IPC_QUEUE_HEAD'(process#));
      IPC_LINK(eipce#) = ipce#;
  END;
  PT_IPC_QUOTA(process#) = 'PT_IPC_QUOTA'(process#) - 1;
  IF 'PT_IPC_WAIT'(process#) = NO;
  THEN: PT_IPC_WAIT(process#) = OFF;
  PT_FLAGS(process#) = READY;
END;
END;

Function: FINDIPCEND
Parameters: FINDIPCEND(ipce#)
Value:
IF IPC_LINK(ipce#) = 0;
THEN: ipce#;
ELSE: FINDIPCEND(IPC_LINK(ipce#));
END;

3.2.16.2 N/A

3.2.16.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCHECK</td>
<td>None</td>
</tr>
<tr>
<td>STOPP</td>
<td>None</td>
</tr>
</tbody>
</table>

82
3.2.16.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function IPCSEND. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

**Data Base References**

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT_FLAGS</td>
<td>PROCESS#</td>
<td>INDEX</td>
</tr>
<tr>
<td>PT_CUR_CLASS</td>
<td>MESSAGE</td>
<td>INDEX2</td>
</tr>
<tr>
<td>PT_CUR_CAT</td>
<td>DOMAIN</td>
<td></td>
</tr>
<tr>
<td>PT_IPC_QUOTA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT_IPC_QUEUE_HEAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS_CURRENT_PROCESS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPC_LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPC_PROCESS#</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPC_DATA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Constants**

| EXEC_PROCESS#    |
| INACTIVE         |
| IPC_WAIT         |
| PT_FLAGS_MASK    |
| READY            |

3.2.16.5 Limitations

If the PT_IPC_QUOTA for the process is zero, IPCSEND ignores the call. IPCSEND returns with no return code if PS_CURRENT_PROCESS is not equal to EXEC_PROCESS#, if PT_CURRENT_CLASS(PROCESS#) is less than PS_CUR_CLASS, if PT_CUR_CAT(PROCESS#) does not equal PT_CUR_CAT(PROCESS#) logical or PS_CUR_CAT, or if PT_FLAGS and PT_FLAGS_MASK equals INACTIVE.

3.2.16.6 Listing

```c
DATA IPCSEND(PROCESS#, MESSAGE, DOMAIN);
```
PROGRAM IPCSEND;
DECLARE
   WORD (INDEX, INDEX2);
IF (PT_FLAGS(PROCESS) & PT_FLAGS_MASK) = INACTIVE;
   THEN:
      RETURN;
   END;
   /* SECURITY CHECK */
IF PS_CURRENT_PROCESS = EXEC_PROCESS;
   THEN:
      IF PT_CUR_CLASS(PROCESS) < PS_CUR_CLASS;
         THEN:
            RETURN;
         END;
      IF PT_CUR_CAT(PROCESS) <= (PT_CUR_CAT(PROCESS) | PS_CUR_CAT);
         THEN:
            RETURN;
         END;
   END;
   /* IMPLEMENTATION CHECK */
IF PT_IPC_QUOTA(PROCESS) = 0;
   THEN:
      RETURN;
   END;
   /* ALLOCATE AN IPC QUEUE ELEMENT */
INDEX := IPC_LINK(0);
IPC_LINK(0) := IPC_LINK(INDEX);
IPC_LINK(INDEX) := 0;
   /* FILL IN IPC ELEMENT */
IPC_PROCE$$.(INDEX) := (PS_CURRENT_PROCESS | DOMAIN);
IPC_DATA(INDEX) := MESSAGE;
   /* IS PROCESS WAITING? */
IF (PT_IPC_QUEUE_HEAD(PROCESS) & BYTE_MASK) = IPC_WAIT;
   THEN: PT_IPC_QUEUE_HEAD(PROCESS) := INDEX;
      PT_FLAGS(PROCESS) := READY;
   ELSIF:
      IF PT_IPC_QUEUE_HEAD(PROCESS) = 0;
         THEN: PT_IPC_QUEUE_HEAD(PROCESS) := INDEX;
            ELSE: INDEX2 := PT_IPC_QUEUE_HEAD(PROCESS);
      CYCLE
         INDEX2 := IPC_LINK(INDEX2);
   END;
      IPC_LINK(INDEX2) := INDEX;
   END;
   /* ADJUST QUOTA */
PT_IPC QUOTA(PROCESS) := PT_IPC QUOTA(PROCESS) - 1;
84
3.2.17 Receive Interprocess Communication (IPCRCV)

The Receive Interprocess Communication CPC, IPCRCV, is a user level external SKCPP function that is called by user level external programs. IPCRCV calls only kernel level internal functions. It is written in Project SUE System Language.

3.2.17.1 Description

IPCRCV receives an interprocess communication message. If PT_IPC_QUEUE_HEAD(PR_CURRENT_PROCESS) equals $\emptyset$, there are no messages on queue. In this case, IPCRCV sets PT_IPC_QUEUE_HEAD(PS_CURRENT_PROCESS) to IPC_WAIT and PT_FLAGS(PS_CURRENT_PROCESS) to BLOCKED. This prevents future allocation of the processor to the current process. Then, to prevent a deadlock, it calls $V$ to increment the kernel semaphore and then calls SLEEP to find and execute the next process that is ready. When a message is available and the process is unblocked and running, it performs a $P$ on the kernel semaphore to restore its original value.

Now, to remove the IPC element from the head of queue, INDEX is set to PT_IPC_QUEUE_HEAD(PS_CURRENT_PROCESS) and PT_IPC_QUEUE_HEAD(PS_CURRENT_PROCESS) is set to IPC_LINK(INDEX).

IPCRCV can now use the information from the IPC message element. It assigns RC the sending process' and domain held in IPC_PROCESS# (INDEX) and KRC2 gets the message held in IPC_DATA(INDEX).

The IPC element is then put back on the free chain and the process' quota is credited. IPC_LINK(INDEX) is set equal to IPC_LINK($\emptyset$) and IPC_LINK($\emptyset$) is set to INDEX. IPCRCV then increments PT_IPC_QUOTA(PS_CURRENT_PROCESS) to conclude the operation.

Function: IPCRCV
Parameters: IPCRCV
Effect:
IF 'PT_IPC_QUEUE HEAD'(TCP) = 0;
THEN: PT_IPC_WAIT(TCP) = ON;
 PT_FLAGS(TCP) = BLOCKED;
 IPCRCV2;
ELSE: IPCUNQUEUE;
END;
Function: IPCUNQUEUE
Parameters: IPCUNQUEUE
Effect:
Let ipce# = 'PT_IPC_QUEUE_HEAD'(TCP);
PT_IPC_QUEUE_HEAD(TCP) = 'IPC_LINK'(ipce#);
RC(TCP) = IPC_PROCESS(ipce#), IPC_DOMAIN(ipce#), IPC_DATA(ipce#);
IPC_LINK(ipce#) = 'IPC_LINK'(0);
IPC_LINK(0) = ipce#;
PT_IPC_QUOTA(TCP) = 'PT_IPC_QUOTA'(TCP) + 1;

Function: IPCRCV2
Parameters: IPCRCV2
Effect:
IF 'PT_IPC_QUEUE_HEAD'(TCP) ≠ 0;
THEN: IPCUNQUEUE;
END;

3.2.17.2 N/A

3.2.17.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

 Called By       Calls

PCHECK           P
       V
       SLEEP

3.2.17.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function IPCRCV. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

Global References | Function Parameters | Local References

PT_IPC_QUEUE_HEAD | None | INDEX
PT_FLAGS
PT_IPC_QUOTA
PS_CURRENT_PROCESS
IPC_LINK
IPC_PROCESS#
IPC_DATA

86
Constants

BLOCKED
IPC_WAIT
KERNEL_SMFR

3.2.17.5 Limitations

None

3.2.17.6 Listing

DATA IPCCV RETURNS (PC);
DECLARE
    WORD (INDEX);

PROGRAM IPCCV;

    /* NO SECURITY CHECKING */
    /* ANYTHING THERE */
    IF PT_IPC_QUEUE_HEAD(PS_CURRENT_PROCESS) = 0;
        THEN: PT_IPC_QUEUE_HEAD(PS_CURRENT_PROCESS) := IPC_WAIT;
        PT_FLAGS(PS_CURRENT_PROCESS) := BLOCKED;
        V(KERNEL_SMFR);
        SLEEP;
        F(KERNEL_SMFR);
    END;

    /* REMOVE FIRST MESSAGE ELEMENT */
    INDEX := PT_IPC_QUEUE_HEAD(PS_CURRENT_PROCESS);
    PT_IPCQUEUE_HEAD(PS_CURRENT_PROCESS) := IPC_LINK(INDEX);

    /* TAKE STUFF OUT OF IPC MESSAGE ELEMENT */
    RC := IPC_PROCESS*(INDEX);
    KC2 := IPC_DATA(INDEX);

    /* PUT BACK ON FREE CHAIN AND INCREMENT QUOTA */
    IPC_LINK(INDEX) := IPC_LINK(0);
    IPC_LINK(0) := INDEX;
    PT_IPC_QUOTA(PS_CURRENT_PROCESS) := PT_IPC_QUOTA(PS_CURRENT_PROCESS) + 1;
3.2.18 Stop Process (STOPP)

The Stop Process CPC, STOPP, is a user level external SKCPP function that is called by user level external programs. STOPP calls both user level external functions and kernel level internal functions. It is written in Project SUE System Language.

3.2.18.1 Description

STOPP terminates a user's ownership of a process. It loops with I going from SEG#_MIN to SEG#_MAX, setting ASTE# equal to PS_SEG(I). Then, if ASTE# logical and SEG_FLAG equals 0, ASTE# indeed contains an AST entry number, so segment I is in the process's WS. If so, STOPP calls DCONNECT to release any segments to which the process has access.

It then sets I equal to PT_IPC_QUEUE_HEAD(PS_CURRENT_PROCESS). If I does not equal 0, the IPC queue must be cleared. To find the last element in the IPC queue, until IPC_LINK(I) equals 0, I is reset to IPC_LINK(I). The entire IPC queue is inserted at the head of the free element chain by letting IPC_LINK(I) equal IPC_LINK(0), IPC_LINK(0) equal PT_IPC_QUEUE_HEAD(PS_CURRENT_PROCESS), and PT_IPC_QUEUE_HEAD(PS_CURRENT_PROCESS) and PT_IPC_QUEUE_HEAD(PS_CURRENT_PROCESS) equal 0.

Next, STOPP prepares to remove the process's kernel stack. It sets ASTE# to PT_KS_ASTE#(PS_CURRENT_PROCESS) which holds the aste# of the process's kernel stack. It adds this to the chain of segments eligible to be swapped out by letting AST_SWAP_CHAIN(ASTE#) equal AST_SWAP_CHAIN(0), and letting AST_SWAP_CHAIN(0) equal ASTE#. It also resets AST_UNLOCK(ASTE#) to AST_UNLOCK(ASTE#) logical or AST_UNLOCK_FLAG.

In order to inform the executive process of the current process's termination, STOPP calls IPCSEND. It then sets PT_FLAGS(PS_CURRENT_PROCESS) to INACTIVE, performs a V on the kernel semaphore to prevent deadlocking, and calls SLEEP to allocate the processor to a ready process.

Function: STOPP
Parameters: STOPP(process#)
Effect:
(∀seg#)
IF (SEG#_MIN ≤ seg# ≤ SEG_MAX) &
PS_SEG_INUSE(process#, seg#);
THEN: DCONNECT(process#, PS_SEG(process#, seg#), seg#);
END;

88
IF 'PT IPC_QUEUE HEAD'(process#) ≠ 0
THEN: Let ipce# = FINDIPSCEND('PT IPC_QUEUE HEAD(process#));
IPC_LINK(ipce#) = 'IPC LINK'(0);
IPC_LINK(0) = 'PT IPC_QUEUE HEAD'(process#);
PT IPC_QUEUE HEAD(process#) = 0;
END:
PT_FLAG (process#) = INACTIVE;
IPCSEND(EXECUTIVE_PROCESS#,0,KERNEL_DOMAIN);
SLEEP;

3.2.18.2 N/A

3.2.18.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCHECK</td>
<td>DCONNECT</td>
</tr>
<tr>
<td></td>
<td>IPCSEND</td>
</tr>
<tr>
<td></td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>SLEEP</td>
</tr>
</tbody>
</table>

3.2.18.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function STOPP. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1 For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

<table>
<thead>
<tr>
<th>Data Base References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global References</td>
<td></td>
<td>Local References</td>
</tr>
<tr>
<td>PT_FLAGS</td>
<td>None</td>
<td>ASTE#</td>
</tr>
<tr>
<td>PT KS_ASTE#</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT IPC_QUEUE HEAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS_CURRENT_PROCESS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_SWAP_CHAIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_UNLOCK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPC_LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Constants            |                     |                 |
| AST_UNLOCK_FLAG      |                     |                 |
| EXEC_PROCESS#        |                     |                 |
| INACTIVE             |                     |                 |
| KERNEL_DOMAIN        |                     |                 |
3.2.18.5 Limitations

None

3.2.18.6 Listing

DATA STOPE:

PROGRAM STOPE;
DECLARE
  WORD (I, AST#, DUMMY);

  CLEAR OUT "B"
  DO I := SEG#_MIN TO SEG#_MAX;
      AST# := PS_SEG(I);
      IF (AST# & SEG_FLAG) = 0;
         THEN: DCONNECT(I, AST#);
      END;
  END;

  CLEAR OUT IPC QUEUE
  I := PT_IPC_QUEUE_HEAD(PS_CURRENT_PROCESS);
  IF I = 0;
     THEN:
        CYCLE
        .... EXIT WHEN IPC_LINK(I) = 0;
        I := IPC_LINK(I);
     END;

  IPC_LINK(I) := IPC_LINK(0);
  IPC_LINK(0) := PT_IPC_QUEUE_HEAD(PS_CURRENT_PROCESS);
  PT_IPC_QUEUE_HEAD(PS_CURRENT_PROCESS) := 0;

  GET Bid OF K STACK

  AST#: := PT_KS_ASTE(PS_CURRENT_PROCESS);
  AST_SWAP_CHAIN(AST#) := AST_SWAP_CHAIN(0);
  AST_SWAP_CHAIN(0) := AST#;
  AST_UNLOCK(AST#) := (AST_UNLOCK(AST#) | AST_UNLOCK_FLAG);

  LET EXEC KNOW WHAT'S HAPPENING

  IPCSEND(EXEC_PROCESS#, 0, KERNEL_DOMAIN);
  PT_FLAGS(PS_CURRENT_PROCESS) := INACTIVE;
  V(KERNEL_SMFR);
  SLEEP;
3.2.19 Read Directory (READIR)

The Read Directory CPC, READIR, is a user level external SKCPP function that is called by user level external programs with the parameters aste# and offset. READIR calls only kernel level internal functions. It is written in the Project SUE System Language.

3.2.19.1 Description

READIR gives interpretive read access to an entry in a directory in the process's address space.

If AST_TYPE(aste#) logical and AST_TYPE_MASK does not equal AST_TYPE_DIRECTORY the specified segment is not a directory, so READIR returns with ERR_FLAG.

Next, READIR must gain access to the directory. If AST_ADR (ASTE#) is zero, the directory is not present in main memory. SWAPIN is called to correct this situation. LSD is then invoked to load the segment descriptors. If DIR_SIZE(OFFSET) equals zero, READIR returns with ERR_FLAG; otherwise checking is complete and the data in the directory that is at the security level of the directory can be returned. CLASS, CAT, SEG_TYPE, and SIZE are assigned the value of DIR_CLASS(OFFSET) logical and DIR_CLASS_MASK, DIR_CAT(OFFSET), DIR_TYPE(OFFSET) logical and DIR_TYPE_MASK, and DIR_SIZE(OFFSET), respectively. READIR then regains access to the user's SR0 stack by assigning to KSDR3 and KSR3 the values of SDR0 and SAR0. It then inserts the data, letting CLASS_APARM, CAT_APARM, SEG_TYPE_APARM, and SIZE_APARM equal CLASS, CAT, SET_TYPE, and SIZE. READIR then returns with RC set to OK_FLAG.

Function: READIR
Parameters: READIR(process#,aste#,offset)
Effect:
IF (AST_TYPE(aste#) = DIRECTORY &
    DIR_SIZE(aste#, offset) ≠ 0
THEN: RC(process#) = DIR_TYPE(aste#, offset),
      DIR_CLASS(aste#, offset),
      DIR_CAT(aste#, offset),
      DIR_SIZE(aste#, offset);
ELSE: RC(process#) = NO;
END;

3.2.19.2 N/A
3.2.19.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

Called By          Calls
PCHECK
SWAPIN
LSD

3.2.19.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function READIR. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST_TYPE</td>
<td>ASTE#</td>
<td>CLASS</td>
</tr>
<tr>
<td>AST_ADR</td>
<td>OFFSET</td>
<td>CAT</td>
</tr>
<tr>
<td>DIR_CLASS</td>
<td></td>
<td>SEG_TYPE</td>
</tr>
<tr>
<td>DIR_CAT</td>
<td></td>
<td>SIZE</td>
</tr>
<tr>
<td>DIR_TYPE</td>
<td></td>
<td>RC</td>
</tr>
<tr>
<td>DIR_SIZE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLASS_ALARM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAT_APARM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEG_TYPE_APARM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIZE_APARM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDR &amp; SAR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constants

AST_TYPE_DIRECTORY  
AST_TYPE_MASK      
DIR_CLASS_MASK     
DIR_KSR_ADR        
DIR_TYPE           
DIR_TYPE_MASK      
ERR_FLAG           
OK_FLAG            
SDR_READ_ACCESS    

92
3.2.19.5 Limitations

READIR returns ERR_FLAG if the specified segment is not a directory or if the specified offset is not in main memory. Otherwise, RC = OK_FLAG.

3.2.19.6 Listing

DATA READIR(ASTE*, OFFSET) RETURNS (RC);

PROGRAM READIR;
DECLARE WORD (CLASS, CAT, SEG_TYPE, SIZE);
    /* IMPLEMENTATION CHECKS
    /* CHECK THAT SPECIFIED SEGMENT IS A DIRECTORY
    IF (AST_TYPE(ASTE#) & AST_TYPE_MASK) != AST_TYPE_DIRECTORY;
    THEN;
        .... RETURN WITH ERR_FLAG;
    END;
    /* GAIN ACCESS TO THE DIRECTORY
    IF AST_ADR(ASTE#) = 0;
        THEN: SWAPIN(ASTE#);
    END;
    LSD(ASTE#, DIR_KSR_ADDR, SCR_READ_ACCESS);
    /* CHECK THAT SPECIFIED OFFSET EXISTS
    IF DIR_SIZE(OFFSET) = 0;
        THEN:
            .... RETURN WITH ERR_FLAG;
    END;
    /* SAVE CLASS, CAT, SEG_TYPE, AND SIZE
    CLASS := DIR_CLASS(OFFSET) & DIR_CLASS_MASK;
    CAT := DIR_CAT(OFFSET);
    SEG_TYPE := DIR_TYPE(OFFSET) & DIR_TYPE_MASK;
    SIZE := DIR_SIZE(OFFSET);
    /* REGAIN ACCESS TO USERS SRO STACK AND INSERT DATA
    KSR3 := SDR0;
    KSAR3 := SAR0;
    CLASS_APARM := CLASS;
    CAT_APARM := CAT;
    SEG_TYPE_APARM := SEG_TYPE;
    SIZE_APARM := SIZE;
    RC := OK_FLAG;

93
3.2.20 **Start Process (STARTP)**

The Start Process CPC, STARTP, is a user level external SKCPP function that is called by only one user level external program, the Executive Process with the parameters process\# user, project, class, cat, proc_offset\#, and new_process\. STARTP calls both user level external functions and kernel level internal functions. It is written in Project SUE System Language, including the Inline feature.

3.2.20.1 **Description**

STARTP initializes a new process when invoked by the Executive Process. If PS\_CURRENT\_PROCESS does not equal EXEC\_PROCESS\#, it returns with ERR\_FLAG. Also, if PT\_FLAGS(PROCESS\#) logical and PT\_FLAGS\_MASK (the FLAGS and LINK entries share a byte) is not INACTIVE, it returns ERR\_FLAG.

STARTP then calls LSD to load the segment descriptors of the new process's process segment so the PS can be initialized. It sets PS\_CURRENT\_PROCESS to PROCESS\#, PS\_USER\_ID to USER, PS\_PROJECT\_ID to PROJECT, PS\_CUR\_CLASS and also PT\_CUR\_CLASS(PROJECT\#) to CLASS, PS\_CUR\_CAT and also PT\_CUR\_CAT(PROCESS\#) to CAT, PS\_MEM\_QUOTA to MEM\_QUOTA, and PT\_IPC\_QUOTA to IPC\_QUOTA. STARTP uses Inline code to find MASK and NOTMASK, it moves the process\# to register 3, decrements it, and negates it. Then, registers 0 and 1 are set to 400016 (all 0's except the second bit, bit 14) and BFFF\_16 (all 1's except the second bit), respectively. It then performs an ASH to shift arithmetically the contents of registers 0 and 1 N places, where N is the number in register 3. If N is positive, a left shift is performed, and the low order bits are filled in with 0's; if N is negative, a right shift is performed and bit 15 is replicated. Since register 3 contains 1 - PROCESS\#, the result in register 0 is a 1 in the bit corresponding to the process number, and 0's elsewhere, the leftmost bit representing process 0 and the rightmost, process 15. Register 1 contains the one's complement of register 0 except that for process 0 it contains 0's in both the leftmost and the rightmost bits. Register 0 is moved to PS\_PROCESS\_MASK and register 1 is moved to PS\_PROCESS\_NOTMASK, which are used in accessing AST\_CPL and AST\_WAL. STARTP then sets each PS\_SAR(I) and PS\_SDR(I) to 0, as I goes from 0 to 15. Also, with I starting at 0 until I equals SEG\# MAX, PS\_SEG(I) is set equal to I + 1 logical or SEG\_FLAG, placing all segment numbers on the free segment chain. Assigning PS\_SEG(SEG\# MAX) the value SEG\_FLAG marks the end of the free segment chain and completes the insertion of the process segment information.
STARTP then puts ROOT in the access space of the new process. It takes segment 1 from the free chain by letting PS_SEG(∅) = PS_SEG(1) and assigned to it ROOT_ASTE#. It then connects the new process to ROOT by resetting AST_CPL(ROOT_ASTE#) to AST_CPL(ROOT_ASTE#) logical or PS_PROCESS_MASK.

Now, access to user and kernel stacks are provided for the new process. GETR is called to gain read access to the process directory directory segment specified by ROOT_ASTE#, PDD_OFFSET; the seg# returned is assigned to PDD_SEG#. Next, read access is gained to the executive's process directory using GETR; similarly, the segment number returned is assigned to PD_SEG#. GETW is now called to provide write access to a user stack identified by an offset into the process directory of PROCESS#. STARTP assigns the seg# GETW returns to SS_SEG#. This segment is then ENABLEd. To get write access to the kernel stack is more difficult because GETW would fail. First, the segment descriptors of the process directory are loaded with LSD. STARTP then sets KS_ASTE# to the aste# of the PROCESS#_MAX + PROCESS# entry to the process directory: DIR_DISK holds the disk address of the entry and HASH converts this to an AST entry number. If KS_ASTE# is ∅, STARTP halts. Otherwise, the segment descriptors of the segment identified by KS_ASTE# are loaded, the AST_DES_COUNT(KS_ASTE#) is incremented, and PT_KS_ASTE#(PROCESS#) is set equal to KS_ASTE#.

STARTP now calls DCONNECT to release from the WS some intermediate directories, the process directory directory and the process directory. It invokes GETR to gain read access to the code directory identified by ROOT_ASTE#, CD_OFFSET and assigns its segment number to CD_SEG#. Next, it calls GETR to gain access to the segment, PS_SEG(CD_SEG#), PROC_OFFSET, and assigns its segment number to PROC_SEG#. A call to ENABLE places PROC_SEG#, which contains the new process's initial code segment, in the AS. The code directory, CD_SEG# can now be released by DCONNECT.

STARTP then calls LSD to switch back to the executive process by loading the descriptors of its process segment in the register at PS_KSR_ADR. LSD is called again to load the segment descriptors of the new process to give the executive process write access to its process segment.

Setting PT_R5(PROCESS#), a general register, equal to zero, PT_FLAGS(PROCESS#) to READY, and PT_IPC_QUEUE_HEAD to zero, completes the initialization of the new segment. STARTP returns with RC equal to OK_FLAG.
Function: STARTP
Parameters: STARTP(process#, user, project, class, cat, new_process#, proc_offset#);

Effect:

IF (process# \neq EXECUTIVE_PROCESS#) 1
   (PT_FLAGS(new_process#) \neq INACTIVE);
THEN:  RC(process#) = NO;
ELSE:  PS-USER_ID(new_process#) = user;
       PS_PROJECT_ID(new_process#) = project;
       PS_CLASS(new_process#) = class;
       PS_CAT(new_process#) = cat;
       PS_MEM_QUOTA(new_process#) = MEM_QUOTA;
       PS_IPC(new_process#) = IPC_QUOTA;
       (Vreg#)
       IF (REG#_MIX \leq reg# \leq REG#_MAX);
       THEN:  PS_SAR(new_process#, reg#) = 0;
               PS_SDR(new_process#, reg#) = 0;
       END;
       (Vseg#)
       IF (SEG#_MIX \leq seg# \leq SEG#_MAX);
       THEN:  PS_SEG_INUSE(new_process#, seg#) = FALSE;
               PS_SEG(new_process#, seg#) = (seg#+1)MODULO(SEG#_MAX+1);
       END;
       PS_SEG(new_process#, 0) = PS_SEG(new_process#, 1);
       PS_SEG(new_process#, 1) = ROOT_ASTE#;
       AST_CPL(ROOT_ASTE#, new_process#) = TRUE;
       GETR(new_process#, ROOT_ASTE#, PDD_OFFSET#);
       Let pdd_seg# = RC(new_process#);
       GETR(new_process#, PS_SEG(new_process#, pdd_seg#), EXECUTIVE_PROCESS#);
       Let pd_seg# = RC(new_process#);
       GETW(new_process#, PS_SEG(new_process#, pd_seg#), new_process#);
       LET stack_seg# = RC(new_process#);
       ENABLE(new_process#, PS_SEG(new_process#, stack_seg#), STACK_REG#);
       DCONNECT(new_process#, PS_SEG(new_process#, pdd_seg#), pdd_seg#);
       GETR(new_process#, ROOT_ASTE#, CD_OFFSET#);
       Let cd_seg# = RC(new_process#);
       GETR(new_process#, PS_SEG(new_process#, cd_seg#), proc_offset#);
       Let proc_seg# = RC(new_process#);
       ENABLE(new_process#, PS_SEG(new_process#, proc_seg#), PROC_REG#);

96
DCONNECT(new_process#, PS_SEG(new_process#, cd_seg#), cd_seg#);
PT_IPC_QUEUE_HEAD(new_process#) = 0;
PT_FLAGS(new_process#) = READY;

END;

3.2.20.2 N/A

3.2.20.3 Interfaces
Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCHECK</td>
<td>GETR</td>
</tr>
<tr>
<td></td>
<td>GETW</td>
</tr>
<tr>
<td></td>
<td>DCONNECT</td>
</tr>
<tr>
<td></td>
<td>ENABLE</td>
</tr>
<tr>
<td></td>
<td>HASH</td>
</tr>
<tr>
<td></td>
<td>LSD</td>
</tr>
</tbody>
</table>

3.2.20.4 Data Organization
Listed below are Security Kernel data base references and constants used by the function STARTP. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Local Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS_CURRENT_PROCESS</td>
<td>USER</td>
<td>PDD_SEG#</td>
</tr>
<tr>
<td>PS_USER_ID</td>
<td>PROJECT</td>
<td>PD_SEG#</td>
</tr>
<tr>
<td>PS_PROJECT_ID</td>
<td>CLASS</td>
<td>CD_SEG#</td>
</tr>
<tr>
<td>PS_CUR_CLASS</td>
<td>CAT</td>
<td>SS_SEG#</td>
</tr>
<tr>
<td>PS_CUR_CAT</td>
<td>PROCESS#</td>
<td>KS_ASTE#</td>
</tr>
<tr>
<td>PS_MEM_QUOTA</td>
<td>PROC_OFFSET</td>
<td>PROC_SEG#</td>
</tr>
<tr>
<td>PS_PROCESS_MASK</td>
<td></td>
<td>I</td>
</tr>
<tr>
<td>PS_PROCESS_NOTMASK</td>
<td></td>
<td>DUMMY</td>
</tr>
<tr>
<td>PS_SAR</td>
<td></td>
<td>RC</td>
</tr>
<tr>
<td>PS_SDR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS_SEG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS_FLAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT_FLAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT_IPC_QUOTA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

97
Global References  Local Parameters  Local References

PT_KSDR2  PT_R5  PT_PS_ASTE#  PT_KSDR1  PT_IPC_QUEUE_HEAD

Constants
ASHR0R3  ASHR1R3  CD_OFFSET  DEC  D1R_KSR_ADR  ERR_FLAG  EXEC_PROCESS#  INACTIVE  IPC_QUOTA  MEM_QUOTA  MOV  NEC  OK_FLAG  PDD_OFFSET  PROCESS#_MAX  PS_KSR_ADR  PT_FLAGS_MASK  PT_KDSR1_ADR  PT_KDSR2_ADR  READY  ROOT_ASTE#  SDR_READ_ACCESS  SEG_FLAGS  SEG#_MAX

3.2.20.5 Limitations

STARTP returns ERR_FLAG if PS_CURRENT_PROCESS does not equal EXEC_PROCESS# or if PT_FLAGS (PROCESS#) and PT_FLAGS_MASK are not INACTIVE. Otherwise, RC = OK_FLAG.

3.2.20.6 Listing

DATA STARTP(USER, PROJECT, CLASS, CAT, PROCESS#, PROC_OFFSET) RETURNS (RC);

PROGRAM STARTP;
DECLARE
  WORD (I, PDD_SEG#, PS_SEG#, CD_SEG#, SS_SEG#, KS_ASTE#, PROC_SEG#, DUMMY);
  /* ONLY EXECUTIVE CAN CALL THIS FUNCTION */
IF PS_CURRENT_PROCESS = EXEC_PROCESS#;
  THEN;
    ....
    RETURN WITH ERR_FLAG;
  END;

98
/* PROCESS HOST BE */

IF (PT_FLAGS(PROCESS#) & PT_FLAGS_MASK) =~ INACTIVE;
THEN:
....
RETURN WITH ERR_FLAG;
END;
/* MAKE "PARTIAL" SWITCH TO USER PROCESS */

LSD(PT_PS_ASTF#(PROCESS#), PS_KSP_ADR, SDR_WHITE_ACCESS);
/* INITIALIZE PS */

PS_CURRENT_PROCESS := PROCESS#;
/* NEED MASK + NOTMASK */

INLINE(MOV, PROCESS#, 0, 3);
INLINE(DEC, 0, 3);
INLINE(MOV, 2, 7, 0, 0, "4000");
INLINE(MOV, 2, 7, 0, 1, "OFF");
INLINE(ASSH, 0);
INLINE(ASSH, 3);
INLINE(MOV, 0, 0, PS_PROCESS_MASK);
INLINE(MOV, 0, 1, PS_PROC_NSMASK);
PS_USER_ID := USER;

PS_PROJECT_ID := PROJECT;
PS_CUR_CLASS := CLASS;
PT_CUR_CLASS(PROCESS#) := CLASS;

PS_CUR_CAT := CAT;

PT_CUR_CAT(PROCESS#) := CAT;

PS_IPC_QUOTA(PROCESS#) := IPC_QUOTA;

DO I := 0 TO 15;
    PS_SAR(I) := 0;
    PS_SDR(I) := 0;
END;

DO I := 0 TO SEG#_MAX;
    PS_SEG(I) := ((I + 1) | SEG_FLAG);
END;

PS_SEG(SEG#_MAX) := SEG_FLAG;
/* PUT ROOT INTO "0" */

PS_SEG(0) := PS_SEG(1);
PS_SEG(1) := ROOT_ASTF#;

AST_CPL(ROOT_ASTF#) := (AST_CPL(ROOT_ASTF#) | PS_PROCESS_MASK);
/* GAIN ACCESS TO STACKS */
/* FIRST PDD */

PDD_SEG# := GETR(ROOT_ASTF#, PDD_OFFSET);
/* NEXT EXEC'S PD */

PD_SEG# := GETP(PS_SEG(PDD_SEG#), EXEC_PROCESS#);
/* NOW STACKS - FIRST 5 STACK */

SS_SEG# := GETW(PS_SEG(PD_SEG#), PROCESS#);
DUMMY := FRABLE(PS_SEG(SS_SEG#), 0);
3.2.21 Change Object (CHANGEO)

The Change Object CPC, CHANGEO is a user level external SKCPP function that is called by one user level external program, the executive process, with the parameters process#, aste#, offset, class and cat. CHANGEO calls only kernel level internal functions. It is written in Project SUE System Language.

3.2.21.1 Description

CHANGEO alters the classification and category of a data segment according to the specification of a trusted subject. First, it checks that the calling process is indeed trusted. If PS_CURRENT_PROCESS is not EXEC_PROCESS#, ERR_FLAG is returned. Also, if WRITEDIR(ASTE#) does not return OK_FLAG, CHANGEO returns ERR_FLAG: the process must have write access to the parent segment, which must be a directory. The implementation requirement is that the segment whose attributes are to be changed must exist; if DIR_SIZE(OFFSET) is zero, it returns ERR_FLAG. Next, CHANGEO must insure that the segment is not in the WS of any process. It sets OASTE# to the aste# which HASH associates with DIR_DISK(OFFSET), the disk address of the segment. If OASTE# is Ø, the segment is inactive and by
definition is not in the WS of any process. Also if AST_CPL logical and WIRED_DOWN_NOTMASK - the wired down bit shares a word with the CPL equals \( \emptyset \), it is not connected to any process. If neither of these conditions holds, some process has the segment in its WS, so the segments attributes cannot be changed: ERR_FLAG is returned. Also if DIR_TYPE(OFFSET) and DIR_TYPE_MASK equals DIR_TYPE_DIRECTORY, CHANGEO returns with ERR_FLAG. Finally, the compatibility rule that security levels must be nondecreasing as one moves down in the hierarchy is implemented. If CLASS is less than AST_CLASS(ASTE#) logical and AST_CLASS_MASK, the classification of the parent, or if the CAT set does not equal logical or AST_CAT(ASTE#), CHANGEO returns with ERR_FLAG.

Otherwise, checking is complete. DIR_CLASS(OFFSET) is reset to the logical or of CLASS and the logical and of DIR_CLASS(OFFSET) and DIR_CLASS_NOTMASK, changing the class without affecting the type and status bits. DIR_CAT(OFFSET) is assigned the value of CAT. If the segment is active, that is, if OASTE// is not equal to zero, ASTE_CLASS (OASTE#) and AST_CAT(OASTE#) must be changed also; they are set equal to DIR_CLASS(OFFSET) and DIR_CAT(OFFSET). RC is set to OK_FLAG and CHANGEO returns.

Function CHANGEO
Parameters: CHANGEO(process#,aste#,offset,class,cat);
Effect:
IF (PS_TYPE(process#) ≠ TRUSTED) | not AST_WAL(aste#,process#) | (Ast_TYPE(aste#) ≠ DIRECTORY) | (DIR_SIZE(aste#,offset) = 0) | (HASH(DIR_DISK(aste#,offset#)) ≠ 0) | (DIR_TYPE(aste#,offset#) ≠ DIRECTORY) | (cat ≠ AST_CAT(aste#)) | (class < AST_CLASS(aste#));
THEN: RC(process#) = NO;
ELSE: DIR_CLASS(aste#) = class;
   DIR_CAT(aste#) = cat
   RC(process#) YES;
END;

3.2.21.2 N/A

3.2.21.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.
3.2.21.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function CHANGEO. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Local Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS_CURRENT_PROCESS</td>
<td>ASTE#</td>
<td>OASTE#</td>
</tr>
<tr>
<td>DIR_SIZE</td>
<td>OFFSET</td>
<td>RC</td>
</tr>
<tr>
<td>DIR_TYPE</td>
<td>CLASS</td>
<td></td>
</tr>
<tr>
<td>DIR_CLASS</td>
<td>CAT</td>
<td></td>
</tr>
<tr>
<td>DIR_CAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_CPL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_CLASS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_CAT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constants

<table>
<thead>
<tr>
<th>AST_CLASS_MASK</th>
<th>ERR_FLAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIR_CLASS_NOTMASK</td>
<td>EXEC_PROCESS#</td>
</tr>
<tr>
<td>DIR_TYPE</td>
<td>INACTIVE</td>
</tr>
<tr>
<td>DIR_TYPE_DIRECTORY</td>
<td>OK_FLAG</td>
</tr>
<tr>
<td>DIR_TYPE_MASK</td>
<td>WIRED_DOWN_MASK</td>
</tr>
</tbody>
</table>

3.2.21.5 Limitations

CHANGEO returns ERR_FLAG if PS_CURRENT_PROCESS does not equal EXEC_PROCESS#, if the intended parent segment is a directory to which the process does not have write access, if the segment does not exist in main memory, if the segment is in the WS of some process, if the segment is a directory, or if the category set is less than classification of the parent segment. Otherwise, RC = OK_FLAG.

3.2.21.6 Listing

DATA CHANGEO (ASTE#, OFFSET, CLASS, CAT) RETURNS (RC);
PROGRAM CHANGED;
DECLARE
    WORD (OAST#);

    /* SECURITY CHECKS
    /* ONLY TRUSTED SUBJECTS CAN USE THIS FUNCTION
    IF PS_CURRENT_PROCESS -> EXEC_PROCESS;
        THEN:
            RETURN WITH ERR_FLAG;
        END;

    /* INTERPRETIVE DIRECTORY WRITE CHECK
    IF WRIITEDIR (ASTE) -> OK_FLAG;
        THEN:
            RETURN WITH ERR_FLAG;
        END;

    IF DIR_SIZE (OFFSET) = 0;
        THEN:
            RETURN WITH ERR_FLAG;
        END;

    /* IF OBJECT IS INACTIVE THEN NO NEED TO DO SECURITY & -PROPERTY CHECK
    OASTE := HASH (DIR_DISK (OFFSET));
    IF (OASTE = 0) & (AST_CPL (OASTE) & WIPED_DOWN_NOTHASK = 0);
        THEN:
            RETURN WITH ERR_FLAG;
        END;

    /* COMPATIBILITY CHECK SIMPLIFIED IF OBJECT IS NOT A DIRECTORY
    IF (DIR_TYPE (OFFSET) & DIR_TYPE_MASK) = DIR_TYPE_DIRECTORY;
        THEN:
            RETURN WITH ERR_FLAG;
        END;

    IF CLASS < (AST_CLASS (ASTE) & AST_CLASS_MASK);
        THEN:
            RETURN WITH ERR_FLAG;
        END;

    IF CAT = (AST_CAT (ASTE) | CAT);
        THEN:
            RETURN WITH ERR_FLAG;
        END;

    /* CHECKING COMPLETE - PERFORM STATE CHANGE
    DIR_CLASS (OFFSET) := (DIR_CLASS (OFFSET) & DIR_CLASS_NOTHASK) | CLASS;
    DIR_CAT (OFFSET) := CAT;

    /* IF WIPED_DOWN CHANGF ASTE CLASS AND CAT ALSO
    IF OASTE = 0;
        THEN: AST_CLASS (OASTE) := DIR_CLASS (OFFSET);
            AST_CAT (OASTE) := CAT;
        END;

    RC := OK_FLAG;
3.2.22 Initialize Hierarchy (INITH)

The Initialize Hierarchy CPC, INITH, is a user level external SKCPP function that is called by only one user level program, the executive process. INITH calls only kernel level internal functions. It is written in Project SUE System Language.

3.2.22.1 Description

INITH allows the executive process to set up the directory structure at system initialization: it copies the attributes of a specified ASTE into a specified directory entry.

It checks that the calling process is trusted and that the directory aste# parameter supplied identifies a directory to which the process has write access. If PS_CURRENT_PROCESS does not equal EXEC_PROCESS# or if WRITEDIR (DASTE#) does not return ERR_FLAG, INITH returns with ERR_FLAG. It also requires that the directory entry specified be empty: if DIR_SIZE(OFFSET) is not zero, it returns with ERR_FLAG.

If the security and implementation requirements have been satisfied, it performs the state change. DIR_CLASS(OFFSET), DIR_CAT(OFFSET), DIR_DISK(OFFSET) and DIR_SIZE(OFFSET) are assigned, respectively, the values of AST_CLASS(ASTE#), AST_DISK, and AST_SIZE(ASTE#). Setting DIR_ACL_HEAD(OFFSET) to zero empties the access control list and completes the procedure. INITH returns with RC set equal to OK_FLAG.

Function: INITH
Parameters: INITH(process#, daste#, offset#, aste#)
Effect:
IF  (PS_TYPE(process#) not TRUSTED
(not AST_WAL(daste#))  
(AST_TYPE(daste#) not DIRECTORY)
(DIR_SIZE(daste#, offset#) 0;
THEN: RC(process#) = NO;
ELSE: DIR_CLASS(daste#, offset#) = AST_CLASS(aste#);  
DIR_CAT(daste#, offset#) = AST_CAT(aste#);  
DIR_DISK(daste#, offset#) = AST_DISK(aste#);  
DIR_SIZE(daste#, offset#) = AST_SIZE(aste#);  
DIR_ACL_HEAD(daste#, offset#) = 0;  
RC(process#) = YES
END;

3.2.22.2 N/A

104
3.2.22.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCHECK</td>
<td>WRITEDIR</td>
</tr>
</tbody>
</table>

3.2.22.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function INITH. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Functional Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS_CURRENT_PROCESS</td>
<td>DASTE#</td>
<td>RC</td>
</tr>
<tr>
<td>DIR_CLASS</td>
<td>OFFSET</td>
<td></td>
</tr>
<tr>
<td>DIR_CAT</td>
<td>ASTE#</td>
<td></td>
</tr>
<tr>
<td>DIR_SIZE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR_DISK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constants

- ERR_FLAG
- EXEC_PROCESS#
- OK_FLAG

3.2.22.5 Limitations

INITH returns ERR_FLAG if PS_CURRENT_PROCESS does not equal EXEC_PROCESS#, if the directory entry specified is not empty, or if the intended parent segment is a directory to which the process does not have write access. Otherwise, RC=OK_FLAG.

3.2.22.6 Listing

```c
DATA INITH(DASTE#, OFFSET, ASTE#) RETURNS (RC);
```
/* SECURITY CHECKS */

IF PS_CURRENT_PROCESS != EXEC_PROCESS#
THEN:
   RETURN WITH ERR_FLAG;
END;

/* INTERPRETIVE DIRECTOY WRITE CHECK */

IF WRITEDIR(DASTE#) == OK_FLAG;
THEN:
   RETURN WITH ERR_FLAG;
END;

/* IMPLEMENTATION CHECKS */

IF DIR_SIZE(OFFSET) == 0;
THEN:
   RETURN WITH ERR_FLAG;
END;

/* PERFORM STATE CHANGE - COPY ASTE ATTRIBUTES INTO DIRECTOY ENTRY */

DIP_CLASS(OFFSET) := AST_CLASS(ASTE#);
DIP_CAT(OFFSET) := AST_CAT(ASTE#);
DIP_DISK(OFFSET) := AST_DISK(ASTE#);
DIP_SIZE(OFFSET) := AST_SIZE(ASTE#);
DIP_ACL_HEAD(OFFSET) := 0;
RC := OK_FLAG;

3.2.23 Get Directory (GETDIR)

The GET Directory CPC, GETDIR, is a kernel level internal SKCPP function that is called by a user level external function. GETDIR calls only kernel level internal functions. It is written in Project SUE System Language.

3.2.23.1 Description

GETDIR insures that a directory segment is swapped into main memory and can be accessed. If AST_ADR(ASTE#) is zero, the ASTE# supplied identifies a segment not in main memory; SWAPIN is called to swap the segment in. LSD is then called to load the segment descriptors in the register at DIR_KSR_ADR to provide write access to the segment.
Function: GETDIR
Parameters: GETDIR(aste#)

IF AST_ADR - 0
THEN: SWAPIN;
LSD;
AST_CHANGED;
ELSE: LSD;
    AST_CHANGED:
END;

3.2.23.2 N/A

3.2.23.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>SWAPIN LSDD</td>
</tr>
</tbody>
</table>

3.2.23.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function GETDIR. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST_ADR</td>
<td>ASTE#</td>
<td>None</td>
</tr>
</tbody>
</table>

Constants

| DIR_KSR_ADR       |
| SDR_WRITE_ACCESS  |

3.2.23.5 Limitations

None.

3.2.23.6 Listing

DATA GETDIR(aste#):
3.2.24 Write Directory (WRITEDIR)

The Write Directory CPC, WRITEDIR, is a kernel level internal SKCPP that is called by user level external functions. WRITEDIR calls kernel level internal functions. It is written in Project SUE System Language.

3.2.24.1 Description

WRITEDIR makes security and implementation checks, and if constraints are met, provides access to a specified directory. It returns ERR_FLAG if AST_TYPE(ASTE#) logical and AST_TYPE_MASK does not equal AST_TYPE_DIRECTORY: that is, if the segment supplied is not a directory. If AST_WAL(ASTE#) logical and PS_PROCESS_MASK, which contains all 0's except a 1 in the bit corresponding to the process#, is zero, it returns with ERR_FLAG.

If these requirements are satisfied, it makes certain that the segment is in main memory. If AST_ASR(ASTE#), which holds the main memory address of the segment, is zero, WRITEDIR calls SWAPIN to swap the segment into main memory. Next, it calls LSD, which loads the descriptors of the directory in the register located at DIR_KSR_ADR with access mode SDR_WRITE_ACCESS. Since write access to the directory has been gained, the directory will be changed, so, to insure that the segment will be copied onto the disk when it is swapped out, the change bit is set. WRITEDIR lets AST_CHANGE(ASTE#) equal AST_CHANGE(ASTE#) logical or AST_CHANGED. Setting RC equal to OK_FLAG, it returns.

Function: WRITEDIR
Parameters: WRITEDIR(aste#)
Effect:
IF not (AST_TYPE(aste#) # DIRECTORY)
   (AST_WAL(aste#) # 0)
THEN: RC(TCP) = NO;
ELSE: RC(TCP) = YES
    IF AST_ADR = 0;
        THEN: SWAPIN:
            LSD;
            AST_CHANGED;
        ELSE: LSD;
            AST_CHANGED;
        END;
    END;
END:

3.2.24.2. N/A

3.2.24.3. Interfaces

Refer to Figure 6, Function Call Matrix, in Paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>SWAPIN</td>
</tr>
<tr>
<td>DELETE</td>
<td></td>
</tr>
<tr>
<td>GIVE</td>
<td>LSD</td>
</tr>
<tr>
<td>RESCIND</td>
<td></td>
</tr>
<tr>
<td>CHANGO</td>
<td></td>
</tr>
<tr>
<td>INITH</td>
<td></td>
</tr>
</tbody>
</table>

3.2.24.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function WRITEDIR. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS_PROCESS_MASK</td>
<td>ASTE#</td>
<td>RC</td>
</tr>
<tr>
<td>AST_ADR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_TYPE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_WAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_CHANGE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constants

AST_CHANGE

109
3.2.24.5 Limitations

WRITEDIR returns ERR_FLAG if the segment supplied is not a directory, or if the process does not have write access to the directory. Otherwise, RC = OK_FLAG.

3.2.24.6 Listing

DATA WRITEDIR(ASTE#) RETURNS (RC);

PROGRAM WRITEDIR;

/
\* CHECKS THAT SPECIFIED SEGMENT IS A DIRECTORY AND THAT IT IS IN CURRENT PROCESS'S WRITE MODE
\*/
IF (AST_TYPE(ASTE#) & AST_TYPE_MASK) != AST_TYPE_DIRECTORY;
    THEN:
        RETURN WITH ERR_FLAG;
    END;

IF (AST_WAL(ASTE#) & PS_PROCESS_MASK) = 0;
    THEN:
        RETURN WITH ERR_FLAG;
    END;

IF AST_ADR(ASTE#) = 0;
    THEN: SWAPIN(ASTE#);
    END;

/*
GAIN ACCESS TO THE DIRECTORY
*/
LSD(ASTE#, DIR_KSR_ADR, SDR_WRITE_ACCESS);

/*
DIRECTORY WILL BE CHANGED - SET CHANGE BIT
*/
AST_CHANGE(ASTE#) := (AST_CHANGE(ASTE#) | AST_CHANGED);
RC := OK_FLAG;

3.2.25 Delete Segment (DELETSEG)

The Delete Segment CPC, DELETSEG, is a kernel level internal SKCPP function that is called by a user level external function. DELETSEG calls only kernel level internal functions. It is written in Project SUE System Language.
3.2.25.1 Description

DELETSEG deletes a data segment or an empty directory segment. First, it removes any elements that may be on the ACL for the segment. It sets INDEX equal to DIR_ACL_HEAD(OFFSER). If INDEX is not zero, there are some ACL elements to be removed. Until ACL_CHAIN(INDEX), which holds the acle# of the next element in the list, equals zero, INDEX is reset to ACL_CHAIN(INDEX). ACL_CHAIN(INDEX) is then set to ACL_CHAIN(∅), linking the end of the ACL to the head of the free chain, and ACL_CHAIN(∅), the pointer to the free acle chain, is set to DIR_ACL_HEAD(OFFSET). Setting DIR_ACL_HEAD(OFFSET) to ∅ completes the transfer of the ACL to the free chain.

DELETSEG then calls SOADD, which removes the segment from the WS of any process whose access rights have been rescinded. Since the segment's ACL has just been emptied, SOADD removes the segment from all WS's.

If the segment to be deleted is active (aged, now), it must be deactivated. When HASH is called with a parameter of DIR_DISK (OFFSET), which contains the disk address of the segment, it returns the aste# of the segment. This value is assigned to OASTE#. If OASTE# is non-zero, the segment is active and eligible for deactivation. The change and status bits are zeroed by letting AST_CHANGE (OASTE#) equal AST_CHAIN(OASTE#) logical and AST_UNCHANGED_MASK and AST_STATUS(OASTE#) equal AST_STATUS(OASTE#) logical and AST_STATUS_NOTMASK. DEACT is then called to swap the segment out of main memory, if necessary, and move its aste from the list of segments eligible for deactivation to the list of free aste's.

DELETSEG then calls DFREE to free the disk space occupied by the segment. It sets DIR_DISK(OFFSET) and DIR_SIZE(OFFSET) to zero to mark the directory entry free. Finally, DELETSEG sets the change bit in the parent directory so it will be copied onto the disk when swapped out of main memory: AST_CHANGE(ASTE#) is set equal to AST_CHANGE(ASTE#) logical or AST_CHANGED. It then returns control to the calling program.

Function: DELETSEG
Parameters: DELETSEG(aste#, offset)
Effect:
IF 'DIR_ACL_HEAD'(aste#, offset) ≠ 0;
THEN: Let acle# = FINDEND(aste#, 'DIR_ACL_HEAD'(aste#, offset));
     ACL_CHAIN(aste#, acle#) = 'ACL_CHAIN'(aste#, 0);
ACL_CHAIN(aste#, 0) = 'DIR_ACL_HEAD'(aste#, offset);
SOADD(aste#, offset);
END;
IF HASH('DIR_DISK'(aste#, offset)) ≠ 0;
THEN: DEACTIVATE(HASH('DIR_DISK'(aste#, offset)));
END;
DISK_FREE('DIR_DISK'(aste#, offset, 'DIR_SIZE'(aste#, offset)));
DIR_SIZE(aste#, offset) = 0;

3.2.25.2 N/A

3.2.25.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>SOADD</td>
</tr>
<tr>
<td></td>
<td>DFREE</td>
</tr>
<tr>
<td></td>
<td>DEACT</td>
</tr>
<tr>
<td></td>
<td>HASH</td>
</tr>
</tbody>
</table>

3.2.25.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function DELETSEG. For data base references refer to Figure 5, Data Base References Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

<table>
<thead>
<tr>
<th>Data Base References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global References</td>
<td>Function Parameters</td>
<td>Local References</td>
</tr>
<tr>
<td>ACL_CHAIN</td>
<td>ASTE#</td>
<td>OASTE#</td>
</tr>
<tr>
<td>ACL_CHANGE</td>
<td></td>
<td>INDEX</td>
</tr>
<tr>
<td>DIR_ACL_HEAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR_DISK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR_SIZE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constants</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_CHANGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_STATUS_NOTMASK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_UNCHANGED_MASK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BMT_SIZE2_ADR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERR_FLAG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.2.25.5 Limitations

None

3.2.25.6 Listing

DATA DELETSEG(ASTE#, OFFSET);
DECLARE
PROCEDURE ACCEPTS (WORD, WORD) (DFREE);

PROGRAM DELETSEG;
DECLARE
WORD (INDEX, OASTE#);

REMovo ANY ELEMENTS THAT MAY BE ON ACL CHAIN
INDEX := DIR_ACL_HEAD(OFFSET);
IF INDEX ^= 0;
THEN:
    CYCLE
    EXIT WHEN ACL_CHAIN(INDEX) = 0;
    INDEX := ACL_CHAIN(INDEX);
END;
ACL_CHAIN(INDEX) := ACL_CHAIN(0);
ACL_CHAIN(O) := DIR_ACL_HEAD(OFFSET);
END;

NOW BUMP EVERYBODY OFF
STADD(ASTE#, OFFSET);

DEACTIVATE IF ASTE# OF OFFSET (OASTE#) IS AGED
OASTE# := HASH(DIR_DISK(OFFSET));
IF OASTE# ^= 0;
THEN:
    SET CHANGE BIT TO UNCHANGED AND STATUS BIT TO INITIALIZED
    AST_CHANGE(OASTE#) := (AST_CHANGE(OASTE#) & AST_UNCHANGED_MASK);
    AST_STATUS(OASTE#) := (AST_STATUS(OASTE#) & AST_STATUS_NOTMASK);
    DEACT(OASTE#);
END;
FREE UP RESOURCES - DISK SPACE AND DIRECTORY ENTRY
DFREE(DIR_DISK(OFFSET), BMT_SIZE2_ADDR);
DIR_DISK(OFFSET) := 0;
DIR_SIZE(OFFSET) := 0;

SET CHANGE BIT IN PARENT
AST_CHANGE(ASTE#) := (AST_CHANGE(ASTE#) | AST_CHANGED);
3.2.26 Connect (CONNECT)

The Connect CPC, CONNECT, is a kernel level internal SKCPP function that is called by user level external functions. CONNECT calls only kernel level internal functions. It is written in Project SUE System Language.

3.2.26.1 Description

CONNECT connects a process to a segment, putting the segment in the WS of the process. It is subject to two implementation constraints. It sets SEG# to PS_SEG(0) logical and SEG_MASK. If SEG# equals zero, there are no free segment numbers, and CONNECT returns with ERR_FLAG. It then insures that the segment is active. It calls HASH which returns the AST entry number associated with DIR_DISK(OFFSET), the disk address of the segment, and assigns this value to ASTE#. If ASTE# is zero, CONNECT must call ACT to activate the segment, and ASTE# is set to the value it returns. If ASTE# is non-zero, showing that the segment is already active, and AST_CPL (ASTE#) logical and PS_PROCESS_MASK is non-zero, the process is already connected to the segment so CONNECT returns with ERR_FLAG.

Otherwise, it can proceed with the connection. If the segment is eligible for deactivation it must be removed from the age chain. If AST_CPL(ASTE#) is 0, this is the case. To find the segment's AST entry number in the age chain, it sets INDEX to 0 and NEXT to AST_AGE_CHAIN(0), which holds the head of the age chain. Then, until it has set NEXT to ASTE#, it resets INDEX and NEXT to AST_AGE_CHAIN(INDEX). Assigning to AST_AGE_CHAIN(INDEX) the value of AST_AGE_CHAIN(ASTE#) and setting AST_AGE_CHAIN(ASTE#) to 0 removes the segment from the chain.

CONNECT now performs the actual connection by resetting AST_CPL(ASTE#) to AST_CPL(ASTE#) logical or PS_PROCESS_MASK. Since PS_PROCESS_MASK consists of all 0's except for a 1 in the bit corresponding to the process#, this sets the CPL bit for the process. If the MODE is WRITE$READ$EXECUTE_ACCESS, the WAL bit for the process must also be set. CONNECT accomplishes this by resetting AST_WAL (ASTE#) to AST_WAL(ASTE#) logical or PS_PROCESS_MASK.

To complete the procedure, PS_SEG(0) is reset to PS_SEG(SEG#), which removes SEG# from the free segment chain, and PS_SEG(SEG#) is set to ASTE#. This allows segment numbers to be mapped onto AST entry numbers, which is necessary because ASTE numbers are system-wide variables to whose values users can not have access. CONNECT returns with RC set to the SEG# with which the user can subsequently refer to the segment.
Function: CONNECT
Parameters: CONNECT(process#, daste#, entry#, mode)

Effect:
IF 'PS_SEG'(process#, 0) = 0;
THEN: RC(process#) = NO;
ELSE: Let flag = 'HASH'(DIR_DISK(daste#, entry#));
IF (flag ≠ 0) & 'AST_CPL'(flag, process#);
THEN: RC(process#) = NO;
ELSE:
IF flag ≠ 0;
THEN: Let aste# = flag;
   IF 'AST_AGE' (aste#) = AGED;
   THEN: UNAGE(aste#);
   END;
ELSE: ACTIVATE(daste#, entry#);
   Let aste# = HASH(DIR_DISK(daste#, entry#));
   UNAGED(aste#);
END;
AST_CPL(aste#, process#) = TRUE;
IF mode = WRITE;
THEN: AST_WAL(aste#, process) = TRUE;
END;
Let seg# = 'PS_SEG'(process#, seg#);
PS_SEG(process#, 0) = 'PS_SEG'(process#, seg#);
PS SEG(process#, seg#) = aste#;
PS_SEG_INUSE(process#, seg#) = TRUE;
RC(process#) = YES, seg#;
END;
END;

3.2.26.2 N/A

3.2.26.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETW</td>
<td>ACT</td>
</tr>
<tr>
<td>GETR</td>
<td>HASH</td>
</tr>
</tbody>
</table>

3.2.26.4 Data Organization

Listed below are Security Kernel data base references and constants used by function CONNECT. For data base references
refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1.
For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS_SEG</td>
<td>DASTE#</td>
<td>INDEX</td>
</tr>
<tr>
<td>PS_PROCESS_MASK</td>
<td>OFFSET</td>
<td>NEXT</td>
</tr>
<tr>
<td>AST CPC</td>
<td>MODE</td>
<td>SEG#</td>
</tr>
<tr>
<td>AST AGE_CHAIN</td>
<td></td>
<td>ASTE#</td>
</tr>
<tr>
<td>AST WAL</td>
<td></td>
<td>HASH_VAL</td>
</tr>
<tr>
<td>DIR_DISK</td>
<td></td>
<td>RC</td>
</tr>
</tbody>
</table>

Constants

ERR_FLAG
OFFSET_MAX
OFFSET_MIN
SEG_MASK
WRITE$READ$EXECUTE_ACCESS

3.2.26.5 Limitations

CONNECT returns EFF_FLAG if there are no free segment numbers or if the process is already connected to the segment. Otherwise, RC = SEG#.

3.2.26.6 Listing

DATA CONNECT(DASTE#, OFFSET, MODE) RETURNS (RC);

PROGRAM CONNECT;
DECLARE
    WORD (INDEX, NEXT, SEG#, ASTE#, HASH_VAL);
    /* FIND A FREE SEG# */
    SEG# := (PS_SEG(0) & SEG_MASK);
    IF SEG# = 0;
    THEN:
        .... RETURN WITH ERR_FLAG;
    END;
    /* DETERMINE IF SEGMENT IS ACTIVE */
    ASTE# := HASH(DIR_DISK(OFFSET));

116
3.2.27 Search Out and Destroy Descriptors (SOADD)

The Search Out and Destroy Descriptors CPC, SOADD, is a kernel level internal SKCPP function that is called by both user level external functions and kernel level internal functions. SOADD calls both user level external functions and kernel level internal functions. It is written in Project SUE System Language.

3.2.27.1 Description

SOADD searches out and destroys descriptors for a segment in processes whose access rights have been restricted. It checks that the segment is active - if it is not, no processes are connected to it at all. It calls HASH to find the aste# associated with the segment's disk address, which is held in DIR_DISK(OFFSET). This value it assigns to ASTE#. Since HASH returns a zero if the segment is active, and since the CPL contains 1's in the bits corresponding to connected processes, if ASTE# and AST_CPL(ASTE#) are both non-zero, SOADD must proceed. Otherwise, no processes have descriptors for the segment and the call is ignored.
It then loops through all processes, PROCESS#, going from PROCESS#_MIN to PROCESS#_MAX. If PT_FLAGS(PROCESS#) logical and PT_FLAGS_MASK (FLAGS and LINK entries share a byte) equal inactive, there is no need to check the process. If it is not INACTIVE, SOADD must ensure that it is not connected to the segment without adequate access rights. To find out, it needs access to the process's process segment (PS). Hence, LSD is called to load the descriptors of the process segment, identified by PT_PS_ASTE#(PROCESS#) into the kernel register 2 at PS_KSR_ADR.

SOADD then determines whether the process is connected to the segment, and if so, in what mode of access. If AST_CPL(ASTE#) logical and PS_PROCESS_MASK, which contains a single 1 in the bit corresponding to the PROCESS#, is non-zero, the CPL bit for the process is set. If it isn't, SOADD continues looping through the processes. If it is, SOADD checks if the process is also on the segment's write access list - if AST_WAL(ASTE#) logical and PS_PROCESS_MASK are zero, the process has read access only, and MODE is set to READ$EXECUTE_ACCESS.

SOADD then calls DSEARCH which searches the ACL of segment DASTE#, OFFSET to see if access MODE is still permitted. If DSEARCH returns OK_FLAG, the MODE of access is permitted; if DSEARCH returns ERR_FLAG, the MODE is no longer permitted, and SOADD destroys the descriptor. To do this, it must find the seg# of the segment: it loops through all the seg#'s from SEG#_MIN to SEG#_MAX until it finds the one which corresponds to the segment, the one for which PS_SEG(SEG#) equals ASTE#. SOADD then calls DCONNECT to release this segment from the process's WS, and continues its loop through the processes.

After it has checked the last process, PROCESS#_MAX, the kernel segmentation register 2 must be restored with the current process's process segment. SOADD calls LSD to load the descriptors of PT_PS_ASTE#(THE_CURRENT_PROCESS) into the register at PS_KSR_ADR and then returns.

Function: SOADD
Parameters: SOADD(daste#, offset)
Effect:
Let aste# = HASH(DIR_DISK(daste#, offset));
IF aste# # 0;
THEN:
IF (PROCESS#_MIN <= process# <= PROCESS#_MAX) &
PT_FLAGS(process#) # INACTIVE) &
AST_CPL(aste#, process#);
THEN:
    IF AST_WAL(aste#, process#);
    THEN:  Let mode = WRITE;
    ELSE:  Let mode = READ;
    END;
    IF not DSEARCH(process#, daste#
    'DIR_ACL_HEAD'(aste#, offset), mode);
    THEN:
        IF (SEG#_MIX <= seg# <= SEG#_MAX) &
            ('PS_SEG'(process#, seg#) = aste#);
            THEN:  DCONNECT(process#, aste#, seg#);
        END;
    END;
    END;
END;

3.2.27.2 N/A

3.2.27.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIVE</td>
<td>DCONNECT</td>
</tr>
<tr>
<td>RESCIND</td>
<td>DSEARCH</td>
</tr>
<tr>
<td>DELETSEG</td>
<td>HASH</td>
</tr>
<tr>
<td></td>
<td>LSD</td>
</tr>
</tbody>
</table>

3.2.27.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function SOADD. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS_PROCESS_MASK</td>
<td>DASTE#</td>
<td>ASTE#</td>
</tr>
<tr>
<td>PS_SEG</td>
<td>OFFSET</td>
<td>PROCESS#</td>
</tr>
</tbody>
</table>

119
3.2.27.5 **Limitations**

None.

3.2.27.6 **Listing**

DATA SOADD(DASTE#, OFFSET);

PROGRAM SOADD;
DECLARE
   WORD (DASTE#, PROCESS#, MODE, REG#, SEG#, DUMMY);
   /* DETERMINE IF SEGMENT TO WHICH ACCESS HAS BEEN RESCINDED IS ACTIVE */
   ASTE# := HASH(DIR_DISK(OFFSET));
   IF (ASTE# <> 0) & (ASTE# <> 0);
   THEN:
      /* LOOP THROUGH ALL PROCESS'S */
      DO PROCESS# := PROCESS#_MIN TO PROCESS#_MAX;
         IF (PT_FLAGS(PROCESS#) & PT_FLAGS_MASK) <> INACTIVE;
            THEN: LSD(PT_PS_ASTE#(PROCESS#), PS_KSR_ADR, SDR_WRITE_ACCESS);
               /* IS THIS PROCESS CONNECTED TO THE SEGMENT? */
               IF (ASTE# <> 0) & PS_PROCESS_MASK) <> 0;
                  THEN: /* YES - DETERMINE MODE OF ACCESS */
                  IF (ASTE# <> 0) & PS_PROCESS_MASK) <> 0;
                     THEN: MODE := READ$EXECUTE_ACCESS;
                        ELSE: MODE := WRITE$READ$EXECUTE_ACCESS;
END;
3.2.28 Directory Search (DSEARCH)

The Directory Search CPC, DSEARCH, is a kernel level internal SKCPP function that is called by both a kernel level internal function and user level external functions. DSEARCH calls only kernel level internal functions. It is written in Project SUE System Language.

3.2.28.1 Description

DSEARCH determines whether the mode of access is permitted for the calling process. It makes sure the directory aste# supplied does identify a directory, returning the ERR_FLAG if AST_TYPE(ASTE#) logical and AST_TYPE_MASK (type, class, status, change, and unlock entries share a byte) does not equal AST_TYPE_DIRECTORY. It then gains access to the directory. If AST_ASR(ASTE#), which holds the main memory address of the segment, equals zero, DSEARCH must call SWAPIN to swap the directory into main memory. It calls LSD to load the directory's descriptors in kernel segmentation register 3 at DIR_KSR_ADR.

It then searches for an ACL element concerning the current process, beginning at the head of the ACL, with INDEX set to DIR_ACL_HEAD(OFFSET). It commences a cycle to find the appropriate ACL element: if INDEX equals zero, the end of the ACL has been reached without finding an element which gives the process access; DSEARCH returns with ERR_FLAG. It sets USER to ACL_USER(INDEX) logical and ACL_USER_MASK and it sets PROJECT to ACL_PROJECT(INDEX).
When USER equals ALL_USERS or USER equals PS_USER_ID and PROJECT equals ALL_PROJECTS or PROJECT equals PS_PROJECT_ID, the cycle is exited; otherwise, INDEX is reset to ACL_CHAIN(INDEX), the next ACL element number and the cycle continued.

It now uses the ACL element it has found to test whether the requested MODE is permitted. If ACL_MODE(INDEX) logical and ACL_MODE_MASK (mode and user share a word) equals NO_ACCESS, DSEARCH returns with ERR_FLAG. Also, if requested MODE is WRITE$READ$EXECUTE_ACCESS and ACL_MODE(INDEX) logical and ACL_MODE_MASK is not WRITE$READ$EXECUTE_ACCESS, ERR_FLAG is returned. Otherwise, DSEARCH returns with RC set to OK_FLAG.

Function: DSEARCH
Parameters: DSEARCH(process#, aste#, acle#, mode)
Value:
IF acle# ≠ 0;
THEN:
IF ((ACL_USER(aste#, acle#) = ALL_USERS) |
  (ACL_USER(aste#, acle#) = PS_USER_ID(process#)) &
  ((ACL_PROJECT(aste#, acle#) = ALL_PROJECTS) |
  (ACL_PROJECT(aste#, acle#) = PS_PROJECT_ID(process#)))
THEN:
IF ACL_MODE(aste#, acle#) = NO;
THEN: FALSE;
ELSE:
IF (mode = WRITE) &
  (ACL_MODE(aste#, acle#) ≠ WRITE);
THEN: FALSE;
ELSE: TRUE;
END;
END;
ELSE: DSEARCH(process#, aste#, ACL_CHAIN(aste#, acle#), mode);
END;
ELSE: FALSE
END;

3.2.28.2 N/A

3.2.28.3 Interfaces
Refer to Figure 6, Function Call Matrix, inparagraph 3.4.

Called By Calls
GETW SWAPIN

122
3.2.28.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function DSEARCH. For Data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table 1, List of Constants, in subparagraph 3.3.2.

### Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST_TYPE</td>
<td>DASTE#</td>
<td>RC</td>
</tr>
<tr>
<td>AST_ADR</td>
<td>OFFSET</td>
<td></td>
</tr>
<tr>
<td>DIR_ACL_HEAD</td>
<td>ASTE#</td>
<td></td>
</tr>
<tr>
<td>ACL_USER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACL_PROJECT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACL_CHAIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACL_MODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS_USER_ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS_PROJECT_ID</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Constants

<table>
<thead>
<tr>
<th>ACL_MODE_MASK</th>
<th>DIR_KSR_ADR</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL_USER_MASK</td>
<td>EFF_FLAG</td>
</tr>
<tr>
<td>ALL_PROJECTS</td>
<td>NO_ACCESS</td>
</tr>
<tr>
<td>ALL_USERS</td>
<td>OK_FLAG</td>
</tr>
<tr>
<td>AST_TYPE_DIRECTORY</td>
<td>SDR_WRITE_ACCESS</td>
</tr>
<tr>
<td>AST_TYPE_MASK</td>
<td>WRITE$READ$EXECUTE_ACCESS</td>
</tr>
</tbody>
</table>

3.2.28.5 Limitations

DSEARCH returns ERR_FLAG if the aste# supplied does not identify a directory, if the end of the ACL has been reached without finding an element that gives the process access, or if the access mode is not permitted. Otherwise, RC = OK_FLAG.

3.2.28.6 Listing

```plaintext
DATA DSEARCH(ASTE#, OFFSET, MODE) RETURNS (RC):
```

123
PROGRAM DSBARCH;
DECLARE
   WORD (INDEX, USER, PROJECT);
   IF (AST_TYPE(ASTE#) & AST_TYPE_MASK) = AST_TYPE_DIRECTORY;
      RETURN WITH ERR_FLAG;
   END;
   /* GAIN ACCESS TO DIRECTORY */
   IF AST_ADR(ASTE#) = 0;
      SWAPIN(ASTE#);
   END;
   LSD(ASTE#, DIR_KSR_ADR, SER_WRITE_ACCESS);
   /* NO NEED TO CHECK A USE BIT - ACL WILL BE EMPTY IF ENTRY IS NOT IN USE */
   /* SEARCH ACL FOR ELEMENT THAT GIVES CURRENT USER PERMISSION TO ACCESS */
   INDEX := DIP_ACL_HEAD(OFFSET);
   CYCLE
      IF INDEX = 0;
         RETURN WITH ERR_FLAG;
      END;
      USER := (ACL_USER(INDEX) & ACL_USER_MASK);
      PROJECT := ACL_PROJECT(INDEX);
      EXIT WHEN ((USER = ALL_USERS) | (USER = PS_USER_ID)) & ((PROJECT = PS_PROJECT_ID) | (PROJECT = ALL_PROJECTS));
      INDEX := ACL_CHAIN(INDEX);
   END;
   IF (ACL_MODE(INDEX) & ACL_MODE_MASK) = NO_ACCESS;
      RETURN WITH ERR_FLAG;
   END;
   IF (MODE = WRITE$READ$EXECUTE_ACCESS) & ((ACL_MODE(INDEX) & ACL_MODE_MASK) =
      WRITE$READ$EXECUTE_ACCESS);
      RETURN WITH ERR_FLAG;
   END;
   RC := OK_FLAG;

3.2.29 Activate Segment (ACT)

The Activate Segment CPC, ACT is a kernel level internal SKCPS function that is called by a user level external functions. ACT calls only kernel level internal functions. It is written in Project SUE System Language.
ACT activates a segment, copying its directory entry into the ASTE and initializing the other fields of the ASTE. First, it allocates an AST entry. It checks if any are on the free chain: if \( \text{AST\_CHAIN}() \) is zero, there are none. In this event it deactivates the ASTE which has been eligible for deactivation longest. It sets \( I \) to \( \emptyset \). It then repeatedly assigns \( \text{NEXT} \) the value of \( \text{AST\_AGE\_CHAIN}(I) \) and \( I \) the value of \( \text{NEXT} \) until \( \text{AST\_AGE\_CHAIN}(\text{NEXT}) = \emptyset \), marking the end of the age chain. \( \text{NEXT} \) now holds the aste\# of the entry which was on the age chain longest. ACT calls \text{DEACT} to free this entry. Now that there is at least one aste\# on the free chain, ASTE\# can be set equal to \( \text{AST\_CHAIN}(\emptyset) \), the aste\# of the first element on the free chain. Assigning \( \text{AST\_CHAIN}(\emptyset) \) the value of \( \text{AST\_CHAIN}(\text{ASTE}\#) \) removes the entry from the free chain. Since the segment is not connected to any process at activate time, it is marked eligible for deactivation by letting \( \text{AST\_AGE\_CHAIN}(\text{ASTE}\#) \) equal \( \text{AST\_AGE\_CHAIN}(\emptyset) \) and \( \text{AST\_AGE\_CHAIN}(\emptyset) \) equal \( \text{ASTE}\# \).

ACT then updates the HASH data base. It calls \text{PREHASH} to calculate the hash value \( \text{HASH\_VAL} \) associated with \( \text{DIR\_DISK}(\text{OFFSET}) \), the disk address of the segment. It then sets \( \text{AST\_CHAIN}(\text{ASTE}\#) \) to \( \text{HASH\_TABLE}(\text{HASH\_VAL}) \) and \( \text{HASH\_TABLE}(\text{HASH\_VAL}) \) to \( \text{ASTE}\# \). This adds the entry to the head of a chain of ASTE's whose entries disk addresses map onto the same hash value.

ACT is now ready to fill in the ASTE it has allocated. Since the segment is not yet connected to any processes and can't be swapped in yet, the main memory address, \( \text{AST\_ADR}(\text{ASTE}\#) \), the descriptor count, \( \text{AST\_DES\_COUNT}(\text{ASTE}\#) \), and the CPL, \( \text{AST\_CPL}(\text{ASTE}\#) \) are all set to zero. The rest of the entry is copied from the directory entry. \( \text{AST\_CLASS}(\text{ASTE}\#) \) is set equal to \( \text{DIR\_CLASS}(\text{OFFSET}) \) which sets the type and status as well as the classification. \( \text{AST\_CAT}, \text{AST\_DISK}, \text{and AST\_SIZE} \) get \( \text{DIR\_CAT}, \text{DIR\_DISK} \) and \( \text{DIR\_SIZE} \), respectively.

Now, if the directory status bit was set to uninitialized, it must be reset. If \( \text{DIR\_STATUS}(\text{OFFSET}) \) logical and \( \text{DIR\_STATUS\_MASK} \) equals \( \text{DIR\_UNINITIALIZED} \), \( \text{DIR\_STATUS}(\text{OFFSET}) \) is reset to \( \text{DIR\_STATUS}(\text{OFFSET}) \) logical and \( \text{DIR\_STATUS\_NOT\_MASK} \) (all 1's except for the status bit, so class and type information is not disturbed). In this case, the directory has been changed, so \( \text{AST\_CHANGE}(\text{DASTE}\#) \) must be reset to \( \text{AST\_CHANGE}(\text{DASTE}\#) \) logical or \( \text{AST\_CHANGED} \) to insure that the directory will be copied onto disk when swapped out.
Finally, ACT initializes the segment’s semaphore. SMFR_COUNT(ASTE#) is set to 1 and SMF_POINTER(ASTE#), the head of a chain of processes blocked on the semaphore, is set to Ø. ACT returns the ASTE# allocated to the segment.

Function: ACTIVATE
Parameters: ACTIVATE(daste#, offset)
Effect:
IF 'AST_CHAIN'(0) = 0;
THEN: Let aste# = NEXTASTE('AST_AGE_CHAIN'(0));
       DEACTIVATE(aste#);
ELSE: Let aste# = 'AST_CHAIN'(aste#);
       AST_CHAIN(0) = 'AST_CHAIN'(aste#);
END:
HASH(DIR_DISK(daste#, offset)) = aste#;
AST_ADR(aste#) = 0;
AST_LOCK(aste#) = UNLOCK;
AST_DES_COUNT(aste#) = 0;
IF (PROCESS#_MIX <= process# <= PROCESS#_MAX);
THEN: AST_CPL(aste#, process#) = FALSE;
       AST_WAL(aste#, process#) = FALSE;
END;
AST_TYPE(aste#) = DIR_TYPE(daste#, offset);
AST_STATUS(aste#) = 'DIR_STATUS'(daste#, offset);
AST_CLASS(aste#) = DIR_CLASS(daste#, offset);
AST_CAT(aste#) = DIR_CAT(daste#, offset);
AST_DISK(aste#) = DIR_DISK(daste#, offset);
AST_SIZE(aste#) = DIR_SIZE(daste#, offset);
IF 'DIR_STATUS'(daste#, offset) = UNINITIALIZED;
THEN: DIR_STATUS(daste#, offset) = INITIALIZED;
END;
AGE(aste#);

3.2.29.2 N/A

3.2.29.3 Interfaces
Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>DEACT</td>
</tr>
<tr>
<td>CONNECT</td>
<td>PREHASH</td>
</tr>
</tbody>
</table>

3.2.29.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function ACT. For data base references
refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST_CHAIN</td>
<td>DASTE#</td>
<td>ASTE#</td>
</tr>
<tr>
<td>AST_AGE_CHAIN</td>
<td>OFFSET</td>
<td>HASH_VAL</td>
</tr>
<tr>
<td>AST_ADR</td>
<td></td>
<td>INDEX</td>
</tr>
<tr>
<td>AST_DES_COUNT</td>
<td></td>
<td>NEXT</td>
</tr>
<tr>
<td>AST_CPL</td>
<td></td>
<td>I</td>
</tr>
<tr>
<td>AST_CHANGE</td>
<td></td>
<td>RC</td>
</tr>
<tr>
<td>AST_CLASS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_CAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_DISK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_SIZE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR_CLASS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR_CAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR_DISK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR_SIZE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR_STATUS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HASH_TABLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMFR_POINTER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMFR_COUNT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constants

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AST_CHANGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR_STATUS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR_STATUS_MASK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR_STATUS_NOTMASK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR_UNINITIALIZED</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.2.29.5 Limitations

3.2.29.6 Listing

DATA ACT(DASTE#, OFFSET) RETURNS (RC);
PROGRAM ACT;
DECLARE
    WORD (ASTE#, I, NEXT, HASH_VAL);
    /* ALLOCATE AN ASTE ENTRY */
    /* HEAD OF FREE ASTE CHAIN IS IN ASTE 0 */
    IF AST_CHAIN(0) = 0;
        THEN:
        /* NO FREE ASTE'S - LOOK ON AGE CHAIN */
        I := 0;
        CYCLE
            NEXT := AST_AGE_CHAIN(I);
            .... EXIT WHEN AST_AGE_CHAIN(NEXT) = 0;
            I := NEXT;
        END;
        DEACT(NEXT);
    END;
    /* A FREE ASTE EXISTS */
    ASTE# := AST_CHAIN(0);
    /* REMOVE THIS ASTE FROM THE FREE CHAIN AND AGE */
    AST_CHAIN(0) := AST_CHAIN(ASTE#);
    AST_AGE_CHAIN(ASTE#) := AST_AGE_CHAIN(0);
    AST_AGE_CHAIN(0) := ASTE#;
    /* UPDATE HASH DATA BASE */
    HASH_VAL := PHEHASH(DIR_DISK(OFFSET));
    AST_CHAIN(ASTE#) := HASH_TABLE(HASH_VAL);
    HASH_TABLE(HASH_VAL) := ASTE#;
    /* CLEAN UP AST ENTRY */
    AST_ADDR(ASTE#) := 0;
    AST_DES_COUNT(ASTE#) := 0;
    AST_CPL(ASTE#) := 0;
    /* FILL IN AST ENTRY */
    AST_CLASS(ASTE#) := DIR_CLASS(OFFSET); /* SETS TYPE, STATUS */
    AST_CAT(ASTE#) := DIR_CAT(OFFSET);
    AST_DISK(ASTE#) := DIR_DISK(OFFSET);
    AST_SIZE(ASTE#) := DIR_SIZE(OFFSET);
    IF (DIR_STATUS(OFFSET) & DIR_STATUS_MASK) = DIR_UNINITIALIZED;
        THEN: DIR_STATUS(OFFSET) := (DIR_STATUS(OFFSET) & DIR_STATUS_NOTMASK);
    /* DIRECTORY HAS BEEN WRITTEN INTO - MUST SET CHANGE BIT */
    AST_CHANGE(ASTE#) := (AST_CHANGE(ASTE#) | AST_CHANGED);
    END;
    /* INITIALIZE SEMAPHORES ASSOCIATED WITH SEGMENT */
    SMFR_COUNT(ASTE#) := 1;
    SMFR_POINTER(ASTE#) := 0;
    RC := ASTE#;
128
3.2.30 Deactivate Segment (DEACT)

The Deactivate Segment CPC, DEACT, is a kernel level internal SKCPP function that is only called by other kernel level internal functions. DEACT calls only kernel level internal functions. It is written in Project SUE System Language.

3.2.30.1 Description

DEACT removes a specified segment from the age chain and frees its active segment table entry. First, it runs through the age chain to find entry ASTE#. Starting with INDEX equal to zero, it repeatedly resets NEXT to AST_AGE_CHAIN(INDEX) and INDEX to NEXT until NEXT equals ASTE#. It links INDEX, the aste just before the specified ASTE to the entry after the specified ASTE by setting AST_AGE_CHAIN(INDEX) to AST_AGE_CHAIN(ASTE#). Setting AST_AGE_CHAIN(ASTE#) to zero marks it unaged.

Next, since an uninitialized segment cannot be deactivated, if AST_STATUS(ASTE#) logical and AST_STATUS_MASK equals AST_UNINITIALIZED, DEACT calls SWAPIN. SWAPIN places the segment in main memory and also causes it to be initialized. Then, if AST_ASR(ASTE#), which holds the segment's main memory address, is non-zero, SWAPOUT is called to remove the segment from main memory.

DEACT then removes the ASTE from the hash data base. It calls PREHASH which finds the HASH_VALUE associated with the segment's disk address AST_DISK(ASTE#). If HASH_TABLE(HASH_VAL), the head of the chain of astes with hash values of HASH_VAL, equals ASTE#, ASTE can be removed from the hash data base simply by resetting HASH_TABLE(HASH_VAL) to AST_CHAIN(ASTE#). Otherwise, DEACT must run through the chain until it finds ASTE. It starts by setting HASH_VAL to AST_CHAIN(ASTE#) and repeatedly setting NEXT to AST_CHAIN(HASH_VAL) and HASH_VAL to NEXT until NEXT equals ASTE#.

Now, ASTE can be removed from the HASH chain by letting AST_CHAIN(HASH_VAL) equal AST_CHAIN(ASTE#).

Finally, DEACT places ASTE# at the head of the free chain. It links it to the first aste on the chain by setting AST_CHAIN(ASTE#) to AST_CHAIN(∅) and resetting the head of the chain, AST_CHAIN(∅) to ASTE#. DEACT is then exited.
Function: DEACTIVATE
Parameters: DEACTIVATE(aste#)
Effect:
UNAGE('AST AGE CHAIN'(0), aste#);
IF 'AST STATUS'(aste#) = UNINITIALIZED;
THEN: SWAPIN(aste#);
   SWAPOUT(aste#);
ELSE:
   IF 'AST ADR'(ASTE#) ≠ 0;
      THEN: SWAPOUT(aste#);
   END;
END;
HASH(AST_DISK(aste#)) = 0;
AST_CHAIN(aste#) = 'AST_CHAIN'(0);
AST_CHAIN(0) = aste#;

Function: AGE
Parameters: AGE(aste#)
Effect:
AST_AGE_CHAIN(aste#, = 'AST_AGE_CHAIN'(0);
AST_AGE_CHAIN(0) = aste#;
AST_AGE(aste#) = AGED;

Function: UNAGE
Parameters: UNAGE(vaste#) = aste#
Effect:
IF 'AST AGE CHAIN'(vaste#) = aste#;
THEN: AST_AGE_CHAIN(vaste#) = AST_AGE_CHAIN(aste#);
      AST_AGE(aste#) = UNAGED;
ELSE: UNAGE('AST AGE CHAIN'(vaste#, aste#));
END;

Function: NEXTASTE
Parameters: NEXTASTE(aste#)
Effect:
IF AST_AGE_CHAIN(aste#) = 0;
THEN: aste#;
ELSE: NEXTASTE(AST_AGE_CHAIN(aste#));
END;

3.2.30.2 N/A
3.2.30.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETSEG</td>
<td>SWAPIN</td>
</tr>
<tr>
<td>ACT</td>
<td>SWAPOUT</td>
</tr>
<tr>
<td></td>
<td>PREHASH</td>
</tr>
</tbody>
</table>

3.2.30.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function DEACT. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

<table>
<thead>
<tr>
<th>Data Base References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global References</td>
<td>ASTE#</td>
<td>HASH.VAL INDEX</td>
</tr>
<tr>
<td>AST_CHAIN</td>
<td></td>
<td>NEXT</td>
</tr>
<tr>
<td>AST_AGE_CHAIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_STATUS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_ADR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HASH_TABLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local References</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HASH_VAL INDEX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEXT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.2.30.5 Limitations

None

3.2.30.6 Listing

```
DATA DEACT(ASTE#);

PROGRAM DEACT:
DECLARE
    WORD (HASH.VAL, INDEX, NEXT);
/* REMOVE FROM AGE CHAIN */
    INDEX := 0;
```
3.2.31 Swap Segment In (SWAPIN)

The Swap Segment in CPC, SWAPIN, is a kernel level internal SKCPP function that is called by both user level external and kernel level internal functions. SWAPIN calls only kernel level internal functions. It is written in Project SUE System Language.

3.2.31.1 Description

SWAPIN swaps a specified active segment into main memory. First, it allocates a free block of space for the segment. It assigns BLOCK# the value of MBT_CHAIN(0) logical and MBT_CHAIN_MASK (the flags, chain, and aste entries all share a word on the MBT). If BLOCK# equals END_BLOCK#, there are no more blocks on the free chain, so a segment must be swapped out. SWAPIN finds the segment which has been eligible for swapping out longest by running through to
the end of the swap chain. It sets I equal to zero. Then, it repeatedly resets NEXT to the next element in the chain, AST_SWAP_CHAIN(I) and I to NEXT until AST_SWAP_CHAIN(NEXT) is zero. Having found the end of the swap chain, it corrects BLOCK# to ASTADR(NEXT), the main memory address of the segment it will get rid of, and calls SWAPOUT to remove NEXT from main memory. There is now at least one block on the free chain. Assigning MBT_CHAIN(∅) the value of MBT_CHAIN(ASTE#) removes BLOCK# from the head of the chain and setting MBT_FLAGS(BLOCK#) to ALLOCATED marks the block allocated.

If the segment is uninitialized, SWAPIN initializes it rather than performing disk I/O. If AST_STATUS(ASTE#) logical and AST_STATUS_MASK (status shares a byte with four other AST entries) equal AST_UNINITIALIZED, INITSEG is called to reset the segment to all zeros and remove all ACL elements if the segment is a directory. Otherwise, DISKIO is called to read the segment from the disk. A P is performed in the DISK_SMFR which stalls the process until the disk operation is complete and a V is performed on the disk semaphore.

SWAPIN then updates the data base. ASTADR(ASTE#) is assigned BLOCK#, the main memory address of the segment, and MBT_ASTE(BLOCK#) gets ASTE# logical or ALLOCATED. Then, since no descriptors for the segment exist yet, SWAPIN marks the segment eligible for swapping out. It adds it to the head of the swap chain by letting AST_SWAP_CHAIN(ASTE#) equal AST_SWAP_CHAIN(∅) and by letting AST_SWAP_CHAIN(∅) equal ASTE#. Resetting AST_UNLOCK(ASTE#) to AST_UNLOCK(ASTE#) logical or AST_UNLOCK_FLAG shows that no descriptors exist for the segment and completes the operation.

Function: SWAPIN
Parameters: SWAPIN(aste#)
Effect:
Let size = AST_SIZE(aste#);
Let block# = FINDFREE('MBT_CHAIN'(0), size);
ALLOCBLOCK('MBT_CHAIN'(0), block#);
IF 'AST_STATUS'(aste#) = UNINITIALIZED;
THEN: INITSEG(aste#, block#);
AST_STATUS(aste#) = INITIALIZED;
AST_CHANGE(block#) = CHANGED;
ELSE: DISKIO(aste#, block#, DISK_READ);
AST_CHANGE(block#) = UNCHANGED;
P(DISK_SEMAPHORE);
END;
ASTADR(aste#) = block#;
MBT_ASTE(block#) = aste#;
UNLOCK(aste#);

Function: FINDFREE
Parameters: FINDFREE(block#, size)
value:
IF block# = END_BLOCK#
THEN: RESTART;
ELSE:
   IF MBT_SIZE(block#) = size
       THEN: block#
       ELSE: FINDFREE(MBT_CHAIN(block#, size)
       END;
END;

Function: ALLOCMEM
Parameters: ALLOCMEM(vblock#, block#)
Effect:
IF 'MBT_CHAIN'(vblock#) = block#
THEN: MBT_CHAIN(vblock#) = MBT_CHAIN(block#);
     MBT_FLAGE(block#) = ALLOCATED;
ELSE: ALLOCMEM('MBT_CHAIN'(vblock#), block#);
END;

3.2.31.2 N/A

3.2.31.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEACT</td>
<td>INITSEG</td>
</tr>
<tr>
<td>GETDIR</td>
<td>SWAPOUT</td>
</tr>
<tr>
<td>READIR</td>
<td>P</td>
</tr>
<tr>
<td>WRITEDIR</td>
<td>DISKIO</td>
</tr>
<tr>
<td>DSEARCH</td>
<td></td>
</tr>
<tr>
<td>ENABLE</td>
<td></td>
</tr>
</tbody>
</table>

3.2.31.4 Data Base Organization

Listed below are Security Kernel data base references and constants used by the function SWAPIN. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.
3.2.31.5 Limitations

None

3.2.31.6 Listing

DATA SWAPIN(ASTE#);
DECLARE
  PROCEDURE ACCEPTS (WORD, WORD) (INITSEG);

PROGRAM SWAPIN;
DECLARE
  WORD (BLOCK#, I, NEXT);
  /* LOOK FOR A FREE BLOCK */
  BLOCK# := (MBT_CHAIN(0) & MBT_CHAIN_MASK);
  IF BLOCK# = END_BLOCK#;
    /* NO FREE BLOCKS - LOOK AT SWAP CHAIN FOR SOMETHING TO SWAPOUT */
  THEN: I := 0;
  CYCLE
    NEXT := AST_SWAP_CHAIN(I);
    .... EXIT WHEN AST_SWAP_CHAIN(NEXT) = 0;
    I := NEXT;
END;

3.2.32 Swap Segment Out (SWAPOUT)

The Swap Segment Out CPC, SWAPOUT, is a kernel level internal SKCPP function that is only called by other kernel level internal functions. SWAPOUT calls only kernel level internal functions. It is written in Project SUE System Language.

3.2.32.1 Description

SWAPOUT removes a specified segment from main memory. It sets BLOCK# to AST_ADR(ASTE#), which holds the main memory address of the segment, and AST_ADR_(ASTE#) to zero to show that the segment is swapped out.

Then, SWAPOUT marks the segment ineligible for swapping out. It goes through the swap chain to find the segment identified by ASTE#. Starting with I equal to Ø and resetting NEXT to AST_SWAP_CHAIN(INDEX) and INDEX to NEXT until NEXT equals ASTE#. INDEX now holds the aste# of the segment before the one designated to be swapped out. SWAPOUT removes ASTE# from the swap chain by setting AST_SWAP_CHAIN(INDEX) to AST_SWAP_CHAIN(ASTE#) and AST_SWAP_CHAIN(ASTE#) to zero. It then resets the unlocked bit to locked by letting AST_UNLOCK(ASTE#) equal AST_UNLOCK(ASTE#) logical and AST_LOCK_MASK (all 1's except for a Ø in the UNLOCK bit).
If the segment has been changed while in main memory, it should be copied onto disk; otherwise SWAPOUT doesn't bother. If AST_CHANGE(ASTE#) logical and AST_CHANGE_MASK equals AST_CHANGED, DISKIO is called to initiate a disk write operation. While the operation is performed, SWAPOUT resets the change bit to AST_CHANGE (ASTE#) logical and AST_UNCHANGED_MASK. To avoid getting ahead of the disk, it performs a P on the DISK_SMFR, stalling the process until the disk write is completed.

SWAPOUT then frees the memory allocated to the segment, putting the block on the free chain in ascending order of block#'s. Starting with INDEX equal to 0, it runs through the chain until the next block#, MBT_CHAIN(INDEX) logical and MBT_CHAIN_MASK, is greater than BLOCK#, repeatedly resetting INDEX to MBT_CHAIN(INDEX) and MBT_CHAIN_MASK. INDEX now contains the block after which BLOCK# should be inserted. Setting MBT_CHAIN(BLOCK#) equal to MBT_CHAIN(INDEX) and MBT_CHAIN (INDEX) to BLOCK# logical or FREE_MEM places BLOCK# in the proper position on the free chain. SWAPOUT is then exited.

Function: SWAPOUT
Parameters: SWAPOUT(aste#)
Effect:
Let block# = 'AST_ADR' (aste#);
LOCK(aste#);
AST_ADR(aste#) = 0;
IF AST_CHANGE(block#) = CHANGED;
THEN: DISKIO(aste#, block#, DISK_WRITE);
      P(DISK_SEMAPHORE);
END;
FREE_MEM('MBT_CHAIN'(0), block#);

Function: FREE_MEM
Parameters: FREE_MEM(vblock#, block#)
Effect:
IF 'MBT_CHAIN(block#) > block#;
THEN: MBT_CHAIN(block#) = 'MBT_CHAIN'(vblock#);
      MBT_CHAIN(vblock#) = block#;
      MBT_FLAGS(block#) = FREE;
      MBT_ASTE(block#) = 0;
ELSE: FREE_MEM('MBT_CHAIN'(vblock#), block#);  
END;

3.2.32.2 N/A

3.2.32.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.
3.2.32.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function SWAPOUT. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST_ADR</td>
<td>ASTF#</td>
<td>BLOCK#</td>
</tr>
<tr>
<td>AST_SWAP_CHAIN</td>
<td></td>
<td>INDEX</td>
</tr>
<tr>
<td>AST_UNLOCK</td>
<td></td>
<td>NEXT</td>
</tr>
<tr>
<td>AST_CHANGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MBT_CHAIN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constants

| ALLOCATED         | AST_CHANGE         | AST_CHANGE_MASK |
| AST_UNLOCK        | AST_LOCK_MASK      | AST_UNCHANGED_MASK |
| DISK_SMFR         | DISK_WRITE         | FREE_MEM         |
| MBT_CHANGE_MASK   |                     |                 |

3.2.32.5 Limitations

None

3.2.32.6 Listing

DATA SWAPOUT(ASTF#):

138
3.2.33 Initialize Segment (INITSEG)

The Initialize Segment CPC, INITSEG, is a kernel level internal SKCPC function that is called by another kernel level internal function. INITSEG calls only one other kernel level internal function. It is written in Project SUE System Language.

3.2.33.1 Description

INITSEG initializes a directory or data segment. It sets
AST_ADR(ASTE#), the segment's main memory address, to the BLOCK# supplied as a parameter. It then calls LSD to load the descriptors of the segment identified by ASTE# in the kernel register at DIR_KSR_ADR. This provides access to the segment.

INITSEG then performs the actual initialization. As I goes from $\emptyset$ to 511, it sets ZERO_ARRAY(I) to $\emptyset$. Thus, assuming all segments are SIZE2, 1k bytes, if the segment is a directory, all entries are marked free, and if it is a data segment, it is set to all zeros. Next, INITSEG tests if AST_TYPE(ASTE#) logical and AST_TYPE_MASK (type shares a byte with four other attributes) equal AST_TYPE_DIRECTORY. If so, the segment is a directory and it must place all ACL elements on the free chain. Letting ACL_CHAIN(I) equal I + 1 as I goes from $\emptyset$ to ACL_MAX links all elements from ACL_MIN = 1 to ACL_MAX to the free chain head, ACL_CHAIN(0). INITSEG then sets ACL_CHAIN(ACL_MAX) to $\emptyset$ to mark the end of the free chain.

To complete the operation, INITSEG updates the AST. AST_STATUS is set to AST_STATUS_NOTMASK. This resets the status bit to $\emptyset$, meaning initialized. The change bit is set by assigning to ACT_CHANGE the value of AST_CHANGE logical or AST_CHANGED, so that the segment will be copied to the disk when it is swapped out. INITSEG then returns.

Function: INITSEG
Parameters: INITSEG(aste#, block#)
Effect:
IF AST_TYPE(aste#) = DIRECTORY;
THEN:
   IF (OFFSET_MIN <= i <= OFFSET_MAX;
      THEN: DIR_SIZE(i) = 0;
      END;
   IF (ACL#_MIN <= j <= ACLE#_MAX;
      THEN: ACL_CHAIN(j) = (j+1) MODULO ACLE#_MAX;
      END;
   ELSE: segment_contents = 0;
   END;
ELSE: segment_contents = 0;
END;

3.2.33.2 N/A

3.2.33.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.
3.2.33.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function INITSEG. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

<table>
<thead>
<tr>
<th>Data Base References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST_ADR</td>
<td>ASTE#</td>
<td>I</td>
</tr>
<tr>
<td>AST_TYPE</td>
<td>BLOCK#</td>
<td></td>
</tr>
<tr>
<td>AST_STATUS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_CHANGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACL_CHAIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constants</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACL_MAX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_CHANGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_STATUS NOTMASK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_TYPE_DIRECTORY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_TYPE_MASK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR_KSR_ADR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDR_WRITE_ACCESS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.2.33.5 Limitations

None

3.2.33.6 Listing

DATA INITSEG(AST#, BLOCK#);

PROGRAM INITSEG;
DECLARE
  WORD (I);
  AST_ADR(ASTE#) := BLOCK#;
  /* GAIN ACCESS TO SEGMENT */

LSD(ASTE#, DIR_KSR_ADR, SDR_WRITE_ACCESS);

    /*    THIS ASSUMES ALL SEGS ARE 1K BYTGS - WILL CHANGE LATER    */
    DO I := 0 TO 511;
        ZERO_ARRAY(I) := 0;
    END;

    IF (AST_TYPE(ASTE#) & AST_TYPE_MASK) = AST_TYPE_DIRECTORY;
    THEN:
        /*    PUT ALL ACL ELEMENTS ON FREE CHAIN    */
        DO I := 0 TO ACL_MAX;
            ACL_CHAIN(I) := I + 1;
        END;
        ACL_CHAIN(ACL_MAX) := 0;
    END;

    END;
    /*    UPDATE AST    */
    AST_STATUS(ASTE#) := (AST_STATUS(ASTE#) & AST_STATUS_NOTMASK) ;
    AST_CHANGE(ASTE#) := (AST_CHANGE(ASTE#) | AST_CHANGED) ;

3.2.34 Prehash (PREHASH)

    The Prehash CPC, PREHASH, is a kernel level internal SKCPP function
that is directly called by other kernel level internal functions.
It is written in Project SUE System Language, including the use
of the Inline feature.

3.2.34.1 Description

    PREHASH converts the disk address of a segment input to it into
an index into HASH_TABLE. It hash codes the 16-bit disk address into
an 8-bit index equal to the exclusive or of the high order 8-bits
with the low order 8-bits of the address.

Function:  PREHASH
Possible values:  HASH_VAL or 0
Initial value:  0
Parameters:  PREHASH(disk_address)
3.2.34.2 Flow Chart

PREHASH

TEMP0 = DISK_ADR

TEMP1 = TEMP0

TEMP1 = TEMP1 / 2^8

TEMP0 = TEMP1 \lor TEMP0

HASH_VAL = TEMP0

HASH_VAL = HASH_VAL \& (2^8 - 1)

PREHASH = HASH_VAL

RETURN
3.2.34.3 Interfaces

 Called By Calls
 ACT None
 DEACT

3.2.34.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function PREHASH. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>DISK_ADR</td>
<td>HASH_VAL</td>
</tr>
</tbody>
</table>

Constants

ASHR1
MOV
SEG_FLAG
XOR10

3.2.34.5 Limitations

None

3.2.34.6 Listing

DATA PREHASH(DISK_ADR) RETURNS (HASH_VAL):

PROGRAM PREHASH;
    INLINE(MOV, DISK_ADR, 0, 0);
    INLINE(MOV, 0, 0, 0, 0, 1);  
    INLINE(ASHR1);
    INLINE("PPP1");
    INLINE(XOR10);
    INLINE(MOV, 0, 0, HASH_VAL);
    HASH_VAL := (HASH_VAL & "GOOD");
3.2.35 **Hash (HASH)**

The Hash CPC, HASH, is a kernel level internal SKCPP function that is directly called by both user level and other kernel level internal functions. It is written in Project SUE System Language, including the use of the Inline feature.

3.2.35.1 **Description**

HASH converts the disk address of a segment input to it into the aste# of the segment, if the segment is active. It hash codes the 16-bit disk address into an 8-bit index into HASH_TABLE equal to the exclusive or of the high order 8-bits with the low order 8-bits of the address. This index is then used to retrieve the associated aste#. If the disk address stored in the AST for this aste# matches the input disk address, this is the required aste#. If it does not match, any chains caused by hashing collisions must be run. This process continues until a match occurs or an aste# of zero is retrieved, indicating that the segment is not active.

Function:  HASH
Possible values:  aste# or 0
Initial value:  0
Parameters:  HASH(disk_address)
3.2.35.2 Flow Chart

```
3.2.35.2 Flow Chart

HASH

TEMP0 = DISK_ADR

TEMP1 = TEMP0

TEMP1 = TEMP1 / 2^8

TEMP0/ = TEMP1 \& TEMP0

HASH_VAL = TEMP0

TEMP1 = HASH_VAL \& (2^8 - 1)

ASTE = HASH_TABLE(TEMP1)

ASTE# = ASTE?

ASTE# = ASTE_CHAIN(ASTE#)

HASH = ASTE#?

RETURN

Compute Hash
Code from Disk
Address

Yes

No

Yes

No

146
```
3.2.35.3 **Interfaces**

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>None</td>
</tr>
<tr>
<td>STARTP</td>
<td></td>
</tr>
<tr>
<td>CHANGEO</td>
<td></td>
</tr>
<tr>
<td>DELETSEG</td>
<td></td>
</tr>
<tr>
<td>SOADD</td>
<td></td>
</tr>
<tr>
<td>CONNECT</td>
<td></td>
</tr>
</tbody>
</table>

3.2.35.4 **Data Organization**

Listed below are Security Kernel data base references and constants used by the function HASH. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

**Data Base References**

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>HASH_TABLE</td>
<td>DISK_ADR</td>
<td>HASH_VAL</td>
</tr>
<tr>
<td>AST_DISK</td>
<td></td>
<td>ASTE#</td>
</tr>
<tr>
<td>AST_CHAIN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Constants**

ASHR1
MOV
XOR10

3.2.35.5 **Limitations**

None

3.2.35.6 **Listing**

```plaintext
DATA HASH (DISK_ADDR) RETURNS (ASTE#);

PROGRAM HASH:
DECLARE
WORD (HASH_VAL);
/* HASH DISK_ADR */
```
3.2.36 Load Segment Descriptors (LSD)

The Load Segment Descriptors CPC, LSD, is a kernel internal SKCPP function that is called by both user level external functions and kernel level internal functions. It is written in PAL-11 assembly language.

3.2.36.1 Description

LSD constructs segmentation descriptors for a specified segment and loads them in a specified register. It converts the AST_SIZE into a form useable in the segment length field. Since the size is in blocks which omit the eight low-order bits of the length, and the descriptors omit only six low-order bits, the size is multiplied by 2. It is decremented because of the MMU's excess one's notation. The size word is then rotated so that the 8 low-order bits become the high-order bits. The register at REG_LOC is assigned the inclusive or of the size word and MODE; thus, its high order byte is the segment length field and its low order byte contains written-into and access control information. REG_LOC, the pointer to the descriptor register, is increased by \(408\) to point to the address register. ASTE# is multiplied by 2 before being used as a pointer to the AST_ADR array of two-byte words. The address is converted from memory blocks to a descriptor address with six low-order bits omitted and assigned to the segment address register. This completes the operation.

Function: LSD
Parameters: LSD(process#, block#, reg#, mode)
Effect:
PS_SAR (process#, reg#) = block#
PS_SDR (process#, reg#) = MIB_SIZE(block#), mode;
3.2.36.2 Flow Chart

```
LSD

TEMPØ = ASTE#

TEMP1 = AST_SIZE (TEMPØ)

TEMP1 = TEMP1 * 2^2

TEMP1 = TEMP1 - 1

TEMP1 = TEMP1 * 2^8

TEMP1 = TEMP1

V MODE

TEMP2 = REG LOC

SDR @ TEMP2 = TEMP1

TEMP2 = TEMP2 + 40^8

TEMPØ = TEMPØ * 2

TEMP1 = AST_ADR (TEMPØ)

TEMP1 = TEMP1 * 2^2

SAR @ TEMP2 + TEMP1

RETURN
```
3.2.36.3 **Interfaces**  

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE</td>
<td>None</td>
</tr>
<tr>
<td>GETDIR</td>
<td></td>
</tr>
<tr>
<td>READIR</td>
<td></td>
</tr>
<tr>
<td>WRITEDIR</td>
<td></td>
</tr>
<tr>
<td>SOADD</td>
<td></td>
</tr>
<tr>
<td>DSEARCH</td>
<td></td>
</tr>
<tr>
<td>INITSEG</td>
<td></td>
</tr>
<tr>
<td>STARTP</td>
<td></td>
</tr>
<tr>
<td>RUN</td>
<td></td>
</tr>
</tbody>
</table>

3.2.36.4 **Data Organization**  

Listed below are Security Kernel data base references and constants used by the function LSD. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

**Data Base References**

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST_SIZE</td>
<td>ASTE#</td>
<td>ASTSIZE</td>
</tr>
<tr>
<td>AST_ADR</td>
<td>REG_LOC</td>
<td>ASTDR</td>
</tr>
<tr>
<td>MODE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Constants**

- ADD
- ASL
- DEC
- JMP
- MOV
- MOVB
- SWAB

3.2.36.5 **Limitations**

None
3.2.36.6 Listing

<table>
<thead>
<tr>
<th>STMT</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R0=0</td>
</tr>
<tr>
<td>2</td>
<td>R1=1</td>
</tr>
<tr>
<td>3</td>
<td>R2=2</td>
</tr>
<tr>
<td>4</td>
<td>R3=3</td>
</tr>
<tr>
<td>5</td>
<td>R4=4</td>
</tr>
<tr>
<td>6</td>
<td>R5=5</td>
</tr>
<tr>
<td>7</td>
<td>R6=6</td>
</tr>
<tr>
<td>8</td>
<td>PC=7</td>
</tr>
<tr>
<td>9</td>
<td>:</td>
</tr>
<tr>
<td>10</td>
<td>LSD: LOAD_SEGMENT_DESCRIPTOR (ASTE#, REG_LOC, MODE);</td>
</tr>
<tr>
<td>11</td>
<td>:</td>
</tr>
<tr>
<td>12</td>
<td>ASTA=007400      ; BASE OF AST_ADDR ARRAY</td>
</tr>
<tr>
<td>13</td>
<td>ASTSIZ=005000    ; BASE OF AST_SIZE ARRAY</td>
</tr>
<tr>
<td>14</td>
<td>:</td>
</tr>
<tr>
<td>15</td>
<td>LSD: MOV R6,R5   ; SUE</td>
</tr>
<tr>
<td>16</td>
<td>ADD #10,R5       ; ENTRY</td>
</tr>
<tr>
<td>17</td>
<td>MOV PC,(R5)+     ; SEQUENCE</td>
</tr>
<tr>
<td>18</td>
<td>MOV -4(R5),R0    ; ASTE#</td>
</tr>
<tr>
<td>19</td>
<td>MOV R5,ASTSIZ(R0),R1 ; SIZE OF THE SEGMENT</td>
</tr>
<tr>
<td>20</td>
<td>ASL R1           ; MY BLOCKS TO THEIR'S</td>
</tr>
<tr>
<td>21</td>
<td>ASL R1</td>
</tr>
<tr>
<td>22</td>
<td>DSC R1</td>
</tr>
<tr>
<td>23</td>
<td>SWAB R1          ; MMU USES EXCESS ONES NOTATION</td>
</tr>
<tr>
<td>24</td>
<td>BIS -10(R5),R1   ; SLF IS IN LEFT BYTE OF DESCRIPTOR REG</td>
</tr>
<tr>
<td>25</td>
<td>MOV -6(R5),R2    ; POINT TO DESCRIPTOR REG</td>
</tr>
<tr>
<td>26</td>
<td>MOV R1,(R2)      ; STORE DESCRIPTOR</td>
</tr>
<tr>
<td>27</td>
<td>ADD #40,R2       ; POINT TO ADDRESS REGISTER</td>
</tr>
<tr>
<td>28</td>
<td>ASL R0           ; ARRAY OF WORDS</td>
</tr>
<tr>
<td>29</td>
<td>MOV R5,ASTA(R0),R1 ; BASE ADDRESS OF SEGMENT</td>
</tr>
<tr>
<td>30</td>
<td>ASL R1           ; MY BLOCKS TO THEIR'S</td>
</tr>
<tr>
<td>31</td>
<td>ASL R1</td>
</tr>
<tr>
<td>32</td>
<td>MOV R1,(R2)      ; STORE IN SEGMENTATION REGISTER</td>
</tr>
<tr>
<td>33</td>
<td>JMP R1,(R2)      ; DONE</td>
</tr>
<tr>
<td>34</td>
<td>END LSD</td>
</tr>
</tbody>
</table>

3.2.37 Disk I/O (DISKIO)

The Disk I/O CPC, DISKIO, is a kernel level internal SKCPC function that is called by other kernel level internal functions. It is written in PAL-11 assembly language.

3.2.37.1 Description

DISKIO initiates a disk input/output operation. If RFDSC, the disk control status, indicates the disk is not ready then there is a hardware failure, which is beyond the scope of the Security Kernel.

Otherwise, it converts the parameters into forms useable by the disk. Multiplying the SIZE in 256-byte blocks by 128 converts the units to words and negating the result yields its two's complement which the disk requires. This value is used as RFDWC, the disk word count. DISKIO then restores the eight low order 0's to the
DISK_ADDRESS by multiplying it by $2^8$. It lets RFDAR, the disk address register, and RFDAE, the disk address extension error register, equal this result. The current memory address, RFCMA, is found by restoring the eight low-order bits to MEMORY_ADDRESS. The disk control status word is then constructed by computing the inclusive or of any extended bits shifted four digits to the left with MODE. Having thus started the disk, DISKIO returns.

Function: DISKIO
Parameters: DISKIO(aste#, block#, command)
Effect:
DISK_ADR = AST_DISK(aste#);
DISK_COUNT = AST_SIZE(aste#)
MEM_ADR = block#;
DISK_COMMAND = command, ENABLE_INTERRUPTS;
3.2.37.2 Flow Chart

DISKIO

Is RFDS C \( \emptyset \)?

Yes

\( \text{TEMP} 1 = \text{SIZE} \)

\( \text{TEMP} 1 = \text{TEMP} 1 \times 2^7 \)

\( \text{TEMP} 1 = \text{TEMP} 1 \)

\( \text{RFWC} = \text{TEMP} 1 \)

\( \text{TEMP} \emptyset = \emptyset \)

\( \text{TEMP} 1 = \text{DISK_ADDRESS} \)

\( \text{TEMP} \emptyset, \text{TEMP} 1 = \text{TEMP} 1 \times 2^8 \)

\( \text{RFDAE} = \text{TEMP} \emptyset \)

\( \text{RFDAE} = \text{TEMP} 1 \)

\( \text{TEMP} \emptyset = \emptyset \)

\( \text{TEMP} 1 = \text{MEMORY_ADDRESS} \)

\( \text{TEMP} \emptyset, \text{TEMP} 1 = \text{TEMP} 1 \times 2^8 \)

\( \text{RFCMA} = \text{TEMP} 1 \)

\( \text{TEMP} \emptyset = \text{TEMP} \emptyset \times 2^4 \)

\( \text{TEMP} \emptyset = \text{TEMP} \emptyset \text{ V MODE} \)

\( \text{RFDS C} = \text{TEMP} \emptyset \)

RETURN

153
3.2.37.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWAPIN</td>
<td>None</td>
</tr>
<tr>
<td>SWAPOUT</td>
<td></td>
</tr>
</tbody>
</table>

3.2.37.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function DISKIO. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

<table>
<thead>
<tr>
<th>Data Base References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global References</td>
</tr>
<tr>
<td>None</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
</tr>
<tr>
<td>CLR</td>
</tr>
<tr>
<td>JMP</td>
</tr>
<tr>
<td>MOV</td>
</tr>
<tr>
<td>NEG</td>
</tr>
</tbody>
</table>

3.2.37.5 Limitations

The disk control status must be negative for DISKIO to operate.
3.2.37.6  Listing

<table>
<thead>
<tr>
<th>Stmt</th>
<th>Source Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R0=0</td>
</tr>
<tr>
<td>2</td>
<td>R1=1</td>
</tr>
<tr>
<td>3</td>
<td>R2=2</td>
</tr>
<tr>
<td>4</td>
<td>R3=3</td>
</tr>
<tr>
<td>5</td>
<td>R4=4</td>
</tr>
<tr>
<td>6</td>
<td>R5=5</td>
</tr>
<tr>
<td>7</td>
<td>R6=6</td>
</tr>
<tr>
<td>8</td>
<td>PC=7</td>
</tr>
<tr>
<td>9</td>
<td>:</td>
</tr>
<tr>
<td>10</td>
<td>DISKIO(DISK_ADDRESS, MEMORY_ADDRESS, SIZE, MODE);</td>
</tr>
<tr>
<td>11</td>
<td>:</td>
</tr>
<tr>
<td>12</td>
<td>RFDSC=177460</td>
</tr>
<tr>
<td>13</td>
<td>RFCMA=177464</td>
</tr>
<tr>
<td>14</td>
<td>RFDAR=177466</td>
</tr>
<tr>
<td>15</td>
<td>RFDAB=177470</td>
</tr>
<tr>
<td>16</td>
<td>:</td>
</tr>
<tr>
<td>17</td>
<td>DISKIO: MOV R6, R5</td>
</tr>
<tr>
<td>18</td>
<td>ADD #12, R5</td>
</tr>
<tr>
<td>19</td>
<td>MOV PC, (R5)+</td>
</tr>
<tr>
<td>20</td>
<td>TSTB @RFDSC</td>
</tr>
<tr>
<td>21</td>
<td>ERROR: BGE ERROR</td>
</tr>
<tr>
<td>22</td>
<td>MOV -10 (R5), R1</td>
</tr>
<tr>
<td>23</td>
<td>.WORD 072127 :ASH R1,#7</td>
</tr>
<tr>
<td>24</td>
<td>.WORD 000007</td>
</tr>
<tr>
<td>25</td>
<td>MOV R1,#RFMA</td>
</tr>
<tr>
<td>26</td>
<td>MOV R1,#RFDSC</td>
</tr>
<tr>
<td>27</td>
<td>MOV R1,#RFDA</td>
</tr>
<tr>
<td>28</td>
<td>CLR R0</td>
</tr>
<tr>
<td>29</td>
<td>MOV -4(R5), R1</td>
</tr>
<tr>
<td>30</td>
<td>.WORD 073027 :ASHC R0, #10</td>
</tr>
<tr>
<td>31</td>
<td>.WORD 000010</td>
</tr>
<tr>
<td>32</td>
<td>MOV R0,#RFDAB</td>
</tr>
<tr>
<td>33</td>
<td>MOV R1,#RFDAR</td>
</tr>
<tr>
<td>34</td>
<td>CLR R0</td>
</tr>
<tr>
<td>35</td>
<td>MOV -6(R5), R1</td>
</tr>
<tr>
<td>36</td>
<td>.WORD 073027 :ASHC R0, #10</td>
</tr>
<tr>
<td>37</td>
<td>.WORD 000010</td>
</tr>
<tr>
<td>38</td>
<td>MOV R1,#RFCA</td>
</tr>
<tr>
<td>39</td>
<td>MOV R0,#RFDSC</td>
</tr>
<tr>
<td>40</td>
<td>.WORD 072027 :ASH R0, #4</td>
</tr>
<tr>
<td>41</td>
<td>.WORD 000004</td>
</tr>
<tr>
<td>42</td>
<td>BIS -12(R5), R0</td>
</tr>
<tr>
<td>43</td>
<td>MOV R0,#RFDSC</td>
</tr>
<tr>
<td>44</td>
<td>JMP @-14(R5)</td>
</tr>
</tbody>
</table>

3.2.38  Disk Allocation (DALLOC)

The Disk Allocation CPC, DALLOC, is a kernel level internal SKCPP function that is only called by a user level external function. It is written in PAL-11 assembly language.

3.2.38.1  Description

DALLOC allocated space on the disk for a segment being created and sets NEXT_DISK_ADDRESS to the address of the space allocated. Its
The parameter is the address of the bit maps table for segments of the desired size; at this time, only size 2, 1K bytes, is implemented. The bit maps table has four entries: the starting and ending addresses of the bit map, the base disk address, and the shift factor, \( \log_2 \text{size} \).

DALLOC searches the bit map for a zero bit, which indicates a segment not in use. It sets all the bits of TEMP4 to 1. Then, starting at the starting address, it checks if the bit map word equals TEMP4. If not, it had found a word with a zero bit, and can proceed; otherwise, it continues its search. If it reaches the end of the bit map without finding a zero bit, the error is unrecoverable, so it halts.

After finding a word which contains a zero bit, it finds a word number. Since the search operates in auto-increment mode, DALLOC subtracts 2 from TEMP2 to obtain the address of the word with the free bit. TEMP3, the byte number of the word within the bit map, is set to TEMP2 minus the starting address of the bit map.

It then searches the word for its first zero bit. It repeatedly performs right shifts, incrementing TEMP4 each time, while the carry bit is set from the rightmost bit before the right shift, TEMP4 now holds the number of the first bit set to zero.

DALLOC now sets the bit for the free segment it found to "in use". It "or's" the word it found with a word containing a 1 in the bit corresponding to the first 0 bit.

DALLOC now translates the byte number and bit number it has found into a disk address. The number of the segment in the disk area has the word number as its high order bits and the bit number as its low order bits. Multiplying this by the size of the segments, that is, \( 2 \times \text{SHIFT}\_\text{FACTOR} \) yields the offset of the allocated disk area from the base of the disk area for segments of this size. Adding BMT(3), the BASE\_DISK\_ADR to the offset produces a disk address. DALLOC returns this value to the user.

Function: DISK\_ALLOC
Parameters: DISK\_ALLOC(size)
Effect:
\[ \forall k \quad \left( \text{BIT}\_\text{MAP}(\text{size}, k) = 0 \right) \land \\
\left( \text{BIT}\_\text{MAP}(\text{size}, k) = 1 \right) \land \\
\left( \text{NEXT}\_\text{DISK}\_\text{ADDRESS} = \text{BASE}(\text{size}) + k \times \text{size} \right) \]
3.2.38.2 Flow Chart

**DALLOC**

1. TEMP0 = BMT ADR
2. TEMP1 = TEMP0
3. TEMP2 = BIT_MAP_START ADR @ TEMP1
4. TEMP1 = TEMP1 + 2
5. TEMP3 = BIT_MAP_END ADR @ TEMP1
6. TEMP4 = 2^16 - 1
7. TEMP1 = BIT_MAP_WORD @ TEMP2
8. TEMP2 = TEMP2 + 2
9. **WFOUND**
   - TEMP2 = TEMP2 - 2
   - TEMP3 = TEMP2
   - TEMP3 = TEMP3 - BIT_MAP_START ADR @ TEMP0
   - TEMP4 = TEMP4 + 1
   - TEMP1 = TEMP1/2
   - **IS THE CARRY BIT SET?**
     - **Yes**
       - TEMP1 = 1
     - **No**
       - TEMP1 = TEMP1*2 TEMP4
       - BIT_MAP_WORD @ TEMP2 = BIT_MAP_WORD @ TEMP4 / TEMP1
       - TEMP3 = TEMP3*2^3
       - TEMP3 = TEMP3 + TEMP4
       - TEMP2 = SHIFT_FACTOR @ (TEMP0 + 6)
       - TEMP3 = TEMP3*2 TEMP2
       - TEMP3 = TEMP3 + BASE_DISK_ADR @ (TEMP0 + 4)
10. **NEXT_DISK_ADDRESS**
11. RETURN
12. **HALT**
3.2.38.3 **Interfaces**

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>None</td>
</tr>
</tbody>
</table>

3.2.38.4 **Data Organization**

Listed below are Security Kernel data base references and constants used by the function DALLOC. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

<table>
<thead>
<tr>
<th>Data Base References</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Global References</strong></td>
</tr>
<tr>
<td>BMT_SIZE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
</tr>
<tr>
<td>ASR</td>
</tr>
<tr>
<td>INC</td>
</tr>
<tr>
<td>JMP</td>
</tr>
<tr>
<td>MOV</td>
</tr>
<tr>
<td>SUB</td>
</tr>
</tbody>
</table>

3.2.38.5 **Limitations**

None
3.2.38.6 Listing

STMT SOURCE STATEMENT

1 R0=%0
2 R1=%1
3 R2=%2
4 R3=%3
5 R4=%4
6 R5=%5
7 R6=%6
8 PC=%7
9 :
10 : DISK_ADR := DALLOC(BMT_ADR);
11 :
12 :
13 DALLOC: NOP
14 MOV R6,R5
15 ADD #4,R5
16 MOV PC,(R5)+
17 SUB #2,R6
18 MOV R4,-(R6)
19 MOV -(R5),R0
20 MOV @0,R1
21 MOV (R1)*,R2
22 MOV (R1),R3
23 MOV #177777,R4
24 :
25 WLOOP: MOV (R2)*,R1
26 CMP R1,R4
27 BNE WFOUND
28 CMP R2,R3
29 BLE WLOCP
30 HALT
31 :
32 WFOUND: SUB #2,R2
33 MOV R2,R3
34 SUB (R0),R3
35 :
36 BLOOP: INC R4
37 ASR R1
38 BCS BLOOP
39 :
40 : NOW MUST TURN BIT ON IN MAP
41 :
42 MOV #1,R1
43 .WORD 072104 ;ASH R1,R4
44 BIS R1,(R2)
45 :
46 : TRANSLATE (WORD#, BIT#, SIZE) INTO A DISK ADDRESS
47 :
48 .WORD 072327 ;ASH R3,3
49 .WORD 3
50 ADD R4,R3
51 MOV 6,(R0),R2
52 .WORD 072302 ;ASH R3,R2
3.2.39 Free Disk (DFREE)

The Free Disk CPC, DFREE, is a kernel level internal SKCPP function that is called by another kernel level internal function. It is written in PAL-11 assembly language.

3.2.39.1 Description

DFREE translates a disk address into a bit address and resets the appropriate bit to mark the disk area free. It subtracts the base disk address in the bit maps table from the disk address supplied to obtain the relative address of the disk area to be freed. Shifting the relative address to the right by the shift factor is the equivalent of dividing by the size: it yields the number of the segment within the disk space. DFREE then performs a combined right shift on this result. This separates the four low-order bits, identifying the bit number, from the high order bits, which identify the word# of the segment's bit in the bit map. The bit# is then moved to the right, and since the left-most bit is duplicated in the right shift, the 12 left bits of the bit# are cleared. DFREE creates a word, TEMP1, with only the bit corresponding to the bit# set to 1. It finds the relative byte address of word# by multiplying word# by 2, and the absolute address, TEMP2 by adding the bit map starting address which it finds in the bit maps table. It then uses TEMP1 to clear the bit at TEMP2 which corresponds to the area to be freed. This updates the bit map and marks the disk space not in use.

Function: DISK_FREE
Parameters: DISK_FREE(disk_address, size)
Effect:
Let k = (disk_address - BASE(size))/size;
BIT_MAP(size, k) = 0;
3.2.39.2 Flow Chart

DFREE

TEMP2 = DISK_ADR

TEMP0 = BMT_ADR

TEMP2 = TEMP2 - BASE_DISK_ADDRESS

TEMP1 = SHIFT_FACTOR

TEMP1 = -TEMP1

TEMP2 = TEMP2 * 2^TEMP1

TEMP2, TEMP3 = TEMP2 * 2^-4

TEMP3 = TEMP3 * 2^-12

TEMP3 = TEMP3^2 - 16 - 15

TEMP1 = 1

TEMP1 = TEMP1 * 2^TEMP3

TEMP2 = TEMP2 * 2

TEMP2 = TEMP2 + BIT_MAP_START_ADR @ TEMP0

BIT_MAP @ TEMP2 = BIT_MAP @ TEMP2^2 - TEMP1

RETURN
3.2.39.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETSEG</td>
<td>None</td>
</tr>
</tbody>
</table>

3.2.39.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function DFREE. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

<table>
<thead>
<tr>
<th>Data Base References</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Global References</strong></td>
</tr>
<tr>
<td>BMT_SIZE</td>
</tr>
</tbody>
</table>

**Constants**

ADD
ASL
JMP
MOV
NEG
SUB

3.2.39.5 Limitations

None
### 3.2.39.6 Listing

<table>
<thead>
<tr>
<th>STMT</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R0=S0</td>
</tr>
<tr>
<td>2</td>
<td>R1=S1</td>
</tr>
<tr>
<td>3</td>
<td>R2=S2</td>
</tr>
<tr>
<td>4</td>
<td>R3=S3</td>
</tr>
<tr>
<td>5</td>
<td>R4=S4</td>
</tr>
<tr>
<td>6</td>
<td>R5=S5</td>
</tr>
<tr>
<td>7</td>
<td>R6=S6</td>
</tr>
<tr>
<td>8</td>
<td>PC=S7</td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>DFREE(DISK_ADDR, BMT_ADDR);</td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>DFREE:</td>
</tr>
<tr>
<td>13</td>
<td>MOV R6,R5</td>
</tr>
<tr>
<td>14</td>
<td>ADD #6,R5</td>
</tr>
<tr>
<td>15</td>
<td>MOV PC,(R5)+</td>
</tr>
<tr>
<td>16</td>
<td>SUB #2,R6</td>
</tr>
<tr>
<td>17</td>
<td>MOV -(R5),R2</td>
</tr>
<tr>
<td>18</td>
<td>MOV -(R5),R0</td>
</tr>
<tr>
<td>19</td>
<td>SUB 4,(R0),R2</td>
</tr>
<tr>
<td>20</td>
<td>MOV 6,(R0),R1</td>
</tr>
<tr>
<td>21</td>
<td>NEG R1</td>
</tr>
<tr>
<td>22</td>
<td>.WORD 072201</td>
</tr>
<tr>
<td>23</td>
<td>;SRL R2,R1</td>
</tr>
<tr>
<td>24</td>
<td>.WORD 073227</td>
</tr>
<tr>
<td>25</td>
<td>;ASHC R2,-4</td>
</tr>
<tr>
<td>26</td>
<td>.WORD -4</td>
</tr>
<tr>
<td>27</td>
<td>.WORD 072327</td>
</tr>
<tr>
<td>28</td>
<td>;ASHC R3,-12</td>
</tr>
<tr>
<td>29</td>
<td>.WORD -14</td>
</tr>
<tr>
<td>30</td>
<td>BIC #177760,R3</td>
</tr>
<tr>
<td>31</td>
<td>;ASH IS ARITHMETIC, NOT LOGICAL, SHIFT</td>
</tr>
<tr>
<td>32</td>
<td>MOV #1,R1</td>
</tr>
<tr>
<td>33</td>
<td>;ONE 1 BIT</td>
</tr>
<tr>
<td>34</td>
<td>.WORD 072103</td>
</tr>
<tr>
<td>35</td>
<td>;ASH R1,R3</td>
</tr>
<tr>
<td>36</td>
<td>.ASL R2</td>
</tr>
<tr>
<td>37</td>
<td>;WORD# TO BYTE#</td>
</tr>
<tr>
<td>38</td>
<td>ADD (R0),R2</td>
</tr>
<tr>
<td>39</td>
<td>;ADD IN START ADDRESS OF BIT MAP</td>
</tr>
<tr>
<td>40</td>
<td>JMP a-10(R5)</td>
</tr>
<tr>
<td>41</td>
<td>;DONE</td>
</tr>
<tr>
<td>42</td>
<td>END DFREE</td>
</tr>
</tbody>
</table>

### 3.2.40 Sleep (SLEEP)

The Sleep CPC, SLEEP, is a kernel level internal SKCPP function that is called by both another kernel level internal function and user level external functions. SLEEP calls a kernel level internal function. It is written in Project SUE System Language, including the Inline feature.

#### 3.2.40.1 Description

SLEEP finds the next process which is ready to run. Using Inline code, it sets the priority level low to enable interrupts. Then it loops through the processes, starting with NEXT_PROCESS equal to THE_CURRENTPROCESS + 1, looking for a ready process. It begins the cycle by checking if NEXT_PROCESS is greater than PROCESS# MAX. If so, it has reached the last process and goes back to the beginning...
by letting NEXT_PROCESS equal PROCESS_MIN. Next, if PT_FLAGS (NEXT_PROCESS) logical and PT_FLAGS_MASK equals READY, it has found a ready process and leaves the process finding loop. Otherwise, it increments NEXT_PROCESS and keeps looking.

If NEXT_PROCESS, the ready process it found, is not THE_CURRENT_PROCESS, SLEEP calls RUN. The segmentation descriptors of the current process are saved in the PT and those of NEXT_PROCESS, which becomes the new current process, are loaded. Otherwise, SLEEP simply returns control to the calling program.

Function: SLEEP
Parameters: SLEEP
Effect:
IF ((#process#)(PT_FLAGS(process#) = READY);
THEN:
  IF process# = TCP;
  THEN: RUN(process#);
  END;
ELSE: SLEEP
END;

3.2.40.2 N/A
3.2.40.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>STOPP</td>
<td>RUN</td>
</tr>
<tr>
<td>IPCRCV</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
</tbody>
</table>

3.2.40.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function SLEEP. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph, 3.3.2.

<table>
<thead>
<tr>
<th>Data Base References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global References</td>
</tr>
<tr>
<td>PT_FLAGS</td>
</tr>
</tbody>
</table>

164
3.2.40.5 Limitations

None

3.2.40.6 Listing

DATA SLEEP;
DECLARE
    WORD (NEXT_PROCESS);

PROGRAM SLEEP;
INLINE(SPLLOW);
/* FIND NEXT PROCESS THAT CAN RUN - ROUND ROBIN SCHEDULER */
NEXT_PROCESS := THE_CURRENT_PROCESS + 1;
CYCLE
    IF NEXT_PROCESS > PROCESS#_MAX;
        THEN: NEXT_PROCESS := PROCESS#_MIN;
    END;
    IF (PT_FLAGS(NEXT_PROCESS) & PT_FLAGS_MASK) = READY;
        THEN:
            EXIT;
        END;
    NEXT_PROCESS := NEXT_PROCESS + 1;
END;
    IF NEXT_PROCESS = THE_CURRENT_PROCESS;
        THEN: RUN(NEXT_PROCESS);
    END;

3.2.41. Run (RUN)

The Run CPC, RUN, is a kernel level internal SKCFF function that is called by another kernel level internal function. RUN calls only kernel level internal functions. It is written in Project SUE System Language.
3.2.41.1 Description

RUN saves information about the current process and prepares to run the next process. Using Inline code, it sets the priority level high to block interrupts.

It then looks for changes, looping through all segmentation registers, REG# going from 0 to REG#_MAX. It sets X to the correct number to access the segmentation registers: REG# + REG_CONSTANT if REG# is greater than CROSS_REG# and REG# otherwise. RUN then checks if the change bit is set, if SDR(X) logical and SDR_CHANGE_MASK equals SDR_CHANGED. If so, VAR is assigned the value of SAR(X), the main memory address of the segment. Since the segmentation registers omit the six low order bits of the address (assumed to be 0) and the memory block table omits the eight low order bits thereof, two arithmetic shift rights are performed, using Inline code, on VAR. Then, if VAR is less than END_BLOCK#, the change bit in the AST entry for the segment is set. VAR gets the aste# from MBT_AST(VAR). AST_CHANGE(VAR) is then reset to AST_CHANGE(VAR) logical or MBT_CHANGED. RUN then continues its loop through the segmentation registers.

Function: RUN
Parameters: RUN(process#)
Effect:
IF (reg#)(SDR(reg#)) & SDR_CHANGE_MASK = SDR_CHANGED;
THEN:
Let VAR = 4*SAR(reg#);
Let block# = MBT_AST(var);
AST_CHANGE(block#) = 'AST_CHANGE'(block#) AST_CHANGED;
END;
PT_KSAR1(TCP) = KSAR1;
PT_KSAR1(TCP) = KSDR1;
LSD(PG_PS_ASTE#(process#) PS_KSR_ASR,WRITE);

Next, RUN saves the kernel segmentation register 1. It sets (THE_CURRENT_PROCESS) to KSDR1 and PT_KSAR1(THE_CURRENT_PROCESS) to KSAR1. It then loads the descriptor for the next process' PS at PS_KSAR_ADR with a call to LSD. Finally, RUN calls SWAP, to make the next process the current process.

3.2.41.2 N/A
3.2.41.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLEEP</td>
<td>LSD</td>
</tr>
<tr>
<td></td>
<td>SWAP</td>
</tr>
</tbody>
</table>

3.2.41.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function RUN. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT_KSAR1</td>
<td>NEXT_PROCESS</td>
<td>X</td>
</tr>
<tr>
<td>PT_KSDR1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AST_CHANGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MBT_ASTE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THE_CURRENT_PROCESS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constants

<table>
<thead>
<tr>
<th>ASR</th>
<th>REG#_MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST_CHANGE</td>
<td>SDR_CHANGE_MASK</td>
</tr>
<tr>
<td>CROSS_REG#</td>
<td>SDR_CHANGED</td>
</tr>
<tr>
<td>END_BLOCK#</td>
<td>SDR_WRITE_ACCESS</td>
</tr>
<tr>
<td>PS_KSR_ADR</td>
<td>SPLHIGH</td>
</tr>
<tr>
<td>REG_CONSTANT</td>
<td>SPLLOW</td>
</tr>
</tbody>
</table>

3.2.41.5 Limitations

None

3.2.41.6 Listing

167
DATA RUN(NEXT_PROCESS):
DECLARE
PROCEDURE ACCEPTS (WORD, WORD) (SWAP),
WORD (REG#, X, VAR);

PROGRAM RUN:

/* BLOCK INTERRUPTS */
INLINE(SPLHIGH):

/* NO NEED TO SAVE PROCESS REGISTERS BECAUSE A COPY IS IN THE PROCESS SEGMENT, */
* BUT CHANGE BIT MUST BE INSPECTED */

DO REG# := 0 TO REG#_MAX;
    IF REG# > CROSS_REG#;
        THEN: X := REG# + REG_CONSTANT;
        FLST: X := REG#;
    END;
    IF (SDR(X) & SDR_CHANGE_MASK) = SDR_CHANGED;
        THEN: VAR := SAR(X);
        INLINE(ASR, VAR); /* THEIR BLOCKS TO MINE */
        INLINE(ASR, VAR);
        IF VAR < END_BLOCK#;
            THEN: VAR := MMT_ASTE(VAR);
            AST_CHANGE(VAR) := (AST_CHANGE(VAR) | AST_CHANGED);
        END;
    END;

END;

/* SAVE KSR */
PT_KSAR1(THE_CURRENT_PROCESS) := KSAR1;
PT_KSDR1(THE_CURRENT_PROCESS) := KSDR1;

/* LOAD DESCRIPTOR FOR NEXT PROCESS'S PS */
LSD(PT_PS_ASTE#(NEXT_PROCESS), PS_KSR_ADR, SDR_WRITE_ACCESS);

/* CLEANER (AND FASTER) TO DO REAL SWAP IN PAL */
SWAP(THE_CURRENT_PROCESS, NEXT_PROCESS):
INLINE(SPLLOW);

3.2.42 Swap (SWAP)

The Swap CPC, SWAP, is a kernel level internal SKCPC function that is called by one other kernel level internal function. It is written in PAL-11 assembly language.

3.2.42.1 Description

SWAP switches from the current process supplied to the next process, as specified. It multiplies the current process parameter by 2 to access the arrays of 2-byte words. It then saves the contents
of general purpose registers 0 to 6 in the process table.

Next, it enters NEXT_PROCESS at the address of TCP, as the current process. The next process parameter is then doubled to access the arrays of words.

SWAP then loads the segmentation registers from the new current process’s process segment. It loops through the supervisor registers, setting SAR0 through SAR7 equal to PS_SAR(0) through PS_SAR(7). Similarly it loops through the user registers and loads SAR8-15 and SDR8-15.

It then loads kernel segmentation registers KSR1, 2, and 3 from the current process’s entry in the process table. General purpose registers 4 through 6 are also filled in from the process table.

SWAP now checks if the current process is a new process. If R5 is non-zero, the process is not new, and SWAP returns to the kernel. Otherwise, it prepares to return out to the user. It clears registers 0 through 3. To R4 it assigns a pointer to the static link, and R6 gets the kernel stack pointer. The user PSW, the PC, the user R6, and the pointer to the static link are pushed onto the stack. The pointer to the static link is then popped from the stack to the static link and the user R6 is popped from the stack to R6 in supervisor mode. A return from interrupt is then executed, restoring the supervisor’s RC and PSW from the kernel stack.

Function: SWAP
Parameters: SWAP(TCP_process#, process#)
Effect:
PT_KSDR3(TCP_process#) = KSDR3;
PT_KSAR3(TCP_process#) = KSAR3;
PT_R4(TCP_process#) = R4;
PT_R5(TCP_process#) = R5;
PT_R6(TCP_process#) = R6;
(Vreg#)((PS_SAR(process#, reg#) = SAR(reg#)) &
(PS_SDR(process#, reg#) = SDR(reg#)))
KSDR1 = PT_KSDR1(process#);
KSAR1 = PT_KSAR1(process#);
KSDR2 = PT_KSDR2(process#);
KSAR2 = PT_KSAR2(process#);
KSDR3 = PT_KSDR3(process#);
KSAR3 = PT_KSAR3(process#);
R4 = PT_R4(process#);
R5 = PT_R5(process#);
TCP = process#;
IF R5 = 0
THEN: R0 = 0;
     R1 = 0;
     R2 = 0;
     R3 = 0;
     R4 = "3E0";
     R6 = "43FE";
     SUPERV STATIC_LINK +"3E0";
     SR6 = "3DC";
     PSE = "7000";
     PC = "4000";

END;
3.2.42.2 Flow Chart

SWAP

TEMP0 = CURRENT PROCESS

TEMP0 = TEMP0 * 2

PT_KDR3 @ (TEMP0 + PTKDR3) = @ KSDR3
PT_KAR3 @ (TEMP0 + PTKAR3) = @ KSAR3

PT_R4 @ (TEMP0 + PTR4) = R4
PT_R5 @ (TEMP0 + PTR5) = R5
PT_R6 @ (TEMP0 + PTR6) = R6

TEMP0 = NEXT_PROCESS

THE_CURRENT_PROCESS @ TCP = TEMP0

TEMP0 = TEMP0 * 2

TEMP1 = PSSAR
TEMP2 = PSSDR
TEMP3 = SAR0
TEMP4 = SDR0
TEMP5 = 8

©

@ TEMP3 = @ TEMP1
TEMP1 = TEMP1 + 2
TEMP3 = TEMP3 + 2

@ TEMP4 = @ TEMP2
TEMP2 = TEMP2 + 2
TEMP4 = TEMP4 + 2

TEMP5 = TEMP5 - 1

IS TEMP5 = \emptyset

No

Yes

TEMP3 = SAR8
TEMP4 = SDR8
TEMP5 = 8

@ TEMP3 = @ TEMP1
TEMP1 = TEMP1 + 2
TEMP3 = TEMP3 + 2

@ TEMP4 = @ TEMP2
TEMP2 = TEMP2 + 2
TEMP4 = TEMP4 + 2

TEMP5 = TEMP5 - 1

IS TEMP5 = \emptyset

No

Yes

171
3.2.42.2 Flow Chart (Concluded)

@ KSDR = @ (TEMP0 + PTKDR1)
@ KSAR = @ (TEMP0 + PTKAR1)
@ KSDR = @ (TEMP0 + PTKDR2)
@ KSAR = @ (TEMP0 + PTKAR2)
@ KSDR = @ (TEMP0 + PTKDR3)
@ KSAR = @ (TEMP0 + PTKAR3)

R4 = PT_R4 (TEMP0 + PTR4)
R5 = PT_R5 (TEMP0 + PTR5)
R6 = PT_R6 (TEMP0 + PTR6)

R0 = 0
R1 = 0
R2 = 0
R3 = 0
R4 = 3E0
R5 = 43FE
R6 = USER PSW = 7000
R6 = PC = 4000
R6 = USER R6 = 03DC
R6 = PSW = 11A0

STATIC LINK = R6
R6 = R6 + 2
SUPERVISOR R6 = R6
R6 = R6 + 2

RETURN FROM INTERRUPT
3.2.42.3 Interfaces

Refer to Figure 6, Function Call Matrix, in paragraph 3.4.

<table>
<thead>
<tr>
<th>Called By</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>None</td>
</tr>
</tbody>
</table>

3.2.42.4 Data Organization

Listed below are Security Kernel data base references and constants used by the function SWAP. For data base references refer to Figure 5, Data Base Reference Matrix, in subparagraph 3.3.1. For constants refer to Table I, List of Constants, in subparagraph 3.3.2.

Data Base References

<table>
<thead>
<tr>
<th>Global References</th>
<th>Function Parameters</th>
<th>Local References</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT_KSDR1</td>
<td>CURRENT_PROCESS</td>
<td>PTKDR1</td>
</tr>
<tr>
<td>PT_KSAR1</td>
<td>NEXT_PROCESS</td>
<td>PTKAR1</td>
</tr>
<tr>
<td>PT_KSDR2</td>
<td></td>
<td>PTKDR2</td>
</tr>
<tr>
<td>PT_KSAR2</td>
<td></td>
<td>PTKAR2</td>
</tr>
<tr>
<td>PT_KSDR3</td>
<td></td>
<td>PTKDR3</td>
</tr>
<tr>
<td>PT_KSAR3</td>
<td></td>
<td>PTKAR3</td>
</tr>
<tr>
<td>PT_R4</td>
<td></td>
<td>PTR4</td>
</tr>
<tr>
<td>PT_R6</td>
<td></td>
<td>PTR5</td>
</tr>
<tr>
<td>PS_SAR</td>
<td></td>
<td>PTR6</td>
</tr>
<tr>
<td>PS_SDR</td>
<td></td>
<td>KSDR1</td>
</tr>
<tr>
<td>SDR</td>
<td></td>
<td>KSAR1</td>
</tr>
<tr>
<td>SAR</td>
<td></td>
<td>KSDR2</td>
</tr>
<tr>
<td>PSW</td>
<td></td>
<td>KSAR2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KSDR3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KSAR3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAR0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SDR0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAR8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SDR8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TCP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PSW</td>
</tr>
</tbody>
</table>

Constants

ADD     DEC
ASL     JMP
CLR     MOV

173
3.2.42.5 Limitations

None

3.2.42.6 Listing

```
STMT SOURCE STATEMENT

1 R0=0
2 R1=1
3 R2=2
4 R3=3
5 R4=4
6 R5=5
7 R6=6
8 PC=7
9 :
10 ; SWAP(CURRENT_PROCESS, NEXT_PROCESS);
11 :
12 :
13 ; RELEVANT DEFINITIONS IN PT
14 :
15 PTKDR1=17000
16 PTKAR1=17040
17 PTKDR2=17100
18 PTKAR2=17140
19 PTKDR3=17200
20 PTKAR3=17240
21 PTR4=17300
22 PTR5=17340
23 PTR6=17400
24 :
25 ; AND PS
26 :
27 PSSDR=20000
28 PSSAR=20040
29 :
30 ; HARDWARE REGS
31 :
32 KSAR1=172302
33 KSAR1=172304
34 KSAR2=172304
35 KSAR2=172304
36 KSAR3=172304
37 KSAR3=172304
38 SAF0=172240
39 SDR0=172240
40 SAR0=177640
41 SDR0=177600
42 :
43 ; AND FINALLY
44 :
45 TCP=16544 ; THE CURRENT PROCESS
46 PSW=177776
47 :
48 SWAP: MOV R6, R5 ; SUE
49 ADD #6, R5 ; ENTRY
50 MOV PC, (R5)+ ; SEQUENCE
51 MOV 1-R(R5), R0 ; CURRENT_PROCESS
52 ASL RO ; ARRAYS ARE OF WORDS
```

174
SAVE KSR3 OF CURRENT PROCESS IN PT

MOV @KSDR3,PTKDR3(R0)
MOV @KSTB3,PTKKB3(R0)

SAVE GPR4-6 IN PT

MOV R4,PTKBR4(R0)
MOV R5,PTKBR5(R0)
MOV R6,PTKBR6(R0)
MOV -6(R5),R0 ; NEXT_PROCESS
MOV R0,#TCP ; SAVE
ASL R0 ; ARRAYS ARE WORDS

LOAD SR0-15 FOR NEXT PROCESS FROM ITS PS

FIRST SR0-7

MOV #PSSAR,R1 ; BASE ADDRESS OF PS_SAR ARRAY
MOV #ESSDR,R2 ; PS_SDR
MOV #SAR0,R3 ; BASE ADDRESS OF SUPERVISOR
MOV #SDR0,R4 ; SEGMENTATION REGISTERS (SR0-7)
MOV #8,R5 ; LOOP CONTROL
SR07: MOV (R1)+,(R3)+
MOV (R2)+,(R4)+
DEC R5
BNE SR07

NOW SR8-15

MOV #SAR8,R3 ; BASE ADDRESS OF USER
MOV #SDR8,R4 ; SEGMENTATION REGISTERS (SR8-15)
MOV #8,R5 ; LOOP CONTROL
SR815: MOV (R1)+,(R3)+
MOV (R2)+,(R4)+
DEC R5
BNE SR815

LOAD KSR1, 2, 63 FROM PT

MOV PTKDB1(R0),@KSDR1
MOV PTKKB1(R0),@KSTB1
MOV PTKBR2(R0),@KSDR2
MOV PTKBR2(R0),@KSTB2
MOV PTKDR3(R0),@KSDR3
MOV PTKDR3(R0),@KSTB3

AND GPR4-6

MOV PTR4(R0),R4
MOV PTR5(R0),R5
MOV PTR6(R0),R6

: IF THIS IS NOT A NEW PROCESS RETURN INTO KERNEL
ELSE RETURN OUT TO USER

MOV R5,R5
BEQ NEW

.WORD 230 ; SPL LOW
JMP @-10(R5)

NEW:
CLR R0
CLR R1
CLR R2
CLR R3

MOV #1740,R4 ; "3E0"
MOV #41776,R6 ; KERNEL STACK POINTER - "43FE"
MOV #070000,-(R6) ; USER PSW - CM=S, PM=U
MOV #040000,-(R6) ; PC - "4000"
MOV #001734,-(R6) ; USER R6
MOV R4,-(R6) ; POINTER TO STATIC LINK
MOV #010340,#PSW ; FOR NEXT INSTRUCTION
.WORD 006637 ; MTP1
.WORD 1740 ; POINTER TO STATIC LINK
.WORD 006606 ; MTP1 R6 ; SET SUPERVISOR MODE R6
RTI

.END SWAP
3.3 Storage Allocation

The Security Kernel consists only of its data base and its code. The Executive and Listener programs as well as the Executive stacks and root directory, while necessary for its running are not actually part of the Security Kernel. The Editor and Exerciser (which allow user interaction) while desirable are not necessary to the running of the Security Kernel; their core space may be used for storing directory and data segments. A memory map is shown in Figure 4.

Of the 64K words of core storage available, the Security Kernel data base occupies 5K words and its code occupies 8.75k words. The root directory requires 0.5K words of core and the hardware registers 4K words of high core. System programs - namely, the Executive (including its work space), and the Listener - occupy 5.625K words of core. The Editor and Exerciser, if present, require an additional 13.5K words of storage space. This leaves about 40K words of core (exclusive of the Editor and Exerciser). This space is allocated consecutively in blocks of 1K bytes; that is, each segment, as it is swapped into main memory, is assigned the lowest free block of storage. No timing requirements are imposed. The only equipment constraint affecting storage allocation is that only 8K bytes may be addressed at a time; hence the Security Kernel and Editor code is stored in smaller segments.

3.3.1 Data Base Characteristics

The Security Kernel global data base structures, generally speaking, fall into two categories: need-to-know and resource management.

Included in the need-to-know category are:

a. Directories, which have a fixed part (DIR_) and a variable part (ACL_), are access matrices which describe access permissions.

b. The Active Segment Table (AST_) is the record of current access. Functions read the directories to fill in some of the information contained in the AST.

c. The Process Table (PT_) and Process Segment (PS_) contain control information about a process. The PT contains information on all segments whereas the PS contains information on a specific segment. The PT contains some information which appears in the PS.
Figure 4. MEMORY MAP (ALL ADDRESSES HEXADECIMAL)
Included in the resource management category are:

a. The Memory Block Table (MBT) which indicates the state of main memory. Entries in the MBT are Active Segment Table Entry numbers (ASTE#).

b. The Bit Map Table (MBT) deals with disk allocation.

c. The Hash Table has but one entry (HASH TABLE) which points to the beginning of a chain of ASTEs.

d. The Interprocess Communication Element Pool (IPC) contains elements which are controlled by a quota mechanism. The head of the IPC queue is located in the PT.

e. Semaphore entries are arrays of 0 to 257. There is one semaphore, which is indexed by ASTE#, for each segment. The head of the chain of blocked processes on a semaphore is located in the PT.

f. Parameters is a table of user input parameters.

g. Segmentation Address Registers (SAR) and Segmentation Descriptor Registers (SDR) are register pairs containing information fully describing a segment. SARs reference the MBT for their information and the SDRs reference the AST entry SIZE.

The Data Base Reference Matrix presented as Figure 5 shows the global data base and the functions that reference each individual structure. Reading across shows which data structures are referenced by that specific function. Reading down shows which functions reference that specific data structure.

The following subparagraphs contain detailed definitions of the contents of the Security Kernel data structures. The address spaces referred to in each subparagraph are the virtual addresses of the specified segmentation register. CONTEXT KERNEL, in Section 10, contains a complete definition of the Security Kernel virtual address space. (Refer to Section 3 paragraph 4 for a description of the dynamic address translation performed by the MMU.)
Figure 5  DATA BASE REFERENCE MATRIX

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DIRECTORIES</th>
<th>ACL</th>
<th>AST</th>
<th>PT</th>
<th>PS</th>
<th>MBT</th>
<th>HASH TABLE</th>
<th>IPC ELT POOL</th>
<th>SEMAPHORES</th>
<th>SDR &amp; SAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>GATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCHECK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CREATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GIVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESCIND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUTERP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUTERV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STARTP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STOPP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHANGEO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INITH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READIR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPCRCV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPCSEND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRI TEDIR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DALLOC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETDIR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETSEG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOADD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DFREE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONNECT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCONNECT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HASH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSEARCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEACT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PREHASH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWAPIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWAPOUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INITSEG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISKIO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLEEP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWAP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.3.1.1 Directories

Functions that reference the directories either read or write the directory entries. In order to do anything with a segment, e.g., create it, delete it, give access to it, the directory that catalogues the segment must be referenced. A directory is a segment of entries that contain the attributes of some other segment. Directories have a fixed part and a variable part. The fixed part is filled in at the time the segment is CREATED. The field names for this part of the directories begin with "DIR_". The variable part is known as the Access Control List. Its field names begin with "ACL_". The ACL is an open-ended list of names of users permitted to access the segment and implement the need-to-know protection. Directories are located in KSR3 address space (refer to Section 10, page 15).

Format of a Directory Entry
(fixed part)

A directory entry is accessed by (aste#, offset): DIR_XXX(aste#, offset)

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NO. of BITS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIR_TYPE</td>
<td>1</td>
<td>DIRECTORY or DATA</td>
</tr>
<tr>
<td>DIR_STATUS</td>
<td>1</td>
<td>UNINITIALIZED OR INITIALIZED</td>
</tr>
<tr>
<td>DIR_CLASS</td>
<td>4</td>
<td>security classification</td>
</tr>
<tr>
<td>DIR_CAT</td>
<td>16</td>
<td>security category set</td>
</tr>
<tr>
<td>DIR_SIZE</td>
<td>8</td>
<td>segment size in blocks</td>
</tr>
<tr>
<td>DIR_DISK</td>
<td>16</td>
<td>disk address of the segment</td>
</tr>
<tr>
<td>DIR_ACL_HEAD</td>
<td>8</td>
<td>head of the ACL chain (or Ø if the list is empty)</td>
</tr>
</tbody>
</table>

DIR_TYPE specifies the type attribute of the segment. Its value is either DIRECTORY (a 1-bit) or DATA (a Ø-bit).

DIR_STATUS indicates whether or not a segment has been initialized. Its value is either UNINITIALIZED (a 1-bit) or INITIALIZED (a Ø-bit).
DIR_CLASS is the classification part of the security attribute. It has values of 1 through 4; 1 equals unclassified, 2 equals confidential, 3 equals secret, and 4 equals top secret.

DIR_CAT is the category set which is the rest of the security attribute. It has possible values of $\emptyset$ through 32767.

DIR_SIZE is the size of the segment in a multiple of 256 bytes. If the value of DIR_SIZE is zero, the directory entry is not being used and the value of all other fields are undefined.

DIR_ACL_HEAD is the head of a chain of ACL elements. If there are no ACL elements then DIR_ACL_HEAD is zero.

**Formal of an Access Control List (ACL) Element**

(variable part)

An ACL element is accessed by (aste#, acle#): ACL_XXX (aste#, acle#)

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NO. of BITS</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL_USER</td>
<td>14</td>
<td>user-id or ALL_USERS</td>
</tr>
<tr>
<td>ACL_PROJECT</td>
<td>8</td>
<td>project_id or ALL_PROJECTS</td>
</tr>
<tr>
<td>ACL_MODE</td>
<td>2</td>
<td>mode of access - WRITE, READ, or NO access</td>
</tr>
<tr>
<td>ACL_CHAIN</td>
<td>8</td>
<td>acle# of next ACL in the chain or $\emptyset$</td>
</tr>
</tbody>
</table>

Whenever a user is on the system the state information of his process includes a two part name identifier - user_id and project_id. An ACL element includes this two part name but either part may be replaced by a flag indicating "don't care". This special flag is represented by ALL_USERS or ALL_PROJECTS.

Each ACL element, in addition to a name, has a permitted mode of access - no access, read access, or write access. The access mode is associated with the ACL element rather than the segment itself to allow different users to have different access rights to the segment. ACL elements are ordered from most significant to least significant. Elements with a specific user_id and project-id come first, an ALL_PROJECTS element will always be last, and elements with a specific user and ALL_PROJECTS will come before an ALL_USERS, specific element.
A directory segment has 63 usable entries (numbered 1 to 63) plus an unusable entry (entry 0) and 127 ACL elements that are shared among all entries. The sharing mechanism employs a chain of free ACL elements - the head of this free chain is ACL_CHAIN(0). A directory is initialized by marking all its entries as free and placing all the ACL elements on the free chain.

All segment attributes except for DIR_STATUS and DIR_DISK are specified by users with write access to the directory and therefore have the security level of the parent directory, but the values of DIR_STATUS and DIR_DISK are a function of system wide activity.

Directories are considered to be "composite" objects. Most of the data in a directory will be at the security level of the directory but some will be at a higher level. The format of the directory is defined within the security perimeter so there is no problem in determining the security level of a particular data item. Since the segment is the smallest object to which access is controlled by the MMU, uncertified software cannot be permitted direct read access to directory segments. If uncertified software is to have read access to a directory it must be via Security Kernel functions that do the reading interpretively and are aware of the nature of the directories.

3.3.1.1.1 Functions Using Directories and Access Control List

<table>
<thead>
<tr>
<th>Directories</th>
<th>Access Control List</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>GIVE</td>
</tr>
<tr>
<td>DELETE</td>
<td>RESCIND</td>
</tr>
<tr>
<td>GIVE</td>
<td>DELETSEG</td>
</tr>
<tr>
<td>RESCIND</td>
<td>DESEARCH</td>
</tr>
<tr>
<td>STARTP</td>
<td>INITSEG</td>
</tr>
<tr>
<td>CHANGED</td>
<td></td>
</tr>
<tr>
<td>INITH</td>
<td></td>
</tr>
<tr>
<td>READIR</td>
<td></td>
</tr>
<tr>
<td>GETW</td>
<td></td>
</tr>
<tr>
<td>GETR</td>
<td></td>
</tr>
<tr>
<td>DELETSEG</td>
<td></td>
</tr>
<tr>
<td>SOADD</td>
<td></td>
</tr>
<tr>
<td>CONNECT</td>
<td></td>
</tr>
<tr>
<td>DESEARCH</td>
<td></td>
</tr>
</tbody>
</table>

183
### 3.3.1.2 Active Segment Table (AST)

The Active Segment Table is a system-wide table that facilitates the main memory sharing of segments among processes. Every segment that is in the work space (WS) of one or more processes or is wired down (a permanent location in core dedicated to the segment) has an entry in the AST. The segment is identified by its aste# (AST entry #). An ASTE is composed of a number of fields. The AST is located in KSRØ address space (refer to Section 10, pages 11 and 12).

**Format of an Active Segment Table Entry**

An AST entry is accessed by aste#: **AST Xxx(aste#)**

<table>
<thead>
<tr>
<th>FIELD</th>
<th>No. of BITS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST_TYPE</td>
<td>1</td>
<td>DIRECTORY or DATA</td>
</tr>
<tr>
<td>AST_STATUS</td>
<td>1</td>
<td>UNINITIALIZED or INITIALIZED</td>
</tr>
<tr>
<td>AST_CLASS</td>
<td>4</td>
<td>security classification</td>
</tr>
<tr>
<td>AST_CAT</td>
<td>16</td>
<td>security category set</td>
</tr>
<tr>
<td>AST_SIZE</td>
<td>8</td>
<td>segment size in blocks</td>
</tr>
<tr>
<td>AST_DISK</td>
<td>16</td>
<td>disk address of the segment</td>
</tr>
<tr>
<td>AST_CHANGE</td>
<td>1</td>
<td>segment altered or unaltered while in core</td>
</tr>
<tr>
<td>AST_CPL</td>
<td>16</td>
<td>connected process list</td>
</tr>
<tr>
<td>AST_WAL</td>
<td>16</td>
<td>write access list</td>
</tr>
<tr>
<td>AST_AGE_CHAIN</td>
<td>16</td>
<td>chain for segments eligible for deactivation</td>
</tr>
<tr>
<td>AST_ADR</td>
<td>16</td>
<td>main memory address of a segment</td>
</tr>
<tr>
<td>AST_DES_COUNT</td>
<td>16</td>
<td>number of descriptors for a segment</td>
</tr>
<tr>
<td>AST_UNLOCK</td>
<td>1</td>
<td>UNLOCKED - AST_DES_COUNT: Ø</td>
</tr>
<tr>
<td>AST_SWAP_CHAIN</td>
<td>16</td>
<td>chain of segments eligible to be swapped out</td>
</tr>
<tr>
<td>AST_CHAIN</td>
<td>16</td>
<td>used by HASH functions and for ASTE chain</td>
</tr>
</tbody>
</table>
The head of chains is accessed by AST_XXX(0).

AST_TYPE, AST_STATUS, AST_CLASS, AST_CAT, AST_SIZE, and AST_DISK correspond to the similarly named fields in a directory entry. These fields in the ASTE are set by copying from the directory entry at the time the segment is activated.

AST_CHANGE indicates if the segment has been modified while enabled. A 1-bit means the segment has been altered, 0-bit means unaltered.

AST_CPL (connected process list) indicates which processes have the active segment in their WS (read access is implied). AST_WAL (write access list) indicates which processes have write access to the segment as well. AST_CPL and AST_WAL are bit maps. Bit 0 indicates whether or not the segment is wired-down (0 indicates not wired-down, 1 indicates wired-down). When one of the remaining bits is a 1, the corresponding process has access to the segment (AST_CPL-read access, AST_WAL-write access). When a process removes a segment from its WS, AST_CPL may become zero (no processes have the segment in their WS). This means that the segment can be deactivated making the ASTE free.

Segments that can be deactivated (as indicated by a zero AST_CPL) are kept on a chain running through AST_AGE_CHAIN. Since ASTE_WAL is not meaningful then AST_CPL is zero; ASTE_WAL and AST_AGE_CHAIN can physically overlay each other.

AST_ADR is the main memory address of a segment if it is swapped in; AST_ADR will be zero if it is swapped out. Since the beginning main memory address of a segment will always be on a 256 byte boundary, AST_ADR need not include the low order (all zero) 8 bits of the address.

AST_DES_COUNT (descriptor count) indicates the number of descriptors that exist for a segment.

Active segments that are eligible to be swapped out are kept on a chain running through the AST_SWAP_CHAIN field. When a process removes a segment from its AS, AST_DES_COUNT may go to zero. This means the segment has become unlocked and can be removed from main memory.

AST_UNLOCK indicates whether or not a segment is on the AST_SWAP_CHAIN (1-bit indicates UNLOCK, 0-bit indicates LOCK). This one bit field allows the AST_DES_COUNT and AST_SWAP_CHAIN fields to be overlayed.
AST_CHAIN is used to chain together ASTE's that are free. The function HASH also uses the AST_CHAIN field to resolve hashing collisions.

3.3.1.2.1 Functions Using the Active Segment Table

- CREATE
- OUTERP
- OUTERY
- STARTP
- STOPP
- CHANGEO
- INITH
- READIR
- ENABLE
- WRI TEDIR
- GETDIR
- DELETSEG
- READIR
- DISABLE
- SWAPIN
- SWAPOUT
- INITSEG
- RUN
- LSD

3.3.1.3 Process Table (PT)

The Process Table has an entry for each process, and each entry consists of several fields. The PT has an area to hold the basic state of all processes when they are not allocated to the processor. The Process Table is located in KSR0 address space (refer to Section 10, page 14).

Format of the Process Table

The Process Table is accessed by process#: PT_XXX(process#)

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NO. of BITS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT_KSDR1</td>
<td>16</td>
<td>kernel segmentation descriptor register 1</td>
</tr>
<tr>
<td>PT_KDAR1</td>
<td>16</td>
<td>kernel segmentation address register 1</td>
</tr>
<tr>
<td>PT_DSDR2</td>
<td>16</td>
<td>kernel segmentation descriptor register 2</td>
</tr>
<tr>
<td>PT_KSAR2</td>
<td>16</td>
<td>kernel segmentation address register 2</td>
</tr>
<tr>
<td>PT_KSDR3</td>
<td>16</td>
<td>kernel segmentation descriptor register 3</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PT_KSAR3</td>
<td>16</td>
<td>kernel segmentation address register 3</td>
</tr>
<tr>
<td>PT_R4</td>
<td>16</td>
<td>general register 4</td>
</tr>
<tr>
<td>PT_R5</td>
<td>16</td>
<td>general register 5</td>
</tr>
<tr>
<td>PT_R6</td>
<td>16</td>
<td>general register 6</td>
</tr>
<tr>
<td>PT_CUR_CLASS</td>
<td>4</td>
<td>security classifications</td>
</tr>
<tr>
<td>PT_CUR_CAT</td>
<td>16</td>
<td>security categories</td>
</tr>
<tr>
<td>PT_KS_ASTE#</td>
<td>16</td>
<td>aste# of the kernel stack</td>
</tr>
<tr>
<td>PT_PS_ASTE#</td>
<td>16</td>
<td>aste# of the process's process segment</td>
</tr>
<tr>
<td>PT_FLAGS</td>
<td>2</td>
<td>READY, BLOCKED or INACTIVE</td>
</tr>
<tr>
<td>PT_LINK</td>
<td>6</td>
<td>chain of processes blocked on a semaphore</td>
</tr>
<tr>
<td>PT_IPC_QUEUE_HEAD</td>
<td>8</td>
<td>head of the IPC queue</td>
</tr>
<tr>
<td>PT_IPC_QUOTA</td>
<td>8</td>
<td>unused ipc element quota</td>
</tr>
</tbody>
</table>

The first nine entries in the PT are only used when a process becomes blocked. They are written when the process is blocked and read when the process becomes unblocked. The next four entries are fixed fields while the last four entries are variable. These fixed and variable fields of the PT are the current attributes of a process.

PT_KSDR1 and PT_DSAR1 hold the location of the process's process segment.

PT_KSDR2 and PT_KSAR2 hold the location of the process's kernel stack.

PT_KSDR3 and PT_KSAR3 hold the location of the process's current directory segment.

PT_R4, PT_R5, and PT_R6 are general registers whose contents are important to the SUE language. They act as accumulators, stack pointers and temporaries. Register 6 has the special function of the processor stack pointer.

PT_CUR_CLASS is the security classification of the process.

PT_CUR_CAT is the security category set of the process.
PT_KS_ASTE# keeps the aste# of the process's kernel stack.

PT_PS_ASTE# keeps the aste# of the process's segment which contains more information about the process.

PT_FLAGS indicates the execution state of a process. Its value is READY, BLOCKED or INACTIVE.

PT_LINK is used to chain together processes that are blocked on the same semaphore.

PT_IPC_QUOTA_HEAD is the beginning of a chain of interprocess communication messages sent to the process. Its value indicates one of three possible states: (1) there are messages that have been sent and not yet read by the process; (2) there are no messages that have been sent to the process and not yet read by the process; and (3) the process has been blocked because it wants to read another message and none is available.

PT_IPC_QUOTA is a number of interprocess communication objects currently available to the user for receiving messages from other processes.

3.3.1.3.1 Functions Using the Process Table

STARTP P
STOPP V
IPCRCV SLEEP
IPCESEND RUN
SOADD SWAP

3.3.1.4 Process Segment (PS)

There is a Process Segment (main memory segment) for each process. The PS is created at initialization time and along with the appropriate PT entry, holds information on the state of the process. The Process Segment is located in KSR1 address space (refer to Section 10, pages 14 and 15).

Format of a Process Segment

Process Segments are accessed by process#: PS_XXX(process#)

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NO. of BITS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS_CURRENT PROCESS</td>
<td>8</td>
<td>process#</td>
</tr>
<tr>
<td>PS_PROCESS_MASK</td>
<td>16</td>
<td>bit mask</td>
</tr>
<tr>
<td>Symbol</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PS_PROCESS_NOTMASK</td>
<td>16</td>
<td>bit mask</td>
</tr>
<tr>
<td>PS_USER_ID</td>
<td>14</td>
<td>user identification</td>
</tr>
<tr>
<td>PS_PROJECT_ID</td>
<td>8</td>
<td>project identification</td>
</tr>
<tr>
<td>PS_CUR_CLASS</td>
<td>4</td>
<td>security classification</td>
</tr>
<tr>
<td>PS_CUR_CAT</td>
<td>16</td>
<td>security category</td>
</tr>
<tr>
<td>PS_MEM_QUOTA</td>
<td>8</td>
<td>unused main memory quota</td>
</tr>
<tr>
<td>PS_SDR</td>
<td>16 x 16 array</td>
<td>save area for user and supervisor domain segmentation registers</td>
</tr>
<tr>
<td>PS_SAR</td>
<td>16 x 16 array</td>
<td>save area for user and supervisor domain segmentation registers</td>
</tr>
<tr>
<td>PS_SEG</td>
<td>15 x 32 array</td>
<td>definition of process's address space</td>
</tr>
</tbody>
</table>

**PS_CURRENT_PROCESS** is the number of the process associated with the PS.

**PS_PROCESS_MASK** and **PS_PROCESS_NOTMASK** are used in accessing the ACL_CPL and ACL_WAL. They are the process# expressed by a 16-bit field; the value of **PS_PROCESS_MASK** is expressed as 2^15-process#, whereas the value of **PS_PROCESS_NOTMASK** is the complement of **PS_PROCESS_MASK**. MASK is all zero except for the bit indicating the process#. NOTMASK is all ones except for the bit indicating the process#.

**PS_USER_ID** and **PS_PROJECT_ID** identify the user associated with the process.

**PS_CUR_CLASS** and **PS_CUR_CAT** define the classification and category which is the current security level of the process.

**PS_MEM_QUOTA** is the amount of main memory allocated to the process for its AS but not currently in use.

**PS_SDR** and **PS_SAR** are two arrays that form the save area for the 8 supervisor (0-7) and 8 user (7-15) segmentation registers.

**PS_SEG** is used for mapping segment numbers (seg#) into aste#'s. Every segment in a users WS has an aste#, but the aste# cannot be available to the user because it is a function of system wide activity. Therefore, when a process has the kernel move
a segment into its WS, the kernel returns a seg# which the process subsequently uses to identify the segment.

### 3.3.1.4.1 Function Using the Process Segment

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCHECK</td>
<td>GETW</td>
</tr>
<tr>
<td>OUTER</td>
<td>GETR</td>
</tr>
<tr>
<td>OUTERV</td>
<td>ENABLE</td>
</tr>
<tr>
<td>STARTP</td>
<td>WRITEDIR</td>
</tr>
<tr>
<td>STopp</td>
<td>SOADD</td>
</tr>
<tr>
<td>CHANGEO</td>
<td>CONNECT</td>
</tr>
<tr>
<td>INITH</td>
<td>DCOnNECT</td>
</tr>
<tr>
<td>IPCRCV</td>
<td>DSEARCH</td>
</tr>
<tr>
<td>IPCSEND</td>
<td>DISABLE</td>
</tr>
<tr>
<td></td>
<td>SWAP</td>
</tr>
</tbody>
</table>

### 3.3.1.5 Memory Block Table (MBT)

The Memory Block Table is a structure used to indicate the state of main memory. Contiguous blocks (256 bytes per block) can be concatenated to form main memory segments of any multiple block size. There is an entry in the MBT for each block with segments represented by several concatenated entries. The Memory Block Table is located in KSR0 address space (refer to Section 10, page 11).

#### Format of the Memory Block Table

The MBT is accessed by block#: MBT Xxx(block#)

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NO. of BITS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBT_FLAGS</td>
<td>2</td>
<td>FREE, ALLOCATED, or CONCATENATED</td>
</tr>
<tr>
<td>MBT_SIZE</td>
<td>8</td>
<td>size of the area in blocks</td>
</tr>
<tr>
<td>MBT_CHAIN</td>
<td>14</td>
<td>chain of free blocks</td>
</tr>
<tr>
<td>MBT_ASTE#</td>
<td>8</td>
<td>astext of the virtual memory segment in the block</td>
</tr>
</tbody>
</table>

If a block is the first in a segment, MBT_FLAGS for that block is either FREE or ALLOCATED; otherwise it is CONCATENATED. The remaining fields are not meaningful for CONCATENATED blocks.

MBT_SIZE is the number of blocks in a segment.
If a block is FREE, MBT_CHAIN is the block# of the next segment in the free chain or the initial address of high core if this is the end of the chain. (A block# is the address of the first byte in a block with the 8 low order \( \emptyset \) bits removed.)

If the block is ALLOCATED, MBT_ASTE is the aste# of the segment bound to it.

3.3.1.5.1 Functions Using the Memory Block Table

DISABLE
SWAPIN
SWAPOUT
RUN

3.3.1.6 Bit Maps Table (BMT)

Bit maps are used exclusively by the disk allocation functions. There is an allocated area on the disk for each of the three segment sizes (the initial implementation of the Security Kernel uses only size 2), a bit map for each area, and a Bit Map Table for each bit map. The Bit Map Table is a table of tables and is located in KSR\( \emptyset \) address space (refer to Section 10, page 13).

Format of the Bit Map Table

The SUE language functions pass the virtual address of the Bit Map Table associated with the size to the PAL-ll routines which will access the corresponding bit map.

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NO. of BITS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMT_SIZE1</td>
<td>64</td>
<td>segment SIZE1 bit map table</td>
</tr>
<tr>
<td>BMT_SIZE2</td>
<td>64</td>
<td>segment SIZE2 bit map table</td>
</tr>
<tr>
<td>BMT_SIZE3</td>
<td>64</td>
<td>segment SIZE3 bit map table</td>
</tr>
<tr>
<td>BIT_MAP2</td>
<td>512</td>
<td>one bit per 1K byte segment</td>
</tr>
<tr>
<td>BIT_MAP1</td>
<td>32</td>
<td>reserved for future use</td>
</tr>
<tr>
<td>BIT_MAP3</td>
<td>32</td>
<td>reserved for future use</td>
</tr>
</tbody>
</table>

BMT_SIZEn is the segment size n Bit Map Table. The Bit Map Table contains the start address (first 16 bits) and end address (next 16 bits) of the bit map, the base address of the disk area (next 16 bits) and a shift register (last 16 bits).
BIT_MAP2, in the initial implementation, is a map of 512 bits. The entire disk is allocated to segment SIZE2, that is, 512 1K byte segments. Each bit in the bit map corresponds to a segment on the disk. When a segment is allocated space on the disk, the corresponding bit in the bit map is set. When the space is freed the bit is cleared.

In future implementations the disk will be separated into three areas, one area for each of the three different sized segments.

3.3.1.6.1 Functions Using the Bit Map Table

DALLOC
DFREE

3.3.1.7 Hash Table

The Hash Table has only one field which is the disk address of a specific process. The Hash Table is located in KSR0 address space (refer to Section 10, page 11).

Format of Hash Table

A Hash Table entry is accessed by hash value: HASH_TABLE(HASH_VALUE)

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NO. of BITS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>HASH_TABLE</td>
<td>16</td>
<td>pointer into the AST_CHAIN</td>
</tr>
</tbody>
</table>

HASH_TABLE could be thought of as AST_CHAIN HEAD as it is actually the first non-free element in the AST_CHAIN.

3.3.1.7.1 Functions Using the Hash Table

HASH
ACT
DEACT

3.3.1.8 Interprocess Communication (IPC) Element Pool

The IPC Element Pool chains messages waiting to be received. The pool is a shared resource of 127 elements controlled by a quota mechanism (each receiving process is restricted to 8 message elements). The IPC Element Pool is located in KSR0 address space (refer to Section 10, page 11).
Format of an IPC Element Pool

An IPC Element is accessed by index:  IPC_XXX(INDEX)

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NO. of BITS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPC_LINK</td>
<td>8</td>
<td>chained IPC entry #</td>
</tr>
<tr>
<td>IPC_PROCESS#</td>
<td>8</td>
<td>sending process # and domain indicator</td>
</tr>
<tr>
<td>IPC_DATA</td>
<td>16</td>
<td>message</td>
</tr>
</tbody>
</table>

IPC_LINK is the chained IPC entry #, that is, the number of the next oldest element in the receiving process's chain of waiting elements.

IPC_PROCESS# contains the number of the sending process and a 1-bit domain indicator.

IPC_DATA contains the message being sent.

3.3.1.8.1 Functions Using the IPC Element Pool

STOPP
IPCRCV
IPCSSEND

3.3.1.9 Semaphores

Semaphore entries are arrays of 0 to 257. The first 255 semaphores are associated with active segment, that is, SMFR# = ASTE#. The kernel semaphore equals 256 and the disk semaphore equals 257. Semaphores are located in KSRØ address space (refer to Section 10, page 13).

Format of Semaphores

A semaphore entry is accessed by aste#:  SMFR_XXX(aste#)

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NO. of BITS</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMFR_COUNT</td>
<td>16</td>
<td>P decrements the count V increments the count</td>
</tr>
<tr>
<td>SMFR_POINTER</td>
<td>16</td>
<td>points to chains of blocked processes</td>
</tr>
</tbody>
</table>
SMFR_COUNT is decremented when a P is performed; when a V is performed it is incremented. The boundaries of SMFR_COUNT are -128 to 127.

SMFR_POINTER is an entry number into the PT_LINK which is the head of the chain of blocked processes (1 chain for each SMFR #).

3.3.1.9.1 Functions Using Semaphores

OUTERP
OUTERV
DCONNECT
ACT
P
V

3.3.1.10 Parameters

Parameters are a special case as they are actually part of the supervisor's data base. The kernel does, however, access this information when needed.

User input parameters are passed to the Security Kernel by placing them in fixed locations on the supervisor's stack. The Security Kernel accesses the supervisor's stack through kernel segmentation register 3. Only those parameters required by the requested function are entered by the user and accessed by the Security Kernel. Parameters are located in KSR3 address space (refer to Section 10, page 15).

Format of Parameter Entries

<table>
<thead>
<tr>
<th>FIELD</th>
<th>NO. of BITS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION_CODE_APARM</td>
<td>16</td>
<td>identifies a requested function</td>
</tr>
<tr>
<td>SEG#_APARM</td>
<td>16</td>
<td>identifies as active segment</td>
</tr>
<tr>
<td>OFFSET_APARM</td>
<td>16</td>
<td>identifies a directory entry</td>
</tr>
<tr>
<td>CLASS_APARM</td>
<td>16</td>
<td>classification</td>
</tr>
<tr>
<td>CAT_APARM</td>
<td>16</td>
<td>category set</td>
</tr>
<tr>
<td>SEG_TYPE_APARM</td>
<td>16</td>
<td>DATA or DIRECTORY</td>
</tr>
<tr>
<td>SIZE_APARM</td>
<td>16</td>
<td>size of a segment in blocks</td>
</tr>
<tr>
<td>MODE_APARM</td>
<td>16</td>
<td>WRITE, READ or NO access</td>
</tr>
<tr>
<td>USER_APARM</td>
<td>16</td>
<td>user_id</td>
</tr>
</tbody>
</table>
FUNCTION_CODE_APARM is a numerical tag from 1 to 20 which identifies a function.

SEG#_APARM is the segment number of a segment in a process's address space (WS).

OFFSET_APARM is the identification of an entry within a directory.

CLASS_APARM is the classification part of the security attribute.

CAT_APARM is the category set which is the rest of the security attribute.

SEG_TYPE_APARM identifies the segment as either DATA or DIRECTORY.

SIZE_APARM is the size of the segment in 256 byte blocks.

MODE_APARM is the mode of access which is either WRITE, READ, or NO access.

USER_APARM and PROJECT_APARM are the user and project identification.

REG#_APARM is the identification of a segmentation register.

PROCESS#_APARM is a numerical identification of a process. For the IPCRCV function this field identifies the process# plus domain.

MESSAGE_APARM is an interprocess communication message.

KRC is a per process return code.

KRC2 is used by the IPCRCV function exclusively and is assigned the value of the IPC message.
3.3.1.10.1 Functions Using Parameters

PCHECK
READIR

3.3.1.11 Segmentation Registers

The kernel functions that access information contained in the segmentation registers are:

GATE
READIR
ENABLE
DISABLE
RUN
LSD
SWAP

3.3.1.11.1 Segment Address Registers (SAR)

The Segment Address Register is a base address register. It contains the base address of the segment in the form of a 12-bit Segment Address Field (SAF). Bits 15-12 of the SAR are not implemented. The SAF is interpreted in address calculations as a multiplier of 32, i.e., the lowest 6 bits are assumed to be 0. The bit stored in bit 0 of the SAF becomes bit 6 of the segment base address, bit 1 of the SAF, bit 7 of the segment base address, etc. Thus, bit 11 of the SAF becomes bit 17 of the segment base addressed.

```
 15     12    11      0
SEGMENT ADDRESS FIELD
```

SEGMENT ADDRESS REGISTER FORMAT

3.3.1.11.2 Segment Descriptor Registers (SDR)

The Segment Descriptor Register (SDR) contains segment length, access control, and written into fields.
The ACF is a 3-bit field (occupying bits 2-0 of the SDR) which describes the access rights to this specific segment. The access codes specify the manner in which a segment may be accessed and whether or not a given access should result in a trap or an abort of the current process. A memory reference that causes an abort is terminated immediately. That is, an aborted "read" reference does not obtain any data from the location and an aborted "write" reference does not change the data in the location. A reference that causes a trap is completed.

<table>
<thead>
<tr>
<th>AFC</th>
<th>KEY</th>
<th>DESCRIPTION</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>non-resident</td>
<td>abort all process</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>read-only and trap</td>
<td>trap on read abort any attempt to write on this segment</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>resident read only</td>
<td>abort any attempt to write on this segment</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
<td>illegal</td>
<td>reserved for future use</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>resident read/write and trap</td>
<td>memory management trap upon completion of a read or write</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
<td>resident read/write and trap when write</td>
<td>memory management trap upon completion of write</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
<td>resident read/write</td>
<td>read or write allowed – no trap or abort</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
<td>illegal</td>
<td>reserved for future use</td>
</tr>
</tbody>
</table>
3.3.11.2.2 Written Into (W)

The W bit (occupying bit 6) indicates whether the segment has been written into since it was swapped into main memory. A W bit of 1 is affirmative indicating that the user has modified the segment and that it must be saved in its current form. A W bit of 0 indicates that the segment has not been modified and that it need not be written onto disk to be saved. The W bit is automatically cleared when either the SDR or SAR of a segment is written into.

3.3.11.2.3 Segment Length Field (SLF)

The 7-bit SLF (occupying bits 14-8) specifies the authorized length of the segment in 32-word blocks. A segment consists of at least one and at most 128 blocks, and occupies contiguous core location.

3.3.11.2.4 Attention (A) and Expansion Direction (ED)

The A bit (bit 7) and ED bit (bit 3) are not currently referenced by the Security Kernel.

3.3.2 Constants and Macros

The following two tables list all constant and macros used by the Security Kernel. Table I lists the constants contained in CONTEXT NOFORN and CONTEXT KERNEL. A description of each constant and its value is included. Table II lists the macros contained in CONTEXT NOFORN, CONTEXT KERNEL and DATA GATE. The effect of each macro and its parameters is included. The listings of CONTEXT NOFORM, CONTEXT KERNEL and DATA GATE can be found in Section 10, pages 2 through 19.

Table I

List of Constants

Values of the following are in hexadecimal unless otherwise indicated.

<table>
<thead>
<tr>
<th>CONSTANT</th>
<th>VALUE</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEM_SIZE</td>
<td>51110</td>
<td>128K bytes = 512 256-byte blocks</td>
</tr>
<tr>
<td>MBT_FLAGS_MASK*</td>
<td>C000</td>
<td>memory block table</td>
</tr>
<tr>
<td>MBT_CHAIN_MASK</td>
<td>3FFF</td>
<td>FLAGS, CHAIN, and ASTE all share a word</td>
</tr>
</tbody>
</table>

*Mask is used as an "anding" operation for selecting a bit from a bit table.
| Table I (Continued) |
|---------------------|-----------------|
| END_BLOCK#         | 3E0             |
| ALLOCATED          | 0               |
| CONCATENATED       | 4000            |
| FREE_MEM           | 8000            |
| RESERVED_MEM       | C000            |
| ASTE#_MIN          | 110             |
| ASTE#_MAX          | 255\text{10}   |
| AST_TYPE_MASK      | 80              |
| AST_STATUS_MASK    | 40              |
| AST_STATUS_NOTMASK | BF              |
| AST_CHANGE_MASK    | 20              |
| AST_UNLOCK_MASK    | 10              |
| AST_LOCK_MASK      | EF              |
| AST_CLASS_MASK     | OF              |
| AST_TYPE_DIRECTORY | AST_TYPE_MASK   |
| AST_CHANGE         | AST_CHANGE_MASK |
| AST_UNCHANGED_MASK | DF              |
| AST_UNINITIALIZED  | AST_STATUS_MASK |
| AST_UNLOCK_FLAG    | AST_UNLOCK_MASK |
| WIRED_DOWN MASK    | 8000            |
| WIRED_DOWN NOTMASK | 7FFF            |
| WIRED_DOWN         | WIRED_DOWN_MASK |
| ROOT_ASTE#         | 110             |
| IPC_MAX            | 127\text{10}   |
| IPC_QUOTA          | 8\text{10}     |

- Setting of memory block table flag field
- Range of active segment table entries
- Active segment table entries TYPE, STATUS, CHANGE, UNLOCK, and CLASS share a byte
- Definition of active segment table TYPE entry
- Definition of active segment table change bit
- Definition of active segment table STATUS entry
- Definition of active segment table UNLOCK entry
- Bit 0 of the connected process list is wired down bit
- Active segment table entry ROOT constant
- Number of elements in the interprocess communications pool
- Receiving processes are restricted to 8 message elements
Table I (Continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMT_SIZE1_ADR</td>
<td>1B00</td>
<td>definition of bit map table SIZE entry address</td>
</tr>
<tr>
<td>BMT_SIZE2_ADR</td>
<td>1B08</td>
<td></td>
</tr>
<tr>
<td>BMT_SIZE3_ADR</td>
<td>1B10</td>
<td></td>
</tr>
<tr>
<td>KERNEL_SMFR</td>
<td>25610</td>
<td>definition of KERNEL and DISK semaphores</td>
</tr>
<tr>
<td>DISK_SMFR</td>
<td>25710</td>
<td></td>
</tr>
<tr>
<td>SMFR_MAX</td>
<td>DISK_SMFR</td>
<td></td>
</tr>
<tr>
<td>SEG#_FLAG</td>
<td>8000</td>
<td>definition of parameter flags</td>
</tr>
<tr>
<td>OFFSET_FLAG</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>CLASS_FLAG</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>REG_FLAG</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>PROCESS#_FLAG</td>
<td>0800</td>
<td></td>
</tr>
<tr>
<td>MODE_FLAG</td>
<td>0400</td>
<td></td>
</tr>
<tr>
<td>PT_KSDR1_ADR</td>
<td>1E00</td>
<td>segmentation descriptor and address registers must be separated by 2016</td>
</tr>
<tr>
<td>PT_KSDR2_ADR</td>
<td>1E40</td>
<td>process table entries FLAGS and LINK share a byte. LINK is only meaningful</td>
</tr>
<tr>
<td>PT_FLAGS_MASK</td>
<td>CO</td>
<td>when FLAGS = BLOCKED</td>
</tr>
<tr>
<td>PT_LINK_MASKS</td>
<td>3F</td>
<td></td>
</tr>
<tr>
<td>BLOCKED</td>
<td>00</td>
<td>definition of flag field</td>
</tr>
<tr>
<td>READY</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>INACTIVE</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>IPC_WAIT</td>
<td>FF</td>
<td>a process is waiting for a message</td>
</tr>
<tr>
<td>PS_KSR_ADR</td>
<td>F4C2</td>
<td>segmentation descriptor and address registers must be separated by 2016</td>
</tr>
<tr>
<td>PS_SDR_ADR</td>
<td>2000</td>
<td>changed to aste# when the segment is allocated</td>
</tr>
<tr>
<td>SEG_FLAG</td>
<td>8000</td>
<td>used to mask out SEG_FLAG</td>
</tr>
<tr>
<td>SEG_MASK</td>
<td>7FPE</td>
<td>processes are restricted to 9K words of memory except the executive which is</td>
</tr>
<tr>
<td>MEM_QUOTA</td>
<td>24</td>
<td>virtually unrestricted</td>
</tr>
<tr>
<td>EXEC_MEM_QUOTA</td>
<td>7F</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>STACK_KSR_ADR</td>
<td>F4C4</td>
<td>definition of the stack segmentation register</td>
</tr>
<tr>
<td>DIR_KSR_ADR</td>
<td>F4C6</td>
<td>definition of the directories segmentation register</td>
</tr>
<tr>
<td>ACL_MAX</td>
<td>127(_{10})</td>
<td>number of active control list elements to be shared</td>
</tr>
<tr>
<td>DIR_TYPE_MASK</td>
<td>80</td>
<td>directory entries</td>
</tr>
<tr>
<td>DIR_STATUS_MASK</td>
<td>40</td>
<td>TYPE, STATUS, and CLASS share a byte</td>
</tr>
<tr>
<td>DIR_STATUS_NOTMASK</td>
<td>BF</td>
<td></td>
</tr>
<tr>
<td>DIR_CLASS_MASK</td>
<td>OF</td>
<td></td>
</tr>
<tr>
<td>DIR_CLASS_NOTMASK</td>
<td>FO</td>
<td></td>
</tr>
<tr>
<td>DIR_TYPE_DIRECTORY</td>
<td>80</td>
<td>definition of directory entries TYPE and STATUS</td>
</tr>
<tr>
<td>DIR_UNINITIALIZED</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>ACL_MODE_MASK</td>
<td>C000</td>
<td>access control list entries MODE and USER share a word</td>
</tr>
<tr>
<td>ACL_USER_MASK</td>
<td>3FFF</td>
<td></td>
</tr>
<tr>
<td>REG_CONSTANT</td>
<td>578</td>
<td>definition of accessing supervisor and user segmentation registers supervisor SRO = 3F480 user SRO - 3FF80 REG_CONSTANT = ((3FF80 - 3F480)/2)-8=578(_{16})</td>
</tr>
<tr>
<td>P_REG#_MAX</td>
<td>587</td>
<td></td>
</tr>
<tr>
<td>CROSS_REG#</td>
<td>710</td>
<td></td>
</tr>
<tr>
<td>SDR_ADR</td>
<td>F480</td>
<td></td>
</tr>
<tr>
<td>SDR_WRITE_ACCESS</td>
<td>0006</td>
<td>definition of descriptor register fields</td>
</tr>
<tr>
<td>SDR_READ_ACCESS</td>
<td>0002</td>
<td></td>
</tr>
<tr>
<td>SDR_CHANGE_MASK</td>
<td>0040</td>
<td></td>
</tr>
<tr>
<td>SDR_CHANGED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PREV_MODE_MASK</td>
<td>3000</td>
<td>on kernel entry call is ignored if not made from supervisor mode</td>
</tr>
<tr>
<td>PREV_MODE_SUPERV</td>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>
Table I (Continued)

| DISK_WRITE | 0043 | disk commands |
| DISK_READ  | 0045 |

The following are hardware instructions. Their values are in octal unless otherwise indicated.

- **ADD** 0006: add instruction
- **ASL** 0063: arithmetic shift left
- **ASR** 0062: arithmetic shift right
- **BCC** 103(1)0*: branch on carry clear instruc.
- **BVS** 102(1)1: branch on overflow clear instruc.
- **CLR** 0050: clear instruction
- **DEC** 0053: decrement instruction
- **INC** 0052: increment instruction
- **JMP** 000001: jump instruction
- **MOV** 0001: move source instruction (word)
- **MOVB** 0011: move source instruction (bytes)
- **NEG** 0054: negate instruction
- **SUB** 0016: subtract source instruction
- **SWAB** 0003: swap bytes
- **TRAP** 104400: trap instruction
- **MUL** 070100: multiply instruction
  \[ R1 = R0 \times R1 \]
- **DIV** 071002: divide instruction
  \[ R0 = R0R1/R2 \]
- **ASHR1** 072127: shift arithmetically (Register 1)
- **ASHROR3** 072003: shift arithmetically (Register 0, Register 3)
- **ASHR1R3** 072103: shift arithmetically (Register 1, Register 3)
- **XORL0** 074100: Exclusive OR
- **MFFIR6** 006506: move from previous instruction space
- **MTPIR6** 006606: move to previous instruction space
- **SPLHIGH** 000237: set priority level high
- **SPLLOW** 000230: set priority level low

*A (1) indicates that the value following is in binary.*

202
Table I (Continued)

The following values are in decimal unless otherwise indicated

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_FUNCTION_CODE</td>
<td>1</td>
</tr>
<tr>
<td>DELETE_FUNCTION_CODE</td>
<td>2</td>
</tr>
<tr>
<td>GIVE_FUNCTION_CODE</td>
<td>3</td>
</tr>
<tr>
<td>RESCIND_FUNCTION_CODE</td>
<td>4</td>
</tr>
<tr>
<td>GETM_FUNCTION_CODE</td>
<td>5</td>
</tr>
<tr>
<td>GETR_FUNCTION_CODE</td>
<td>6</td>
</tr>
<tr>
<td>RELEASE_FUNCTION_CODE</td>
<td>7</td>
</tr>
<tr>
<td>ENABLE_FUNCTION_CODE</td>
<td>8</td>
</tr>
<tr>
<td>DISABLE_FUNCTION_CODE</td>
<td>9</td>
</tr>
<tr>
<td>P_FUNCTION_CODE</td>
<td>10</td>
</tr>
<tr>
<td>V_FUNCTION_CODE</td>
<td>11</td>
</tr>
<tr>
<td>T_FUNCTION_CODE</td>
<td>12</td>
</tr>
<tr>
<td>IPCSEND_FUNCTION_CODE</td>
<td>13</td>
</tr>
<tr>
<td>IPCRCV_FUNCTION_CODE</td>
<td>14</td>
</tr>
<tr>
<td>STARTP_FUNCTION_CODE</td>
<td>15</td>
</tr>
<tr>
<td>ST0PP_FUNCTION_CODE</td>
<td>16</td>
</tr>
<tr>
<td>CHANCE0_FUNCTION_CODE</td>
<td>17</td>
</tr>
<tr>
<td>PROCID_FUNCTION_CODE</td>
<td>18</td>
</tr>
<tr>
<td>INITH_FUNCTION_CODE</td>
<td>19</td>
</tr>
<tr>
<td>READIR_FUNCTION_CODE</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Code MIN</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION_CODE_MIN</td>
<td>1</td>
</tr>
<tr>
<td>FUNCTION_CODE_MAX</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Segment Number Min</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEG#_MIN</td>
<td>1</td>
</tr>
<tr>
<td>SEG#_MAX</td>
<td>31</td>
</tr>
<tr>
<td>ROOT_SEG#</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset Min</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFFSET_MIN</td>
<td>1</td>
</tr>
<tr>
<td>OFFSET_MAX</td>
<td>63</td>
</tr>
<tr>
<td>PDD_OFFSET</td>
<td>1</td>
</tr>
<tr>
<td>IOD_OFFSET</td>
<td>2</td>
</tr>
<tr>
<td>CD_OFFSET</td>
<td>3</td>
</tr>
<tr>
<td>FMS_OFFSET</td>
<td>4</td>
</tr>
<tr>
<td>UNCLASSIFIED</td>
<td>1</td>
</tr>
<tr>
<td>CONFIDENTIAL</td>
<td>2</td>
</tr>
<tr>
<td>SECRET</td>
<td>3</td>
</tr>
<tr>
<td>TOP_SECRET</td>
<td>4</td>
</tr>
<tr>
<td>CLASS_MIN</td>
<td>1</td>
</tr>
<tr>
<td>CLASS_MAX</td>
<td>4</td>
</tr>
</tbody>
</table>

203
Table I (Continued)

<table>
<thead>
<tr>
<th>SEG_TYPE_DIRECTORY</th>
<th>80_{16}</th>
<th>definition of segment type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEG_TYPE_DATA</td>
<td>00_{16}</td>
<td></td>
</tr>
<tr>
<td>SIZE1</td>
<td>1</td>
<td>256 bytes</td>
</tr>
<tr>
<td>SIZE2</td>
<td>4</td>
<td>1K bytes</td>
</tr>
<tr>
<td>SIZE3</td>
<td>16</td>
<td>4K bytes</td>
</tr>
<tr>
<td>NO_ACCESS</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>READ$EXECUTE_ACCESS</td>
<td>4000_{16}</td>
<td></td>
</tr>
<tr>
<td>WRITE$READ$EXECUTE_ACCESS</td>
<td>C000_{16}</td>
<td></td>
</tr>
<tr>
<td>ALL_USERS</td>
<td>3FFF</td>
<td>definition of user and project</td>
</tr>
<tr>
<td>ALL_PROJECTS</td>
<td>7F_{16}</td>
<td></td>
</tr>
<tr>
<td>SYSTEM_PROJECT</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>REG# MIN</td>
<td>0</td>
<td>range of register numbers</td>
</tr>
<tr>
<td>REG# MAX</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>PROCESS# MIN</td>
<td>1</td>
<td>definition of process numbers</td>
</tr>
<tr>
<td>PROCESS# MAX</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>PROCESS# 2MAX</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>EXEC_PROCESS#</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>TTY_PROCESS#</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>DECW_PROCESS#</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>SCOPE1_PROCESS#</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>SCOPE2_PROCESS#</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>USER_PROCESS# MIN</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>USER_PROCESS# MAX</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>PROCESS# MASK</td>
<td>7F_{16}</td>
<td>definition of IPCRCV PROCESS# (PROCESS# plus DOMAIN)</td>
</tr>
<tr>
<td>DOMAIN MASK</td>
<td>80_{16}</td>
<td></td>
</tr>
<tr>
<td>KERNEL_DOMAIN</td>
<td>DOMAIN_MASK</td>
<td></td>
</tr>
<tr>
<td>OF_FLAG</td>
<td>FFFF_{16}</td>
<td>definition of kernel return code</td>
</tr>
<tr>
<td>ERR_FLAG</td>
<td>FFFE_{16}</td>
<td></td>
</tr>
<tr>
<td>SEVERE_FLAG</td>
<td>FFFF_{16}</td>
<td></td>
</tr>
<tr>
<td>TRUE</td>
<td>1_{10}</td>
<td></td>
</tr>
<tr>
<td>FALSE</td>
<td>0_{10}</td>
<td></td>
</tr>
<tr>
<td>MAXIMUM_INTEGER</td>
<td>32767_{10}</td>
<td>character varying maximum length</td>
</tr>
<tr>
<td>MAX_NEG_INTEGER</td>
<td>-32768_{10}</td>
<td></td>
</tr>
<tr>
<td>CV_MAX_LEN</td>
<td>72_{10}</td>
<td></td>
</tr>
<tr>
<td>CARRIAGE_RETURN</td>
<td>D_{16}</td>
<td></td>
</tr>
<tr>
<td>LINE_FEED</td>
<td>25_{16}</td>
<td></td>
</tr>
</tbody>
</table>
Table I (Concluded)

<table>
<thead>
<tr>
<th>LOW_CHARACTER</th>
<th>HIGH_CHARACTER</th>
<th>NEW_LINE</th>
<th>END_OF_FILE</th>
<th>BS_CHAR</th>
<th>CANCEL_CHAR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>@</td>
<td>LINE_FEED</td>
</tr>
</tbody>
</table>

Table II

List of Macros

<table>
<thead>
<tr>
<th>MACRO_NAME</th>
<th>PARAMETERS</th>
<th>EFFECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTIPLY</td>
<td>OP1, OP2, PRODUCT, FLAG</td>
<td>Places the product of OP1 and OP2 in PRODUCT and sets FLAG if result is less than $-2^{15}$ or greater than or equal to $2^{15}$.</td>
</tr>
<tr>
<td>DIVIDE</td>
<td>DIVIDEND, DIVISOR, QUOTIENT, FLAG</td>
<td>Places the result of DIVIDEND/DIVISOR in quotient and sets FLAG if dividing by zero is attempted.</td>
</tr>
<tr>
<td>MODULO</td>
<td>DIVIDEND, DIVISOR, REMAINDER, FLAG</td>
<td>Finds the REMAINDER and sets FLAG in the event of an overflow.</td>
</tr>
<tr>
<td>KCREATE</td>
<td>SEG#, OFFSET, CLASS, CAT, SEG_TYPE_SIZE, RC</td>
<td>Calls kernel function CREATE and sets RC to KERNEL_RC.</td>
</tr>
<tr>
<td>KDELETE</td>
<td>SEG#, OFFSET, RC</td>
<td>Calls kernel function DELETE and sets RC.</td>
</tr>
<tr>
<td>Command</td>
<td>Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>KGIVE</td>
<td>SEG#, OFFSET, USER, USER, PROJECT, RC</td>
<td>Calls kernel function GIVE and sets RC.</td>
</tr>
<tr>
<td>KRESCIND</td>
<td>SEG#, OFFSET, USER, USER, PROJECT, RC</td>
<td>Calls kernel function RESCIND and sets RC.</td>
</tr>
<tr>
<td>KGETW</td>
<td>SEG#, OFFSET, RC</td>
<td>Calls kernel function GETW and sets RC.</td>
</tr>
<tr>
<td>KRELEASE</td>
<td>SEG#</td>
<td>Calls kernel function DCONNECT</td>
</tr>
<tr>
<td>KENABLE</td>
<td>SEG#, REG#, RC</td>
<td>Calls kernel function ENABLE and sets RC.</td>
</tr>
<tr>
<td>KDISABLE</td>
<td>REG#</td>
<td>Calls kernel function DISABLE and sets RC.</td>
</tr>
<tr>
<td>KP</td>
<td>SEG#, RC</td>
<td>Calls kernel function OUTERP and sets RC.</td>
</tr>
<tr>
<td>KV</td>
<td>SEG#, RC</td>
<td>Calls kernel function OUTERV and sets RC.</td>
</tr>
<tr>
<td>KT</td>
<td>SEG#, RC</td>
<td>Enters kernel to read semaphore and sets RC.</td>
</tr>
<tr>
<td>KIPSEND</td>
<td>PROCESS#, MESSAGE</td>
<td>Calls kernel function IPCSEND.</td>
</tr>
<tr>
<td>KIPRCV</td>
<td>PROCESS#, MESSAGE</td>
<td>Calls kernel function IPCRCV to read MESSAGE and sending process#.</td>
</tr>
<tr>
<td>KSTARTP</td>
<td>USER_ID, PROJECT_ID, CLASS, CAT, PROCESS#, PROC_OFFSET, RC</td>
<td>Calls kernel function STARTP and sets RC.</td>
</tr>
<tr>
<td>KSTOPP</td>
<td></td>
<td>Calls kernel function STOPP.</td>
</tr>
<tr>
<td>KCHANGEO</td>
<td>SEG#, OFFSET, CLASS, CAT, RC</td>
<td>Calls kernel function CHANGEO and sets RC.</td>
</tr>
</tbody>
</table>
3.4 Security Kernel Function Call Matrix

The Function Call Matrix presented in Figure 6 lists the forty-two functions of the Security Kernel. Reading across the matrix shows which functions are called by a specific function. Reading down shows which functions a specific function calls. The matrix also shows whether a specific function is externally or internally callable, the number of functions it calls, and the number of functions that call it. Eighteen of the functions are externally callable. The remaining twenty-four are called either directly or indirectly by the externally callable functions. The non-callable functions are invisible outside the kernel domain and deal basically with the management of the PDP-11/45's physical resources. The externally callable functions may be invoked by any process operating in supervisor domain with the exception of STARTP, CHANGEO, and INITH which may be called by one trustworthy process, the Executive Process.
Figure 6 KERNEL FUNCTION CALL MATRIX
3.4.1 **Program Interrupts**

Program interrupts occur when a new process requests the service of the CPU. Before servicing the new process, however, the CPU finishes executing the instruction it is working on. The interrupt is then handled by the invocation of the KERNEL_ENTRY macro, which causes the contents of the current process's registers to be saved. The PC and PSW of the interrupt vector now become the new process's PC and PSW. The function V is then called to increment the count on the semaphore associated with the new process's I/O segment. Macro KERNEL_EXIT is then invoked which causes the general purpose registers, the PC and the PSW to be restored with what they contained before the interrupt.

3.4.2 **Subprogram Referencing**

The following figures depict the calling flow of the Security Kernel. To facilitate readability the Security Kernel has been broken down into three levels; the entry point GATE (Figure 7), the eighteen externally callable functions (Figure 8 through 21), and the internal function SWAPIN (Figure 22).
Figure 7. GATE Call Diagram
Figure 8. CREATE Call Diagram

See Figure 22
Figure 9. DELETE Call Diagram
Figure 10. GIVE and RESCIND Call Diagram
Figure 11. GETW and GETR Call Diagram
Figure 12. DCONNECT Call Diagram

Figure 13. ENABLE Call Diagram
Figure 14. OUTERP Call Diagram

Figure 15. OUTERV Call Diagram
Figure 16. IPCRCV Call Diagram

Figure 17. STARTP Call Diagram
Figure 18. STOPP Call Diagram

Figure 19. CHANGEO Call Diagram
INITH

WRITEDIR

SWAPIN        LSD

See Figure 22

Figure 20. INITH Call Diagram

READIR

SWAPIN        LSD

See Figure 22

Figure 21. READIR Call Diagram
Figure 22. SWAPIN Call Diagram
3.4.3 Special Control Features

This paragraph briefly describes three system programs which may be run in conjunction with the Security Kernel.

3.4.3.1 STARTUP

STARTUP sets up a basic environment in which to run the Security Kernel. It reads the Kernel, Executive, and Listener code from magnetic tape into core. It then initializes the root directory, activates the Executive's working space, and invokes the Executive.

3.4.3.2 EXECUTIVE

EXECUTIVE creates basic directories (the process directory directory, the I/O directory and the code directory) subordinate to the root. It places code segments into the code directory, I/O segments into the I/O directory, and creates its own process directory off the process directory directory. It then establishes a Listener program for each connected terminal. Two subprograms of the Executive, PSTART and PSTOP, are called to initiate and terminate Listener and user processes.

3.4.3.3 LISTENER

LISTENER accepts a user's correct start command, then loads the user, project, class and category into a message segment to be read by the Executive, which will start a user process with the specified security characteristics. There is a Listener process for each user station.
4. QUALITY ASSURANCE

4.1 Validation Criteria

Department of Defense Regulation DoD 5200.1-R (ref., paragraph 2.1b this specification) governs the Classification, Downgrading, Declassification, and Safeguarding of Classified Information pursuant to DoD Directive 5200.1 (ref., paragraph 2.1a), the Department of Defense Information Security Program. This program and Regulation addresses the problem of protection of official information relating to National Security, to the extent and for such period as is necessary. The Regulation establishes the bases for identification of information to be protected; establishes a progressive system for classification, downgrading and declassification; prescribes safeguarding policies and procedures to be followed; and establishes a monitoring system to insure the effectiveness of the Information Security Program throughout the Department of Defense.

DoD 5200.1-R provides the following definition of information: "knowledge that can be communicated in any form". It also provides the following policy with respect to certain official information: "To protect against actions hostile to the United States,... it is essential that such official information... be given only limited dissemination". To implement this policy, it states that such information be designated as needing protection, i.e., that it be classified. To further aid in implementing this policy the regulation states that "the dissemination of classified information orally, in writing, or by other means, shall be limited to those persons whose official duties require knowledge or possession (need-to-know) thereof" and, more specifically, no person shall be eligible for access to classified information unless a determination has been made as to his trustworthiness, i.e., unless he has been given the requisite level of security clearance.

4.1.1 Information Security Model

In order to implement a computer system providing the requisite security of official information from any possibility of compromise, it is necessary that that system behave in the machine domain in precise and complete correspondence with the regulations and intent of the DoD Information Security Program. The concepts of regulation DoD 5200.1-R (of people, information, and limiting access to information), provide the basis for representing the DoD Information Security Program in the form of an Information Security Model. This model will be validated, by the approving authority, to be a precise and sufficient algorithmic statement of the functions corresponding to the requirements and definitions of DoD 5200.1-R. Upon validation,
this model shall be the controlling criterion against which the acceptability of the Security Kernel Computer Program Product (described in Section 3 of this specification) will be measured for validation.

4.1.1.1 Elements of the Information Security Model

The Information Security Model, which is a precise algorithmic statement of security functions, consists of four elements: subjects, objects, an access control mechanism, and an authorization data base. The model describes the security requirements to be satisfied by subjects (people or processes) for accessing objects, in any specifically identified mode, under control of the Security Kernel. In the meaning of this paragraph, objects can be files, messages, buffers, terminals, I/O devices, etc. Objects can be accessed by subjects only in accordance with the compromise prevention requirements stipulated by the access control mechanism of the Security Kernel.

4.1.2 Validation Tests and Demonstrations

A specific program of demonstrations and tests shall be performed to verify that the functionality of the Security Kernel Computer Program Product (SKCPP) precisely and completely corresponds with the concepts of the Information Security Model, and also that no functionality of the SKCPP fails to correspond precisely and completely with one or more concepts of the model. These demonstrations and tests will take the form of rigorous, logically sound proofs of correspondence, and may be performed in sequential steps of validation which form a step-by-step validation correspondence proof chain stretching between the executable machine code (the least abstract representation of the Security Kernel) and the Information Security Model (the most abstract representation of the Security Kernel). Such a validation chain is discussed in paragraph 4.1.2.1 below.

4.1.2.1 The Validation (Correspondence Proof) Chain

The process of validation of the Security Kernel has as its goal the clear and rigorous proof that the conceptual solution of the real-world problem of prevention of compromise of information security, as represented by the Information Security Model, has been precisely implemented on the particular hardware/software system that will deal with that real-world problem.

Specifically, it is required that the functionality of the hardware/software system that consists of the binary language representation of the Security Kernel, correctly installed and operating in a
DEC PDP-11/45 computer with Memory Management Unit, be rigorously proved to completely and exclusively correspond to the functionality described by the Information Security Model.

The said validation goal requires that all aspects of the proofs be thoroughly rigorous and that they be clearly documentable. Unfortunately, the formal language and semantics in which the SKCPP is expressed are not directly comparable with the logical structure of the Information Security Model. This fact would make direct correspondence between these two representations impracticable to prove and document. Instead, a multi-step, continuous chain of correspondence proofs, similar in form to that illustrated in Figure 23, shall be performed. In this generalized validation chain design begins with the most abstract representation (the Information Security Model) and proceeds in steps through more concrete representations until it reaches the most concrete form, the useable system (hardware/software binary machine language) representation of the Security Kernel. At each link in the chain, it is required that any and all state transformations that are possible in the less abstract (more concrete) representation be proved to correspond exactly and completely with expected state transformations in the more abstract representation.

Figure 23. The Validation Chain
4.1.2.2 Validation Chain Components

Figure 23 illustrates four kinds of representations, at differing levels of abstraction, that may be used to implement validating the proof of correspondence between the useable hardware/software binary "machine language" representation of the Security Kernel and the "mathematical model" of computer security of information. The four representations, leading from the most abstract to the most concrete, are:

a. mathematical model,

b. formal specification,

c. algorithmic representation, and

d. useable system.

As used here, the "mathematical model" refers to the model described in reference 2.2a; the "formal specification" refers to the "Parnas specification" for each computer program component which appears as paragraphs 3.2.n.1 in Section 3 of this specification; and the "algorithmic representation" is the SKCPP SUE language and PAL-ll representations described in Section 3 of this specification.

The illustration implies that correspondence proofs will follow the path through the identical representations to those used in design to show that less abstract corresponds with the more abstract representation. However, the actual proofs of correspondence for validation need not follow the identical path through the representations that were used in development of the SKCPP, provided that the correspondence proofs constitute a rigorously continuous chain of proofs that prove the correspondence between the useable machine language Security Kernel and the Information Security Model.

The correspondence proof chain may, for example, follow a chain of proofs such as the following:

a. Useable hardware/software binary machine language representation.

b. PAL-ll assembler language representation.

c. SUE language representation.

d. PARNAS formal specification.

e. Information Security Model.
5. PREPARATION FOR DELIVERY

This section states the requirements incumbent upon the Contractor for preparing the Security Kernel Computer Program Product (SKCPP) for delivery to the Government and for insuring the integrity and security of the product as delivered for validation and final delivery.

5.1 Preparation of Useable Machine Language SKCPP

Prior to the start of validation tests and demonstrations, the Contractor shall take all the actions necessary to prepare, produce and protect the integrity of a precise binary machine language representation of the SKCPP described in the SUE and the PAL-11 languages in Section 3 of this specification. The Contractor shall use the (combined SUE and PAL-11) high-order language SKCPP representation of this specification as the source level code, in a security controlled IBM 360 environment where the Project SUE system and the PDP-11 cross assembler both execute, to compile the SKCPP machine language load module on 9-track magnetic tape media for transfer to the PDP-11/45. The Contractor shall protect the integrity of this preparation process and the resulting machine language code media as required by paragraph 5.1.1 below.

5.1.1 Protection of SKCPP Integrity

The Contractor shall take all the action necessary to insure, protect, and preserve the accuracy and integrity of the SKCPP binary machine language load module for the PDP-11/45. These actions shall include but not be limited to:

a. Protection of the integrity of the source level code through supervisory control and monitoring by a specific hierarchical group of persons, referred to hereinafter as the Kernel Control Group (KCG), selected by the Contractor and the Government and approved by the authority designated by the Government agency responsible for the SKCPP procurement. The Contractor shall provide the KCG with free access to monitor and review the correctness of the source level code and all the processes employed in compiling the SKCPP binary machine language load module in the tape media. The KCG shall have complete configuration control for the SKCPP. The Contractor shall be responsible for submitting to the KCG, in advance of implementation, any contemplated modification whatsoever to the SKCPP source level code, the compilation environment and procedures or the contents of
the load module. No changes shall be made to any of these entities without prior approval by the KCG.

b. Protection of the physical media, in which the binary machine language version of the Security Kernel resides, from any possibility of unauthorized alteration; this protection shall be commensurate with the level of protection required by the highest level of information security classification and special access categories for which the system, in which the said Security Kernel will be installed, must be cleared.

c. Those actions necessary to allow disclosing the contents of and information describing the SKCPP binary machine language load module as if it were unclassified, while protecting its security against modification to the extent required by (b) above.

d. The Contractor shall maintain, at all times subsequent to its initial compilation, a duplicate copy of the machine language version of the SKCPP, and shall also maintain an up-to-date history of any and all modifications that occur to the original copy of the SKCPP.

e. Whenever modification occurs to the binary machine language SKCPP, the duplicate copy shall be modified to maintain its identity with the original copy, and the SKCPP must immediately be purged from the computer until its revalidation has been completed and approved.

f. Revalidation after modification of the SKCPP binary machine language media shall be performed and documented following procedures meeting the requirements of Sections 4.1.2, 4.1.2.1, and 4.1.2.2 of this specification.
6. NOTES

The following notes are provided informally to assist the potential Security Kernel user.

NOTES ON THE MITRE PDP-11/45 PROTOTYPE
SECURITY KERNEL

The PDP-11/45 kernel distribution consists of two 9 track magnetic tapes and documentation. The tapes are referenced as the PDP-11 tape and the OS/360 tape. These notes document the contents of the two tapes and will serve as a guide to additional documentation:


b. Memo, "Project 7070 versus IBM OS/TSO and the Project SUE System Language", by J. A. Larkins, August, 1975. This memo was originally intended for use by MITRE Project 7070 personnel, so that some of the information it contains is relevant only to the MITRE IBM system.

c. Notes on "Using the Kernel Exerciser".

d. Notes on "Using the ALTER Program".

e. PDP-11/45 Configuration Chart.

f. PDPTAPE1 job output and tape dump.

g. OSTAPE2 job output.

h. SUE compilation listings for
   - STARTUP
   - KERNEL
   - EXEC
   - LISTENER
   - EXERCISE
i. PAL assembly listings for
   - VDUMP
   - BOOT
   - DALLOC
   - DFREE
   - LSD
   - DISKIO
   - SWAP

For information on the SUE Compiler itself, contact Dr. R. C. Holt, Computer Systems Research Group, University of Toronto.

PDP-11 TAPE

The PDP-11 tape can be bootloaded onto a PDP-11 by the firmware Bootstrap Loader (MR11-DB). It contains object code of the Security Kernel, a program which initializes the Kernel, programs that run in conjunction with the Kernel, and a simple test program. The tape distributed has been successfully loaded onto MITRE's PDP-11/45, but it is configuration dependent. Physically, the tape consists of eight records followed by a file mark. The following paragraphs describe the contents of each record.

Record 1

Record 1 is a short record of all zeros. It is only on the tape because MITRE's bootstrap loader skips the first record and loads the second.

Record 2

Record 2 is BOOT, a short program written in PAL-11. BOOT is loaded by the bootstrap loader starting at location 0 and then control is passed to it (at 0). BOOT loads record 3 starting at location 8000 and then passes control to it. Since record 3 is a SUE program, BOOT like most other programs on the tape is cognizant of the SUE runtime environment and initializes general purpose registers 4 and 6 appropriately.

Record 3

Record 3 contains STARTUP. STARTUP does the initialization that is necessary before the first Security Kernel function can be invoked - it puts the system into Z₀, the initial secure state. STARTUP initializes the Security Kernel's data bases, reads in the rest of the
tape, and transforms itself into the executive process. STARTUP
does a few things that may be unnecessarily complex. The main
memory in which STARTUP runs is allocated to the Process Segments,
to the two executive stacks and to the ROOT directory. The stacks
are not accessed until STARTUP is finished, but it must initialize
the Process Segments and the ROOT while it is running. The final
transition into the executive process is also a little complex.

Record 4

Record 4 contains VDUMP, a simple, stand-alone debugging pro-
gram written in PAL-11. It uses hexadecimal notation and displays
main memory locations when started. Since it runs with the MMU, it
can access any area in core, given the corresponding descriptors.
VDUMP is invoked by manually branching to location \texttt{3400} (hexadecimal).

Record 5

Record 5 contains the Security Kernel.

Record 6

Record 6 contains EXEC, the code that the executive process
executes. EXEC runs on the Security Kernel but the executive pro-
cess has special privileges - it is the root process and is the
only process that can create new processes. It is also the only
trusted subject in the system.

EXEC has two phases - a one-time-only phase and a steady-state
phase. In the one-time-only phase it establishes the initial
hierarchy by creating some directories and putting code and I/O seg-
ments into them. In the steady-state EXEC responds to user logon
and logoff requests by starting processes that run the EXERCISER or
the LISTENER.

Record 7

Record 7 contains the LISTENER, a program that the executive
runs in a process for each free terminal. The LISTENER responds to
user logon requests at its process's terminal. If the request is
valid the LISTENER destroys itself, an event that is detectable by
the executive. The response of the executive is to start a user
process running the EXERCISER for the terminal. Communication
between LISTENER processes and the executive is through shared data
segments. The format of the logon request is:

\texttt{START <user-id> <project-id> <class> [<cat>]}
user-id must be a decimal number greater than 7 and less than 32767.
project-id must be a decimal number greater than 1 and less than 127.
class must be T, S, C, or U.
cat is an optional parameter that can be any 16 bit decimal number.
The LISTENER does not perform any type of user authentication.

Record 8

Record 8 contains the EXERCISER, a test program that permits a
user at a terminal to invoke arbitrary Kernel functions with arbi-
trary input parameters. Further details are given in the enclosed
outline, "Using the Kernel Exerciser".

GENERATING THE PDP-11 TAPE

The PDP-11 tape is generated (on the S/360) with the following
JCL and control cards:

```
// EXEC WLSTAPE
BOOT
STARTUP 1
VDUMP
KERNEL 4
EXEC 1
LISTENER 2
EXERCISE 10
```

columns: 1

The catalogued procedure WLSTAPE and the program it invokes,
RELOCATE, are included on the OS/360 tape. RELOCATE writes record 1
onto the tape and then starts reading the control cards. Each con-
trol card gives the member name of a program in a library and a
relocation indicator. (Ø is assumed if the relocation indicator is
omitted.) For each control card—RELOCATE reads the program into
S/360 core with a LOAD macro, undoes the relocation performed by OS,
redo's the relocation for the PDP-11, performs a byte reversal
(because the PDP-11 puts the even byte in the right hand side of the
halfword), and dumps the program onto the tape. RELOCATE is able to
redo the relocation because it knows the format of object programs
produced by the SUE-11 compiler—only VCON's need be relocated, and
all VCONs begin at a fixed point relative to the beginning of each
procedure. RELOCATE requires that the entry point of each program
be a relative \( \theta \). This requirement is easily satisfied by following a simple convention for compilations and link edits. The relocation factor for the PDP-11 is the relocation indicator \( x^2 \) (hexadecimal). RELOCATE is not idiot proof - an improperly formed program could cause it to loop.

Since it is likely that you will have to modify our PDP-11 software, the OS/360 tape includes the bulk of our program development environment, in addition to the PDP-11 source and object. The document entitled "Project 7070 versus IBM 370 OS/TSO and The Project SUE System Language", should serve as a guide to our software development system. The output of the job that created the tape, OSTAPE2, is included. It should be sufficient to determine how to unload the tape. The following paragraphs describe each file on the tape. Unless otherwise noted, each file is an unloaded partitioned data set. An alias filename is given whenever it may be referenced in two different ways.

**File 1 - SUE.P7070.LINKLIB**
(Alias: SUE.VERSION1.P7070.LINKLIB)

This library is used as the input to the PDP-11 tape generation process. It contains OS load modules with all external references resolved.

**File 2 - SUE.GN.KERNEL.LINKLIB**

The object deck output of compilations is run through the linkage editor and into this library - it contains load modules with unresolved external references. A subsequent link edit that resolves the external references uses this library as input and the File 1 library as output.

**File 3 - TS0231.SUE.GN.KERNEL.SOURCE**

This file contains the source card images of most of the PDP-11 software. Most of the members contain SUE code, but some contain PAL-11 source. A copy of the PAL-11s cross assembler that we use can be obtained for $25.00 from

Mr. William F. Decker
University of Iowa
Iowa City, Iowa
File 4 - TSØ231.SUE.P7070.SOURCE
   (Alias:  SUE.VERSION1.P7070.SOURCE)

   This file contains the source card images for the rest of our
   PDP-11 software.

File 5 - SUE.GN.KERNEL.DATA

File 6 - SUE.GN.KERNEL.PROGRAM

File 7 - SUE.P7070.DATA
   (Alias:  SUE.VERSION1.P7070.DATA)

File 8 - SUE.P7070.PROGRAM
   (Alias:  SUE.VERSION1.P7070.PROGRAM)

   The members in these files are the output of the SCRUNCH pre-
   processor and the input to the SUE-11 compiler.

File 9 - SUE.DISTRIEB.LOADMOD
   (Alias:  SUE.VERSION1.LOADMOD)

   This file contains the object code of three programs - NIT,
   RELOCATE, and WLSALTER.  NIT transforms the output of the PAL-lls
   cross assembler we use into an OS object deck so that further pro-
   cessing by the linkage editor can take place.  NIT is not intended to
   be fully general or idiot proof, and will only work for small and
   simple programs (no external references) of the type that we have
   written.  The catalogue procedure that invokes NIT is WLSPAL.  The
   following control cards should be used:

   //EXEC WLSPAL,FILE='<filename>',MEMBER=<name>,TSOID=TSØ231
   //NIT.SYSIN DD *
   <name>

   RELOCATE has already been discussed.

   The load module WLSALTER is used for maintaining source input as
   card images on a private disk pack.  A listing of the user's source
   is always produced after execution of the ALTER program.  This pro-
   gram is a modification of a MITRE utility program written in OS
   Assembler.  Line numbers are added to the print file records when a
   change is made to the source file, but the OUTPUT subroutine's out
   file is never numbered.  The ALTER program is referenced in the
   "Project 7070 versus IBM 370..." document (SUEAS) with further de-
   tails found in the enclosed documentation entitled "Using the ALTER
   Program".
File 10 - SUE.WLS.SYSTEM.SOURCE
(Alias: SUE.VERSION1.SYSTEM.SOURCE)

This file contains the source of NIT, RELOCATE, and WLSALTER. NIT is written in PL/1 and can probably be compiled by the F or Optimizing compilers. RELOCATE and WLSALTER are written in OS Assembler. RELOCATE uses a few macros included on this tape.

File 11 - SUE.VERSION1.PROC

This file contains catalogued procedures. The procedures that begin with "SUE" are documented in "Project 7070 versus IBM 370..." - they are similar but not identical to the procedures distributed by the University of Toronto. The only other procedures are WLSPAL and WLSTAPE.

File 12 - WLS.MACLIB

OS Assembler macros used by RELOCATE.

File 13 - TS0231.JCL
(Alias: SUE.VERSION1.JCL)

This file contains JCL and control cards to compile/assemble all of the PDP-11 source on the PDP-11 tape.

CONFIGURATION DEPENDENCIES

Our PDP-11 software has configuration dependencies imbedded in program constants and code. There are three types of dependency - main memory, secondary storage, and terminals. A constant in the context block KERNEL, MEM_SIZE, sets our memory size to 128K bytes. If you make this constant larger you must also increase the size of the Memory Block Table, and this change will require adjusting the locations of all the Kernel data structures that follow it. There should be no problems running in less than 128K bytes if restricted to two or three processes. Main Memory requirements and the impact on the number of processes that can be used have not yet been adequately determined.

The Kernel supports an RF11 for secondary storage. Only the PAL-11 routine DISKIO in the Kernel knows about the RF11. If you are willing to restrict your experimentation you can probably just turn off the disk I/O, removing the calls to DISKIO and the P's on the disk semaphore from SWAPIN and SWAPOUT. (See Dijkstra's "The" paper for a discussion of this topic.)
We currently support four TTY-like terminals. The UNIBUS addresses of the control registers for these terminals have been set so that they all begin at the same offset relative to a 256 byte segment. The included configuration chart gives a complete listing of our UNIBUS addresses. One terminal has the standard address for the system console. Changing the number of terminals requires, among other things, changing the constant USER_PROCESS#_MAX in the context block NOFORM, changing the code in STARTUP that initializes I/O vectors and wires down I/O segments, adding interrupt handlers to the GATE program in the Kernel, changing the constants in PROGRAM EXEC that define ASTE#'s for I/O and program code segments, and changing the code in EXEC that puts I/O segments into the hierarchy.

**USING THE ALTER PROGRAM**

The ALTER program uses two input files and three output files. The data to be updated is referenced as a sequential input file called IN. The second input file is used to contain the ALTER control cards, which specify additions or deletions of whole cards and the changes to a given card image. Alterations to be input file are made on the basis of their numeric sequence number as read in (not on the basis of any number which may appear on the card image).

When the ALTER program copies the user's input file, two output files are generated, one for the printer (PRINT) and the other for the user's updated version (OUT). The third sequential output file is used to list the control cards and any error messages.

The MFT and MVT versions of OS/370 will generate a //SYSIN DD * card automatically if they encounter unspecified data cards in the input stream. If SYSIN is present (including DD DUMMY and the implied DD * cases), then sequence numbers will be generated on the output records in columns 77-80; otherwise the records will be copied unmodified.

**CONTROL CARDS**

A control card is a card from the SYSIN data set having a number sign (#) in column one. There are three types of control cards (described below). All control cards have an integer immediately following the number sign. The control cards must be arranged so that these integers (alter numbers) are in strictly ascending order. A card from SYSIN that is not a control card is copied to OUT and PRINT files after the last mentioned alter number. Note that insertions may follow all types of control cards and that it is not possible to insert a card commencing with a number sign.
1. **Position control card:**

```
#n
```

This card begins with a number sign, an integer, and two blanks; anything else on the card is ignored. It causes no change directly but is used to position the input for insertions.

2. **Deletion control card:**

```
#n,m
```

This card begins with a number sign, an integer, a comma, another integer, and two blanks; anything else on the card is ignored. The second integer must be not less than the first. The corresponding cards from the IN file (n through m inclusive) are deleted (not copied to OUT and PRINT files). Replacements for the deleted cards can, of course, appear after the deletion control card.

3. **Alter control card:**

```
#n A .pattern.replacement[.]
```

This card begins with a number sign, an integer, a blank, an "A", another blank, a control character, a pattern, another control character, a replacement string, and optionally a third control character. The control character may be any character (including blank). The pattern may be any string of one or more characters excluding the control character. The replacement may be any string; it is considered to end at the last non blank character on the card unless the character is a control character, in which case the replacement terminates at the character to the left of the last non blank character. Note that the replacement can have zero length or can itself contain instances of the control character. Note also that comments are not permitted on this control card.

The effect of the alter control card is to replace the first (from left to right) instance of the pattern with the replacement. The characters to the right of the pattern on the original card are moved left or right as necessary except that blanks are propagated left from column 77 (the sequence number field is not eligible for left shifts). Any characters shifted right from column 76 are lost. Only one replacement may be made on a card.

**ERROR CONDITIONS**

1. Alter number too large. (Return code 4.) This condition occurs if an alter number that is greater than the number of input records is encountered. The remaining input on SYSIN is ignored.
2. Illegal control card. (Return code 8.) This can mean an incorrect format or a card out of sequence; in either case it is ignored.

3. Match not found for alter. (Return code 16.) If no match is found then no change is made.

The return code from the ALTER program is the sum of the individual codes except that each is counted at most once. Thus, a return code of 20 would mean that SYSIN input was ignored, one or more alter matches failed, and there were no illegal control cards.

Example

11 EXEC SUEAS, FILE='SAMPLE'
#2,2
HELLO SUE
#38 A .DATA.
#44 A .DSN.DS.
#88,90
/*

The changes caused by the above control cards would be:

1. The second card image is replaced by the data line "HELLO SUE".
2. The first occurrence of "DATA" on the 38th card image is deleted.
3. The "DSN" is changed to "DS" on the 44th card image.
4. Card images 88 through 90 are deleted.

USING THE KERNEL EXERCISER

The basic functions of the Security Kernel as it is currently running on the PDP-11/45 may be accessed by way of the Kernel Exerciser. Outlined below are the procedures for calling up and running this program.

I. Steps to mounting the system tape and initiating execution:

1. Mount the system tape - PDPTAPE1 - on drive 0 and press LOAD. The ONLINE button should light.
2. On the system operator's console, the control knobs
for the Address Display Select should be set at CONSOLE PHYSICAL, and for the Data Display Select at DATA PATHS.

3. Boot load the system tape by entering 773136(octal) into the console switch register. Press in sequence HALT, LOAD ADDR, ENABLE, START.

4. The TTY Decwriter, and two TELTERM scopes should echo a carriage return (cr) signaling that the keyboards are unlocked. (Note that all communication is in TTY mode.)

II. Logging onto the system:

Every user logs onto the system by entering:

START /userid/ /projectid/ /classification/ /category/ (cr)

/userid/ - any decimal integer from 8 to 32766
/projectid/ - any decimal integer from 2 to 126
/classification/ - U (unclassified), S (secret), C (confidential), or T (top secret)
/category/ - any decimal integer from -32768 to 32767

The system response is: HELLO, THE KERNEL EXERCISER IS IN CONTROL

At this point, any of the Kernel commands, as described in Section IV, may be entered with the appropriate arguments.

III. Error Recovery

Should unrecognizable data be entered, the system response is:

ERROR, TRY AGAIN

Cancel a line by pressing the line feed key; correct a line by using the "@" key as a backspace key.
IV. The Kernel Commands

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>ARGUMENTS</th>
<th>RETURN CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>/seg#/ /offset/ /class/ /cat/</td>
<td>OK or ERROR</td>
</tr>
<tr>
<td>DELETE</td>
<td>/seg#/ /offset/</td>
<td>OK or ERROR</td>
</tr>
<tr>
<td>GIVE</td>
<td>/seg#/ /offset/ /mode/ /user/ /project/</td>
<td>OK or ERROR</td>
</tr>
<tr>
<td>RESCIND</td>
<td>/seg#/ /offset/ /user/ /project/</td>
<td>/seg#/ or ERROR</td>
</tr>
<tr>
<td>GETW</td>
<td>/seg#/ /offset/</td>
<td>/seg#/ or ERROR</td>
</tr>
<tr>
<td>GETR</td>
<td>/seg#/ /offset/</td>
<td>none</td>
</tr>
<tr>
<td>RELEASE</td>
<td>/seg#/</td>
<td>OK or ERROR</td>
</tr>
<tr>
<td>ENABLE</td>
<td>/seg#/ /reg#/</td>
<td>none</td>
</tr>
<tr>
<td>DISABLE</td>
<td>/reg#/</td>
<td>OK or ERROR</td>
</tr>
<tr>
<td>P</td>
<td>/seg#/</td>
<td>OK or ERROR</td>
</tr>
<tr>
<td>V</td>
<td>/seg#/</td>
<td>OK or ERROR</td>
</tr>
<tr>
<td>T</td>
<td>/seg#/</td>
<td>none</td>
</tr>
<tr>
<td>IPCSEND</td>
<td>/process#/ /message/</td>
<td>none</td>
</tr>
<tr>
<td>IPCRCV</td>
<td>none</td>
<td>(OK, /message/, /process#/)</td>
</tr>
<tr>
<td>STOPP</td>
<td>none</td>
<td>STOPP ACCEPTED</td>
</tr>
<tr>
<td>READIR</td>
<td>/seg#/ /offset/</td>
<td>(OK, /class/, /cat/, /seg_type/, /size/) or ERROR</td>
</tr>
</tbody>
</table>

ARGUMENT VALUES

<table>
<thead>
<tr>
<th>********</th>
<th>**</th>
<th>**</th>
</tr>
</thead>
<tbody>
<tr>
<td>/class/:</td>
<td>1-4</td>
<td>1 - unclassified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 - confidential</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 - secret</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 - top secret</td>
</tr>
<tr>
<td>/cat/:</td>
<td>0-32767</td>
<td></td>
</tr>
<tr>
<td>/message/:</td>
<td>1-32767</td>
<td></td>
</tr>
<tr>
<td>/mode/:</td>
<td>W - write access</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R - read access</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N - no access</td>
<td></td>
</tr>
<tr>
<td>/offset/:</td>
<td>1-63</td>
<td>Root offsets -</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - Process Directory Directory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 - I/O Directory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 - Code Directory</td>
</tr>
</tbody>
</table>
/process#: 1 - executive  
2 - TTY  
3 - DECwriter  
4 - Telterm #1  
5 - Telterm #2

/project/:  1-127  
1 - System project  
127 - All Projects

/reg#/:  0-15  
0 - Stack reg  
1 - I/O reg  
2 - Code reg

/seg#/:  1-31  
1 - Root seg  
3 - Code seg  
4 - Stack seg  
5 - I/O seg

/seg-type/  
0 - data  
128 - directory

/size/  
1 - 256 bytes  Only size 4 (1K bytes) is implemented  
4 - 1k bytes  
16 - 4k bytes

/user/:  8-16383  16383 - All users

ERROR CODES  
***** *****

-1: OK  Operation performed  
-2: ERROR  Security or Implementation Violation  
-3: SEVERE ERROR  Parameter out of bounds or segment not in address space

PROJECT 7070 VERSUS IBM 370 OS/TSO AND THE PROJECT SUE SYSTEM LANGUAGE

The Project SUE System Language and compiler were chosen as tools to implement the security model on the PDP-11/45 because its structure allows the construction using structured programming (the language has no GOTO statement) of an operating system which, as far as a high level language and machine code is concerned, can be proved correct. This language supports Top-Down construction and testing of all modules within the limited purpose operating system being developed by Project 7070. The compiler and assembler
run on the IBM 370 and the conventions described here were invented to help the programmer to cope more easily with the intricacies of the program structure, compiler requirements and OS/370 and thus allow the programmer to concentrate on producing correct code.

All Project 7070 files are kept on-line and TSO is used to edit those files because it is faster than a batch editor and one avoids the pitfalls of keypunch errors, errors in JCL and long waits, among other things. Jobs can be submitted either through normal batch processing or through the Remote Job Entry (RJE) facility of TSO and output is received in the ordinary manner. If one wishes one can fetch output to an on-line data set and examine it at the terminal. These possibilities are limited only by budget and the programmer's imagination.

The Sue Program Structure

The SUE System Language is a block-structured language somewhat like ALGOL, but with COBOL-like structures known as compilation blocks. There exist in the language three types of these structures, Context Blocks, Data Blocks and Program Blocks. The Context Blocks contain declarations for global types, absolute locations for variables in virtual space and global Macro declarations. The Program Blocks contain executable code and all declarations of local variables and types if that is feasible.

Any Sue program or procedure must consist of a Data Block (which need not contain any statements) and a Program Block both of which have the same name in the Block Head (e.g., 'Data NAME1;' are headings for the Data and Program Blocks respectively of a program called NAME1). If the program contains no internally called procedures, the Data and Program Blocks can be fed into the compiler one right after the other. If, on the other hand, internal procedures are declared, then the Program Block for any procedure must be preceded by its own data block and those of its enclosing procedures and must be followed by the Program Blocks of its enclosing procedures. The proper sequence for Data and Program Blocks is illustrated in Figure 1a. Note that the structure in Figure 1b is similar to nested Fortran DO loops and that the outermost Program Block may be omitted if the programmer does not wish it to be compiled. This compilation schedule is responsible for the structure of the Project 7070 SUE files.
The Compilation Procedure

There are two steps in any compilation of the source code of the SUE System Language - 1) a preprocessing step called SCRUNCH which produces input to 2) the compile step. Because of this scheme every SUE file has 4 PDSs allocated to it for code - 1 PDS containing source code, 2 PDSs containing SCRUNCHED Code and 1 PDS containing link edited object code. Figures 2a and 2b summarize the functions and illustrate the naming conventions for these PDSs.

The SOURCE PDS has members whose names are the procedure names which may be suffixed by either "D" or "P". For stand-alone Procedures, both the Data and Program Blocks are contained in one member with the same name as the procedure name. Procedures containing internal procedures have Data and Program blocks in separate members with names suffixed by "D" and "P" respectively. For example - a stand-alone procedure called NAME1 would have all its source code contained in a member of the SOURCE PDS called NAME1 - in other words it exists as one member in the source PDS; while a procedure named NAME2 containing internal procedures would have its Data Block contained in a SOURCE PDS member called NAME2D and its Program Block contained in a SOURCE PDS member called NAME2P - thus this program, unlike the stand-alone procedure, exists as 2 members in the SOURCE PDS rather than just 1. At the beginning and end of each member are control statements (beginning with "$$") which tell the Scrunch Preprocessor when an end of file condition has been reached (last statement of SOURCE PDS member) and where and under what name to place the output from the scrunch step (the first statement of the SOURCE PDS member). In Figures 3a and 3b we see that each member begins with the $$OUT_NAME statement. The parentheses immediately to the left of the equals sign contains the name of the particular Scrunch PDS (Data or Program) to which the output from the preprocessor will be read. All SOURCE member names ending with "D" will have "DATA" within the parentheses. Those ending with "P" will have "PROGRAM" within the parentheses. To the right of the equals sign is placed the name by which the member will be called. This name is, by convention, the Procedure or Program name unless that name is more than 8 letters or contains a break character ('_'). The last statement of each SOURCE PDS member is the $$EOF which indicates an end of records condition to the preprocessor.
Figure 1a. A SUE Program (Generalized) With No Internal Procedures
Figure 1b. Schematic Diagram Showing Compilation Order and Scope of Various Blocks of a Procedure with Internally Declared Procedures
1) **SOURCE** - card images which are input to the SCRUNCH Preprocessor

2) **DATA** - output from SCRUNCH, input to compiler

3) **PROGRAM** - output from SCRUNCH, input to compiler

4) **LINKLIB** - Link edited object code, output of the compiler

Figure 2a. Names of SUE PDSs Which Contain Code and their Function

SUE. `<filename>.<PDSname>`

N.B The Source PDS has the programmer's TSO account number prefixed to the data set name.

`<file name>`, by convention, is the programmer's initials followed by a period and the filename (e.g., JAL.SYSPROC)

Figure 2b. Generalized Names for USE PDSs.
Figure 3a. Generalized Diagram of Source Code for Member Name1 Showing Scrunch Control Cards. (This is a procedure without contained procedures. Resides in Source PDS as NAME1.)

Figure 3b.1 Data Block for a Program Called Program Name Which has Internal Procedures. (Resides in SOURCE PDS as NAME2D.)
The Scrunch Step

In order to call the Scrunch Preprocessor, the programmer codes a JCL EXEC control card calling the Procedure SUES specifying the file name (FILE='<file name>') and member name (MEMBER='<membername>'). The procedure will invoke the Scrunch Program using the parameters given by the programmer to identify the Data Set Name and Member Name of the member of the SOURCE PDS to be scrunched. SUES must be invoked for every member to be scrunched and the procedure will overwrite any existing scrunched code in the member of the Scrunch Data Set if that member exists. If that member does not exist it will be created. Therefore, when changes are made to source code, that code can be scrunched without first having to delete the scrunched code associated with the original source code.
The Compilation

The compiler is invoked by one of 2 procedures - SUEC or SUECL. The first is intended to produce only a listing with error messages, the data set containing object code being deleted by OS. The second procedure passes the object code to the Linkage Editor, which it invokes.

When calling SUEC the programmer must specify the filename (FILE='<filename>') and include control records after the EXEC card which tell the compiler which members of what Scrunch PDS to compile and in what order these members are to be compiled. Figures 4a and 4b give the general form and an example.

Link Editing

SUECL is called in much the same way and with the same control record scheme. The only difference is in the addition of a specification of a member name (MEMBER=<membername>). This will be substituted in the JCL for this procedure to name a member in the LINKLIB PDS to which the link-edited object code will be written. The Linkage Editor is invoked in this procedure with the NCAL option specified. This allows the storage of object modules with unresolved external references until the segments of code to which they refer have been coded, compiled and separately link-edited. After the programmer has coded all procedures for a particular system of procedures (i.e., after all internal procedures have been coded, compiled and object modules placed in the LINKLIB PDS), SUEL is called to invoke the Linkage editor, the NCAL option, in order to completely link edit all object modules of a system of procedures and store the resultant load module in the Linklib PDS. The programmer is required to specify in the EXEC Statement a value for a member name in the LINKLIB PDS (MEMBER=<membername>) to which the load module is to be written and s/he will be required to use control statements of the form - INCLUDE SYSLIB(<membername>) - which indicates to the Linkage Editor which members (<membername>) of the Linklib PDS are to be processed. There will be as many of these statements as there are members to be link edited. The process will bomb if there are any procedure calls for which no procedure has been coded, compiled and stored in the Linklib PDS.

USING TSO

In order for the Programmer to use TSO each file is allocated 2 additional PDSs: 1) a CNTL PDS with a data set name of
Figure 4a. General Form for Compiler Control Records

```
// EXEC SUEC, FILE='<file name>'
DATA NAME1 -L  (no listing of this block)
DATA NAME2 D   (list symbol and type tables)
DATA NAME3
PROGRAM NAME3
DATA NAME 4
PROGRAM NAME4
PROGRAM NAME2 DE  (list symbol and type tables, emitted code)
/*

Figure 4b. Example of Invocation of Compiler with Control Records
```
<tsoid>. CNTL and 2) A CLIST PDS with a data set name of <tsoid>. X.CLIST. The first is storage for JCL card images and other control cards for submission to batch processing through the Remote Job Entry (RJE) facility of TSO (see caveat #3 before using RJE). The second is storage for any command lists which a programmer may want to create. One useful command list passes values for a file name and a member name to a generalized call to the editor for a member of a PDS and executes the call. Other uses are left to the programmer's imagination.

Project 7070 uses a catalogued procedure to allocate PDSs for files. The programmer calls the catalogued Procedure SUEALLOC and specifies his TSO account number (TSOID=<tsoid>) and the file name (FILE=<filename>).

USING MITRE BATCH FACILITIES

All procedures mentioned here can be called in the batch job stream. Members can be created by IEBUPDTE; they can be updated by using the procedure SUEAS which calls the ALTER program and uses the same utility control statements (see the Facility Manual). In addition this procedure rewrites the Scrunch data set. There is a card to printer procedure called SUESC which takes card input and produces a compiler diagnostic. With TSO, however, these two procedures are not necessary and all the others may be called through JCL which the programmer has stored in dynamically created CNTL data sets. Of course, in order to use TSO the programmer has to have some way of identifying himself to the system as a legitimate user and this is done through the TSO account number.

GETTING READY FOR TSO

In order to obtain a TSO account number (which is also the "TSOID") the programmer can call User Assistance at X2525, and after giving Lee Gera the pertinent information, he will assign the programmer a number of the form TS0XXX and an eight character Password. After Lee places the programmer's account on the system the programmer may proceed to do his thing. However, to do it on TSO s/he must first read the TSO User's guide (GC28-6697) which will introduce her/him to TSO and have ready the TSO Command Language Reference which will serve as a reference guide to the Function, Syntax and Operands of TSO commands. In addition, the programmer will have to have a MITRE Computer Facilities Manual in order to see the changes and extensions that MITRE has made to the original IBM Product in order to make it more compatible with humans. These changes and extensions are listed and explained in Chapter 7. All these and any other manuals may be obtained from Stella Theokas in the keypunch area of the computer facility.
Caveats

(1) It may not be obvious from the following attachments that a SUE file resides on 2 disk packs. The SOURCE, CNTL and CLIST PDS-reside on the public pack and the DATA, PROGRAM and LINKLIB PDSs reside on a private pack (serial number DP5006). With the present LOGON procedure available to Project 7070 (the default procedure), only those PDSs residing on the public packs can be accessed through TSO. However, others may be assessed by any Job entered through RJE.

(2) All control cards for the compiler and Linkage Editor begin in column 1.

(3) One slight failing of RJE is that there is no way for the operator to know that a private storage medium is required unless the JCL for the Job tells him. This is done at MITRE through a HASP SETUP card image. This is described in the Facilities Manual on Page 6.2. The user should request Volume DP5006 - a disk pack.

Summary with Examples of SUE Procedure Calls

1) SUEALLOC: Allocates the 6 PDSs required for working with the SUE language. SOURCE, CNTL and CLIST are allocated on the Public Packs while DATA, PROGRAM and LINKLIB are allocated to the Project 7070 private Pack. File name must be specified.

   e.g., //anyname EXEC SUEALLOC,FILE='JAL.SYSPROC',TSOID=TS0999

2) SUES: Scrunches source code and places the result on either the Data or Program PDS according to what control records appear with the source code (see Figures 3a and 3b). The programmer's TSOID, file name and member name of the Source PDS member to be processed, must be specified.

   e.g., //anyname EXEC SUES,FILE='JAL.SYSPROC',MEMBER=DELETEP,
       TSOID=TS0999

3) SUEC: Compiles scrunched code, produces listings and diagnostics and scratches files containing object code. Programmer's file name must be specified and control records must be placed after the EXEC CARD (see the section on compilation and Figure 5b).
e.g., //anyname EXEC SUEC, FILE='JAL.SYSPROC'
     *
     *
compiler control records
     *
     *
     */

N.B. the member name specified is the member in the LINKLIB PDS to which the object code will be written.

5) SUEL: Link edits files specified by control records (see section on Link Editing) and terminates at discovery of unresolved external reference or at successful completion. When successful, places fully link edited code in LINKLIB PDS under a specified member name. File name and member name must be given.

e.g., //anyname EXEC SUEL, FILE='JAL.SYSPROC', MEMBER=DELETE
     *
     *
     Linkage Editor
     Control Records
     *
     *
     */

6) SUEAS: Batch editor for Project 7070. Calls Alter and Scrunch Programs. Filename and member from SOURCE PDS must be specified. Control cards and change cards are as for the ALTER Program. See the Facility Manual Page 5.80 for more information.

e.g., //anyname EXEC SUEAS, FILE='JAL.SYSPROC', MEMBER=DELETEP
     *
     *
     control and change cards for ALTER
     *
     *
     */

7) SUESC: Executes scrunch and compile on card input. Output is to printer for listing and diagnostics. All files are temporary.

e.g., //anyname EXEC SUESC
     *
     *
     card input
     *
     *
     /
8) SUESCRTH: Deletes all PDSs for a particular file name. TSOID and file name must be specified.

e.g., //anyname EXEC SUESCRTH,TSOID=TS(999),FILE='JAL.SYSPROC'

Examples of Program Execution and Development Under TSO

A guide and scenarios are given below to illustrate the steps a SUE programmer must use to allocate files for her/himself, create and edit PDS members and submit jobs.

Allocation of Files

The allocation of files can be done either through TSO or through the batch. It seems frivolous to spend money on the submission of such a short job through RJE so batch processing is recommended. The cards needed are a job card and the execute card for SUEALLOC (see example in the previous section). Don't forget to put the Serial # DP5006 on the back of the green request card under "Disk Packs Required".

Creating Members in Your Newly Allocated Disk Area

There are two ways of doing this - IEBUPDTE with cards or through TSO. IEBUPDTE JCL is first given. For a more comprehensive look at this utility the programmer should consult the OS Utilities Manual (GC28-6586).

```
//jobcard (if necessary)
// EXEC PGM=IEBUPDTE, PARAM-NEW
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=<tsoid>.<filename>.SOURCE,DISP=SHR
//SYSUT2 DD DSN=<tosid>.<filename>.SOURCE,DISP=SHR
//SYSIN DD *
.* ADD LIST=ALL,NAME= membernamel
.* ADD LIST=ALL,NAME= membername2
  (card images)

* *
*/
```

N.B. <membernamel> and <membername2> are the names the programmer has chosen according to the conventions stated previously for source PDS members. There may be any number of ADD statements—one for each member to be added.
To do the above through TSO one would make the following call to the editor:

```
edit '{tsoid}.SUE.<filename>.SOURCE(<membername>)}
DATA NEW NONUM
```

Since the member is new the user is placed into input mode immediately and is ready to write whatever input s/he wants. One must do this for each member to be placed on the PDS. Typing out such a long command can become tedious when one does it over and over again - as one sometimes must for a large series of editing changes to many members of many data sets - therefore, the CLIST facility is used. The following command is used to create the member whose membername is 'e' in the CLIST PDS named X:

```
edit x(e) clist new
```

When the terminal is in input mode the following entries are made:

```
proc 2 fname mem
eedit'{tsoid}.sue.&fname..source(&mem.}'data nonum
```

N.B. This is for editing an already existing member. If one wanted to create a new member one would place the keyword "NEW" after the attribute parameter. A carriage return brings the terminal to edit mode and the command

```
se
```

saves and ends the edit session. To learn more about the PROC statement in TSO please consult the TSO Command Language Reference. When one wants to call the editor one types in the exec command specifying the file name and member name like so:

```
exec x(e) '{filename}<membername>}
```

The terminal will be returned to you in Edit mode for the member specified, or input mode if NEW is specified in the CLIST.

In order to run jobs through RJE one must have JCL card images on disk and accessible to the terminal. The CNTL PDS is the data set which will contain the JCL necessary. In order to place the JCL into members of the PDS one uses the EDIT command to create the new member:

```
edit cntl(<membername>) cntl new nonum
```
Now the terminal is in input mode and one can enter the JCL and, when necessary, any other information such as compiler or linkage editor control records and setup records. Here is a sample scenario where members of the CNTL PDS are created for scrunching a program called WRITES (which contains internal procedures) and compiling it with already scrunched NOFORN (a context block), WHYNOT (another context block), SEARCH (a stand along procedure), APPENDI (which has internal procedures) and APPENDS (another stand alone procedure), all of which are in the SUE file 'JAL.SYSPROC'. <CR> means that the carriage return button is pushed after a line is entered. The command SE saves the input and/or changes and exits from the command EDIT.

1) Make a Job Card

Command: edit 'ts0999.cntl(jobcard)' cntl new nonum

Input: //TS0999A JOB (7070, D73, DESK), 'LARKINS JA', CLASS=D
       // REGION=256K, TIME=(20), NOTIFY=TS0999

   <CR>

Command: se

2) Scrunch both the DATA and PROGRAM blocks of WRITES

Command: edit 'ts0999.cntl(SWRITES)' cntl new nonum

Input: //SCRUNCHD EXEC SUES, FILE='JAL.SYSPROC', MEMBER=WRITESD, TSOID=TS0999
       //SCRUNCHP EXEC SUES, FILE='JAL.SYSPROC', MEMBER=WRITSP, TSOID=TS0999

   <CR>

Command: se

3) Compile the program and produce only diagnostics and listing.

Command: edit 'ts0999.cntl(cmpwrite)' cntl new nonum

Input: //CMPWRITE EXEC SUEC, FILE='JAL.SYSPROC'
       DATA NOFORN (no listing)
       DATA WHYNOT
       DATA WRITE D (symbol and type table listed)
       DATA SEARCH

   255
DATA APPENDI
DATA APPENDS
PROGRAM APPENDI
PROGRAM WRITES DE  (Emitted Code, Symbol table
and type table listed)

<CR>

Command:  se

4) Submit the job through RJE

Command:  sub (cntl(jobcard) cntl(swrites)cntl(cmpwrite))

A HASP job number will be returned by the system and should be
copied down for use by the STATUS command and other commands which
are outlined in the Facility Manual.

If the programmer wanted to link edit the object code and store
it the following step would replace step 3.

Command:  edit 'ts0999.cntl(cmpwrite)' cnt1 new nonum

Input:  //CMPWRITE EXEC SUECL,FILE='JAL.SYSPROC',MEMBER=WRITES
        DATA NOPORN   L
        DATA WHYNOT   L
        DATA WRITES   D
        DATA SEARCH   L
        DATA APPENDI  L
        DATA APPENDS  L
        PROGRAM APPENDI  L
        PROGRAM WRITES  E

The link edited object code will be stored in the member WRITES
of the LINKLIB PDS. The submit command would look the same.

If the programmer had finally coded his whole system of proce-
dures and placed each of their object codes in the LINKLIB PDS
and wanted to obtain fully link edited object codes s/he would code
the following JCL.

We assume that the members WRITEN, SPVRGATE, CHANGEB, IYPYE, READS
and READN are members of LINKLIB.

256
Command: edit 'ts0999.cntl(fileproc)' cntl new nonum

Input: //LINKED EXEC SUEL,FILE='JAL.SYSPROC',MEMBER=FILEPROC
INCLUDE SYSLIB(SPVRGATE)
INCLUDE SYSLIB(WRITES)
INCLUDE SYSLIB(WRITEN)
INCLUDE SYSLIB(READS)
INCLUDE SYSLIB(REDN)
INCLUDE SYSLIB(IYPE)
INCLUDE SYSLIB(CHANGEB)

/*

Command: se

Now to submit the job the programmer will issue the following command:

SUBMIT (CNTL(JOBCARD) CNTL(FILEPROC))

If the programmer wants to delete any members from the PDS on the PUBLIC PACK (SOURCE,CNTL,CLIST) s/he must use the delete command. The following is a command that deletes CNTL member SWrites.

delete cntl(swritesss)

The 3270 Display Terminal

The FSE subcommand of EDIT utilizes the 3270 display terminal to its fullest to ease the updating of card images. It can be used to edit the CNTL and SOURCE PDS members. This program enables the user to directly change her/his code on the screen without the mediation of the insert, delete, and change subcommands. For more information see pp. 7-5.2 ff of the facility manual.

The next section contains listings of the JCL for SUE Procedures.
//SUEALLOC PROC
//IEFBR14 EXEC PGM=IEFBR14,ACCT=COST
//SOURCE DD UNIT=PUBLIC,DISP=(NEW,CATLG),
// DCB=(DSORG=PO,RECFM=FB,BLKSIZ=3120,LRECL=80),
// SPACE=(TRK,(15,10,10)),
// DSN=SUE.6TSID..SFILE..SOURCE
//DATA DD UNIT=PACK,VOL=(PRIVATE,SER=EP5006),DISP=(NEW,KEEP),
// DCB=(BLKSIZ=2048,RECFM=F),SPACE=(2048,(100,100,100)),
// DSN=SUE.6FILE..DATA
//PROGRAM DD UNIT=PACK,VOL=(PRIVATE,SER=EP5006),DISP=(NEW,KEEP),
// DCB=(BLKSIZ=2048,RECFM=F),SPACE=(2048,(50,50,50)),
// DSN=SUE.6FILE..PROGRAM
//LINKLIB DD UNIT=PACK,VOL=(PRIVATE,SER=EP5006),DISP=(NEW,KEEP),
// DCB=(DSORG=PO,RECFM=U,BLKSIZ=7294),
// SPACE=(TRK,(20,10,10)),
// DSN=SUE.6FILE..LINKLIB
//CNTL DD UNIT=PUBLIC,DISP=(NEW,CATLG),
// DCB=(DSORG=PO,RECFM=FB,BLKSIZ=20000,LRECL=80),
// SPACE=(TRK,(5,5,10)),
// DSN=6TSID..CNTL
//X DD UNIT=PUBLIC,DISP=(NEW,CATLG),
// DCB=(DSORG=PO,RECFM=VB,BLKSIZ=1680,LRECL=255),
// SPACE=(TRK,(1,1,2)),
// DSN=6TSID..X.CLIST
//SOEAS PROC SCANNER=SCRUNCH,JOE=SUER,JGROUP=SUER,SCROUT=DUFLMY
//ALTER EXEC PGM=WLSALTER,ACCT=COST
//STEPLIB DD DSN=SUE.DISTRIB.LOADMOD,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSDUMP DD SYSOUT=A
//PRINT DD SYSOUT=A
//IN DD UNIT=PACK, VOL=(PRIVATE,SER=DP5006),DISP=SHR,
//     DSN=SUE.&FILE..SOURCE(&MEMBER)
//OUT DD UNIT=PACK, VOL=(PRIVATE,SER=EP5006),DISP=(OLD,PASS),
//     DSN=SUE.&FILE..SOURCE(&MEMBER)
//*
//*
//SCRUNCH EXEC PGM=KMON,REGION=100K,
// PARM='FREE=30000,JOE=SUER,JGROUP=SUER',ACCT=COST
//STEPLIB DD DSN=SUE.DISTRIB.LOADMOD,DISP=SHR
//PROGRAM DD DSN=SUE.DISTRIB.&SCANNER,DISP=SHR
//SYSPRINT DD &SCROUT
//SYSPUNCH DD DUMMY
//FILE1 DD VOL=(PRIVATE,SER=DP5006), DSN=SUE.&FILE..PROGRAM,
//     UNIT=PACK,DISP=OLD,
//FILE2 DD VOL=(PRIVATE,SER=DP5006), DSN=SUE.&FILE..DATA,
//     UNIT=PACK,DISP=OLD,
//FILE6 DD VOL=(PRIVATE,SER=DP5006), DSN=SUE.&FILE..PROGRAM,
//     UNIT=PACK,DISP=OLD,
//FILE7 DD VOL=(PRIVATE,SER=DP5006), DSN=SUE.&FILE..DATA,
//     UNIT=PACK,DISP=OLD,
//OUTPUT3 DD DSN=MODNAME,UNIT=SYSDA,DISP=(MOD,PASS),SPACE=(TRK,1),
//     DCB=BLKSIZE=90
//SYSIN DD DSN=.*ALTER.OUT,DISP=SHR
//SUESC PROC SCANNER=SCRUNCH,SCROUT=DUMMY,VERSION=SUE11,
// FREE=12000, JOBNAME=SUEGROUP, CNTBLD=60000
//SCRUNCH EXEC PGM=XMON,REGION=100K,
// PARM='FREE=30000, JOBNAME=6JOBNAME', ACCT=COST
//STEFLIB DD DISP=SHR, DSN=SUE.DISTRIB.LOADMOD
//PROGRAM DD DISP=SHR, DSN=SUE.DISTRIB.ESCRANNER
//SYSPRINT DD 6SCROOT
//SYSPUNCH DD DUMMY
//FILE1 DD DSN=6&TOKENS1,UNIT=SYSDA, DISP=(MOD, PASS),
// SPACE=(TRK,(10,5,17)),
// DCB=(DSORG=PO,RECFM=F,BLKSIZE=2048)
//FILE2 DD DSN=6&TOKENS2,UNIT=SYSDA, DISP=(MOD, PASS),
// SPACE=(TRK,(10,5,17)),
// DCB=(DSORG=PO,RECFM=F,BLKSIZE=2048)
//FILE6 DD DUMMY
//FILE7 DD DUMMY
//OUTPUT3 DD DSN=6&MCNAME, UNIT=SYSDA, DISP=(MOD, PASS), SPACE=(TRK,1),
// DCB=BLKSIZE=80
//*
//SUE EXEC PGM=XMON,REGION=310K,COND=(C,LT,SCRUNCH),ACCT=COST,
// PARM='FREE=6FREE, CONTBID=6CMTLD, JOENAM=6JOBNAME'
//STEFLIB DD DISP=SHR, DSN=SUE.DISTRIB.LOADMOD
//PROGRAM DD DISP=SHR, DSN=SUE.DISTRIB.ESCRANNER
//SYSPRINT DD SYSPUT=A, DCB=(RECFM=FBA, LRECL=133, BLKSIZE=532, BUFNO=2)
//FILE1 DD DSN=*.SCRUNCH.FILE1, DISP=(CLD, PASS)
//FILE2 DD DSN=*.SCRUNCH.FILE2, DISP=(CLD, PASS)
//OUTPUT3 DD DSN=6&SYSLIN, UNIT=SYSDA, DISP=(MOD, PASS),
// SPACE=(TRK, (20,5)), DCB=(RECFM=FB, LRECL=80, BLKSIZE=800)
//SYSIN DD DSN=6&MODNAME, DISP=(OLD, DELETE)
//SUES PROC SCANNER=SCRUNCH, JOBNAM=SUEGROUP //SCRUNCH EXEC PGM=XMON, REGION=100K, ACCT=COST, // PARM='FREE=30000, JOBNAME=SUEGROUP' //STEPLIB DD DSN=SUE.DISTRIBUTION.LOADMOD, DISP=SHR //PROGRAM DD DSN=SUE.DISTRIBUTION.SCANNER, DISP=SHR //SYSPRINT DD DUMMY //SYSPUNCH DD DUMMY //FILE1 DD VOL=(PRIVATE, SER=DP5006), DSN=SUE.FILE..PROGRAM, UNIT=PACK, DISP=OLD //FILE2 DD VOL=(PRIVATE, SER=DP5006), DSN=SUE.FILE..DATA, UNIT=PACK, DISP=OLD //FILE3 DD VOL=(PRIVATE, SER=DP5006), DSN=SUE.FILE..PROGRAM, UNIT=PACK, DISP=OLD //FILE4 DD VOL=(PRIVATE, SER=DP5006), DSN=SUE.FILE..DATA, UNIT=PACK, DISP=OLD //FILE5 DD VOL=(PRIVATE, SER=DP5006), DSN=SUE.FILE..PROGRAM, UNIT=PACK, DISP=OLD //FILE6 DD VOL=(PRIVATE, SER=DP5006), DSN=SUE.FILE..DATA, UNIT=PACK, DISP=OLD //FILE7 DD VOL=(PRIVATE, SER=DP5006), DSN=SUE.FILE..DATA, UNIT=PACK, DISP=OLD //OUTPUT3 DD DSN=SUE.NAME, UNIT=SYSDA, DISP=(MOD, PASS), SPACE=(TBK, 1), DCB=BLKSIZE=60 //SYSIN DD DSN=SYSID..SUE.FILE..SOURCE(SM36ER), DISP=SHR
//SUEC  PROC   VERSION=SUE11, FREE=12000,
//          JOBNAME=SUEGROUP, CNTRLD=60000, SYS=P7070
//SUE   EXEC   PGM=XMON, REGION=310K, ACCT=COST,
//          PARM='FREE=&FREE, CNTRLD=&CNTRLD, JOBNAME=&JOBNAME'
//STELIB    DD   DSN=SUE.DISTRIB.LOADMOD, DISP=SHR
//PROGRAM    DD   DSN=SUE.DISTRIB.VERSION, DISP=SHR
//SYSPRINT    DD   SYSOUT=A, DCE= (RECFM=FBA, LRECL=133, BLKSIZ=532, BUFNO=2)
//FILE1    DD   VOL= (PRIVATE,SER=DP5006), DSN=SUE.&FILE..PROGRAM,
//            UNIT=PACK, DISP=SHR
//FILE2    DD   VOL= (PRIVATE,SER=DP5006), DSN=SUE.&FILE..DATA,
//            UNIT=PACK, DISP=SHR
//OUTPUT3   DD   DSN=&SYSLIN, DISP=(MOD,PASS), UNIT=SYSDA,
//            SPACE=(TRK, (20,5)), DCB=(RECFM=F8, LRECL=80, BLKSIZE=800)
EXEC PGM=XHCN, REGION=310K, ACCT=COST,
PARM=('FREE=12000, CONTROL=6CNTRLD, JOBNAME=SUEGROIJP, SYS=P7070'

EXEC PGM=XHCN, REGION=310K, ACCT=COST,
PARM=('FREE=12000, CONTROL=6CNTRLD, JOBNAME=SUEGROIJP, SYS=P7070'

STEPLIB DD DSN=SUE.DISTLIB.LOADLIB, DISP=SHR

PROGRAM DD DSN=SUE.DISTLIB, VERSION, DISP=SHR

STEPLIB DD DSN=SUE.DISTLIB, LOADLIB, DISP=SHR

SYSPRINT DD SYSOUT=A, DCB=(RECFM=FB, LRECL=133, BLKSIZE=532, BUFNO=2)

FILE1 DD VOL=(PRIVATE, SER=DP5006), DSN=SUE.FILE..PROGRAM,
UNIT=PACK, DISP=SHR

FILE1 DD VOL=(PRIVATE, SER=DP5006), DSN=SUE.FILE..PROGRAM,
UNIT=PACK, DISP=SHR

FILE2 DD VOL=(PRIVATE, SER=DP5006), DSN=SUE.FILE..DATA,
UNIT=PACK, DISP=SHR

FILE2 DD VOL=(PRIVATE, SER=DP5006), DSN=SUE.FILE..DATA,
UNIT=PACK, DISP=SHR

OUTPUT3 DD DSN=SUE.SYSLIN, DISP=(MOD, PASS), UNIT=sysda,
SPACE=(TRK,(20,5)), DCB=(RECFM=FB, LRECL=80, BLKSIZE=800)

EXEC PGM=IEWL, REGION=96K, COND=(0, LT, SUE),
PARM=(XREF, LIST, NCAL), ACCT=COST

STEPLIB DD DSN=SUE.DISTLIB, LOADLIB, DISP=SHR

SYSPRINT DD SYSOUT=A

SYSLIN DD DSN=.*, OUTPUT3, DISP=(OLD, DELETE)

SYSLIN DD DSN=.*, OUTPUT3, DISP=(OLD, DELETE)

SYSTIN DD DNAME=SYSIN

SYSTIN DD DNAME=SYSIN

SYSLMOD DD UNIT=PACK, VOL=(PRIVATE, SER=DP5006), DISP=OLD,

SYSLMOD DD UNIT=PACK, VOL=(PRIVATE, SER=DP5006), DISP=OLD,

SYSLMOD DD UNIT=PACK, VOL=(PRIVATE, SER=DP5006), DISP=OLD,

SYSLMOD DD UNIT=PACK, VOL=(PRIVATE, SER=DP5006), DISP=OLD,
//SUEL    PROC SYS=P7070
//LK2D    EXEC PGM=IF, PARM='XREF,LIST', ACCT=COST
//SYSPRINT DD SYSOUT=A
//SYSLIN  DD DNAME=SYSLIN
//SYSLMOD DD UNIT=PACK, VOL=(PRIVATE, SER=DP5006), DISP=CLD,
//          DSN=SUE.SSYS..LI
//SYSUT1 DD UNIT=(DISK, SEP=SYSLMOD), SPACE=(1024, (200, 20))
//SYSLIB  DD VOL=(PRIVATE, SER=DP5006), DSN=SUE.SFILE..LINKLIB,
//         UNIT=PACK, DISP=SHR
// SUESCRTH PROC
// IZEPBR14 EXEC PGM=IZEPBR14, ACCT=COST
// CNTL DD DISP=(OLD,DELETE), DSN=TSO.OID..CNTL
// X DD DISP=(OLD,DELETE), DSN=TSO.OID.X.CLIST
// SOURCE DD DISP=(OLD,DELETE), DSN=TSO.OID..SUE.&FILE..SOURCE
// PROGRAM DD DISP=(OLD,DELETE), UNIT=PACK, VOL=(PRIVATE, SER=DP5006),
// DSN=SUE.&FILE..PROGRAM
// DATA DD DISP=(OLD,DELETE), UNIT=PACK, VOL=(PRIVATE, SER=DP5006),
// DSN=SUE.&FILE..DATA
// LINKLIB DD DISP=(OLD,DELETE), UNIT=PACK, VOL=(PRIVATE, SER=DP5006),
// DSN=SUE.&FILE..LINKLIB
# PDP-11/45 Interrupt Vectors and Control Registers - 2 June 1975

<table>
<thead>
<tr>
<th>Definition</th>
<th>Octal</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interrupt Vectors</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Time Out, Bus Error, Odd Address, Stack Violation</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Reserved Instruction</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>Debugging</td>
<td>14</td>
<td>E</td>
</tr>
<tr>
<td>IOT</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>Power Fail</td>
<td>24</td>
<td>14</td>
</tr>
<tr>
<td>EMT</td>
<td>30</td>
<td>18</td>
</tr>
<tr>
<td>TRAP</td>
<td>34</td>
<td>1C</td>
</tr>
<tr>
<td>Dec Writer In (BR4)</td>
<td>60</td>
<td>30</td>
</tr>
<tr>
<td>DEC Writer Out (BR4)</td>
<td>64</td>
<td>34</td>
</tr>
<tr>
<td>Programmable Real-Time Clock (KW11-P, BR6)</td>
<td>104</td>
<td>44</td>
</tr>
<tr>
<td>General Purpose DMA Interface (DR11-B, BR5)</td>
<td>124</td>
<td>54</td>
</tr>
<tr>
<td>Line Printer Control (LS11, BR4)</td>
<td>200</td>
<td>80</td>
</tr>
<tr>
<td>Disk Control (RF11, BR5)</td>
<td>204</td>
<td>84</td>
</tr>
<tr>
<td>DEC Tape Control (TC11, BR6)</td>
<td>214</td>
<td>8C</td>
</tr>
<tr>
<td>Magnetic Tape Control (TM11, BR6)</td>
<td>224</td>
<td>94</td>
</tr>
<tr>
<td>Card Reader Control (CR11, BR6)</td>
<td>230</td>
<td>98</td>
</tr>
<tr>
<td>Programming Interrupt Request</td>
<td>240</td>
<td>A0</td>
</tr>
<tr>
<td>Memory Management Unit Traps and Violations</td>
<td>250</td>
<td>A8</td>
</tr>
<tr>
<td>DIVA Disk (BR5)</td>
<td>254</td>
<td>AC</td>
</tr>
<tr>
<td><strong>Start of Floating Vectors</strong> (BR5 unless otherwise noted, each device has a pair of vectors - in and out)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Telterm #1 Control (BR4)</td>
<td>300</td>
<td>C0</td>
</tr>
<tr>
<td>Telterm #2 Control (BR4) (BR-90 Lab)</td>
<td>310</td>
<td>C8</td>
</tr>
<tr>
<td>Asynchronous Line Interfaces (DC11-DA)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#1</td>
<td>320</td>
<td>D0</td>
</tr>
<tr>
<td>#2</td>
<td>330</td>
<td>D8</td>
</tr>
<tr>
<td>Definition</td>
<td>Octal</td>
<td>Hex</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>#3</td>
<td>340</td>
<td>E0</td>
</tr>
<tr>
<td>#4</td>
<td>350</td>
<td>E8</td>
</tr>
<tr>
<td>Synchronous Interface (DP11-DA) BR90</td>
<td>360</td>
<td>F0</td>
</tr>
<tr>
<td>TTY</td>
<td>370</td>
<td>F8</td>
</tr>
<tr>
<td>End of Floating Vectors</td>
<td>377</td>
<td>FF</td>
</tr>
<tr>
<td><strong>Control Registers</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TTY (4 registers)</td>
<td>760160</td>
<td>3E070</td>
</tr>
<tr>
<td>DIVA Disk (8 words)</td>
<td>764000</td>
<td>3E800</td>
</tr>
<tr>
<td>Programmable Read-Time Clock (KW11-P) Secure Mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Count and Status</td>
<td>770540</td>
<td>3F160</td>
</tr>
<tr>
<td>Count Set Buffer</td>
<td>770542</td>
<td>3F162</td>
</tr>
<tr>
<td>Counter</td>
<td>770544</td>
<td>3F164</td>
</tr>
<tr>
<td>Supervisor Segmentation Registers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I Space Descriptor Registers (0-7)</td>
<td>772200</td>
<td>3F480</td>
</tr>
<tr>
<td>D Space Descriptor Registers (0-7)</td>
<td>772220</td>
<td>3F490</td>
</tr>
<tr>
<td>I Space Address Registers (0-7)</td>
<td>772240</td>
<td>3F4A0</td>
</tr>
<tr>
<td>D Space Address Registers (0-7)</td>
<td>772260</td>
<td>3F4B0</td>
</tr>
<tr>
<td>Kernel Segmentation Registers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I Space Descriptor Registers (0-7)</td>
<td>772300</td>
<td>3F4C0</td>
</tr>
<tr>
<td>D Space Descriptor Registers (0-7)</td>
<td>772320</td>
<td>3F4D0</td>
</tr>
<tr>
<td>I Space Address Registers (0-7)</td>
<td>772340</td>
<td>3F4E0</td>
</tr>
<tr>
<td>D Space Address Registers (0-7)</td>
<td>772360</td>
<td>3F4F0</td>
</tr>
<tr>
<td>General Purpose DMA Interface (DR11-B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word Count</td>
<td>772410</td>
<td>3F508</td>
</tr>
<tr>
<td>Bus Address</td>
<td>772412</td>
<td>3F50A</td>
</tr>
<tr>
<td>Status and Command</td>
<td>772414</td>
<td>3F50C</td>
</tr>
<tr>
<td>Data Buffer</td>
<td>772416</td>
<td>3F50E</td>
</tr>
<tr>
<td>11/45 SSR3</td>
<td>772516</td>
<td>3F54E</td>
</tr>
<tr>
<td>Definition</td>
<td>Octal</td>
<td>Hex</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>Magnetic Tape (TM11)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td>772520</td>
<td>3F550</td>
</tr>
<tr>
<td>Control</td>
<td>772522</td>
<td>3F552</td>
</tr>
<tr>
<td>Byte Counter</td>
<td>772524</td>
<td>3F554</td>
</tr>
<tr>
<td>Memory Address</td>
<td>772526</td>
<td>3F556</td>
</tr>
<tr>
<td>Data Buffer</td>
<td>772530</td>
<td>3F558</td>
</tr>
<tr>
<td>Read Lines</td>
<td>772532</td>
<td>3F55A</td>
</tr>
<tr>
<td>Programmable Real-Time Clock (KW11-P)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Normal Mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Count and Status</td>
<td>772540</td>
<td>3F560</td>
</tr>
<tr>
<td>Count Set Buffer</td>
<td>772542</td>
<td>3F562</td>
</tr>
<tr>
<td>Counter</td>
<td>772544</td>
<td>3F564</td>
</tr>
<tr>
<td>Bootstrap Loader</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disk (RF11)</td>
<td>773100</td>
<td>3F640</td>
</tr>
<tr>
<td>DEC Tape (TC11)</td>
<td>773120</td>
<td>3F650</td>
</tr>
<tr>
<td>Magnetic Tape (TM11)</td>
<td>773126</td>
<td>3F65E</td>
</tr>
<tr>
<td>Asynchronous Line Adapters (DC11-DA, 4 registers each)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#1</td>
<td>774000</td>
<td>3F800</td>
</tr>
<tr>
<td>#2</td>
<td>774010</td>
<td>3F808</td>
</tr>
<tr>
<td>#3</td>
<td>774020</td>
<td>3F810</td>
</tr>
<tr>
<td>#4</td>
<td>774030</td>
<td>3F818</td>
</tr>
<tr>
<td>Synchronous Interface (DP11-DA, 6 registers/8 bytes) BR90</td>
<td>774770</td>
<td>3F9F8</td>
</tr>
<tr>
<td>Line Printer (LS11) Secure Mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td>775564</td>
<td>3FB74</td>
</tr>
<tr>
<td>Data Buffer</td>
<td>775566</td>
<td>3FB76</td>
</tr>
<tr>
<td>Telterm #1 (4 registers)</td>
<td>776160</td>
<td>3FC70</td>
</tr>
<tr>
<td>Telterm #2 (4 registers) (BR-90 Lab)</td>
<td>776560</td>
<td>3FD70</td>
</tr>
<tr>
<td>Definition</td>
<td>Octal</td>
<td>Hex</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td><strong>Card Reader (CR11)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td>777160</td>
<td>3FE70</td>
</tr>
<tr>
<td>Data Buffer</td>
<td>777162</td>
<td>3FE72</td>
</tr>
<tr>
<td>Data Buffer</td>
<td>777164</td>
<td>3FE74</td>
</tr>
<tr>
<td><strong>DEC Tape (TC11)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control and Status</td>
<td>777340</td>
<td>3FEE0</td>
</tr>
<tr>
<td>Command</td>
<td>777342</td>
<td>3FEE2</td>
</tr>
<tr>
<td>Word Count</td>
<td>777344</td>
<td>3FEE4</td>
</tr>
<tr>
<td>Bus Address</td>
<td>777346</td>
<td>3FEE6</td>
</tr>
<tr>
<td>Data Register</td>
<td>777350</td>
<td>3FEE8</td>
</tr>
<tr>
<td><strong>Disk (RF11)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control Status</td>
<td>777460</td>
<td>3FF30</td>
</tr>
<tr>
<td>Word Count</td>
<td>777462</td>
<td>3FF32</td>
</tr>
<tr>
<td>Current Memory Address</td>
<td>777464</td>
<td>3FF34</td>
</tr>
<tr>
<td>Disk Address</td>
<td>777466</td>
<td>3FF36</td>
</tr>
<tr>
<td>Disk Error Status</td>
<td>777470</td>
<td>3FF38</td>
</tr>
<tr>
<td>Disk Data Buffer</td>
<td>777472</td>
<td>3FF3A</td>
</tr>
<tr>
<td>Maintenance</td>
<td>777474</td>
<td>3FF3C</td>
</tr>
<tr>
<td>Address of Disk Segment</td>
<td>777476</td>
<td>3FF3E</td>
</tr>
<tr>
<td><strong>Line Printer (LS11) Normal Mode</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td>777514</td>
<td>3FF4C</td>
</tr>
<tr>
<td>Data Buffer</td>
<td>777516</td>
<td>3FF4E</td>
</tr>
<tr>
<td><strong>DEC Writer (4 registers)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Console Switch Register</td>
<td>777570</td>
<td>3FF78</td>
</tr>
<tr>
<td>11/45 SSRO</td>
<td>777572</td>
<td>3FF7A</td>
</tr>
<tr>
<td>11/45 SSR1</td>
<td>777574</td>
<td>3FF7C</td>
</tr>
<tr>
<td>11/45 SSR2</td>
<td>777576</td>
<td>3FF7E</td>
</tr>
<tr>
<td><strong>User Segmentation Registers</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I Space Descriptor (0–7)</td>
<td>777600</td>
<td>3FF80</td>
</tr>
<tr>
<td>D Space Descriptor (0–7)</td>
<td>777620</td>
<td>3FF90</td>
</tr>
<tr>
<td>Definition</td>
<td>Octal</td>
<td>Hex</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>I Space Address (0-7)</td>
<td>777640</td>
<td>3FFA0</td>
</tr>
<tr>
<td>D Space Address (0-7)</td>
<td>777660</td>
<td>3FFB0</td>
</tr>
<tr>
<td>11/45 PIRQ Register</td>
<td>777772</td>
<td>3FFFFA</td>
</tr>
<tr>
<td>11/45 Stack Limiter Register</td>
<td>777774</td>
<td>3FFFC</td>
</tr>
<tr>
<td>CPU Status (PSW)</td>
<td>777776</td>
<td>3FFFFE</td>
</tr>
</tbody>
</table>
ADDENDA

FLOW CHARTS
A FREE ASTE# EXISTS

NO ASTE# ON FREE CHAIN: LOOK ON AGE CHAIN FOR "OLDEST" ASTE#

I = Φ

NEXT = AST.Age-CHAIN (I)

I = NEXT

AST.Age-CHAIN (NEXT) = Φ

CALL DEACT

AST ENTRY IS DEACTIVATED: A FREE AST ENTRY EXISTS

ASTE# = AST.Chain (Φ)

REMOVE FROM FREE CHAIN
AST.Chain (Φ) = AST.Chain (ASTE#)

PUT ON AGE CHAIN
AST.Age-CHAIN (ASTE#) = AST.Age-CHAIN (Φ)
AST.Age-CHAIN (Φ) = ASTE#

CALL PREHASH

HASH VALUE ASSOCIATED WITH SEGMENT'S DISK ADDRESS CALCULATED
ADD ENTRY TO HEAD OF CHAIN OF ASTE'S WHOSE DISK ADDRESSES MAP ONTO THE SAME HASH-VAL

\[
\text{ASTE\_CHAIN (ASTE\#)} = \text{HASH\_TABLE (HASH\_VAL)} \equiv \text{HASH\_TABLE (HASH\_VAL)} = \text{ASTE\#}
\]

CLEAN UP AST ENTRY

\[
\text{ASTE\_ADR (ASTE\#)} = 0
\]

\[
\text{ASTE\_DES\_COUNT (ASTE\#)} = 0
\]

\[
\text{ASTE\_CPL (ASTE\#)} = 0
\]

FILL IN AST ENTRY FROM DIRECTORY

\[
\text{ASTE\_CLASS (ASTE\#)} = \text{DIR\_CLASS (OFFSET)}
\]

\[
\text{ASTE\_CAT (ASTE\#)} = \text{DIR\_DISK (OFFSET)}
\]

\[
\text{ASTE\_DISK (ASTE\#)} = \text{DIR\_DISK (OFFSET)}
\]

\[
\text{ASTE\_SIZE (ASTE\#)} = \text{DIR\_SIZE (OFFSET)}
\]

DIRECTORY WRITTEN INTO SET CHANGE BIT

RESET TO INITIALIZED

\[
\text{DIRECTORY STATUS} = \begin{cases} 
\text{INITIALIZED} & \text{Y} \\
\text{UNINITIALIZED} & \text{N}
\end{cases}
\]

INITIALIZE SEGMENTS SEMAPHORE

\[
\text{SMFR\_COUNT (ASTE\#)} = 1
\]

\[
\text{SMFR\_POINTER (ASTE\#)} = 0
\]

RETURN ASTE\#
RETURN ERROR

CLASS < AST_CLASS (ASTE #) & AST CLASS MASK

RETURN ERROR

CAT ≠ AST CAT (ASTE #)  CAT

PERFORM STATE CHANGE

DIR CLASS (OFFSET) = DIR CLASS (OFFSET) & DIR CLASS MASK \ CLASS
DIR CAT (OFFSET) = CAT

OASTE ≠ 0

SEGMENT ACTIVE
PERFORM STATE CHANGE

ASTE CLASS (OASTE #) = DIR CLASS (OFFSET)
ASTE CAT (OASTE #) = CAT

RETURN OK
CONNECT

FIND FREE SEGMENT
SEG # = (PS SEG (0) & SEG MASK)

RETURN ERROR
NO FREE SEGMENT #

CALL HASH

ASTE # = HASH OF DISK ADDRESS

RETURN ERROR
SEGMENT ALREADY CONNECTED

ASTE CPL (ASTE #) & PS-PROCESS-MASK ≠ Ø

ASE # ≠ Ø

CALL ACT

SEGMENT ACTIVATED

ASTE CPL (ASTE #)

SEARCH AGE CHAIN
INDEX = 0

NEXT = ASTE AGE-CHAIN (INDEX)

INDEX = NEXT

B

A

276
REMOVE FROM AGE CHAIN
AST_AGE_CHAIN (INDEX) =
AST_AGE_CHAIN (ASTE #) = AST_AGE_CHAIN (ASTE #) = 0

CONNECT PROCESS
AST_CPL (ASTE) = AST_CPL | PS_PROCESS_MASK

ADD TO WAL
AST_WAL (ASTE #) = AST_WAL (ASTE #) | PS_PROCESS_MASK

MODE:
READ & WRITE
EXECUTE_ACCESS

REMOVE SEG # FROM FREE_CHAIN
PS_SEG (0) = PS_SEG (SEG #)
PS_SEG (SEG #) = ASTE #

RETURN SEG #
NEW SEG CLASS < PARENT SEG'S CLASS

RETURN ERROR

DIRECTORY IS ON PROCESS'S WAL

CLASS = AST_CLASS

NEW SEG CAT DOES NOT INCLUDE PARENT SEG'S CAT

SIZE NOT WITHIN LIMITS

OFFSET NOT A FREE DIRECTORY ENTRY

WRITE DIRECTORY ENTRIES FOR NEW SEG
DIR_TYPE = SEG_TYPE
UNINITIALIZED 1 CLASS
DIR_CAT = CAT
DIR_DISK = DISK_ADR
DIR_SIZE = SIZE

RETURN OK
DCONNECT

BLOCK# = AST_ADDR (ASTE#)

\[ \text{Y} \]

\( \text{BLOCK} \neq 0 \) & AST_UNLOCK (ASTE#) & AST_UNLOCK_MASK = UNLOCK_FLAG

SEGMENT IS IN ACTIVE SPACE OF PROCESS; DISABLE IT

REG# = 0 TO REG# MAX

\[ \text{N} \]

REG# + 1

PS_SAR (REG = BLOCK)

\[ \text{Y} \]

CALL DISABLE

DECONNECT PROCESS

AST_CPL (ASTE#) = AST_CPL (ASTE) & PS_PROCESS_NOT_MASK

AST_WAL (ASTE#) = AST_WAL (ASTE#) & PS_PROCESS_NOT_MASK

\[ \text{N} \]

AST_WAL (ASTE#) = 0 & AST_CPL (ASTE#) & WIRED_DOWN MASK ≠ WIRED_DOWN

FREE PROCESSES BLOCKED ON SEGMENT SEMAPHORE

\[ \text{V} \]

SMFR_COUNT (ASTE) >= 0

\[ \text{Y} \]

A
SMFR.COUNT (ASTE #) = 1
SMFR POINTER (ASTE #) = 0

AST.CPL (ASTE #) = 0

N

Y

ADD TO AGE CHAIN
AST.AGE.CHAIN (ASTE #) =
AST.AGE.CHAIN (0) = AST.AGE.CHAIN (0)
AST.AGE.CHAIN (0) = ASTE 

PUT ON CHAIN OF FREE SEGMENTS
PS_SEG (SEG #) = PS_SEG (0)
PS_SEG (0) = SEG # + 1 SEG_FLAG

RETURN
DEACT

INDEX = 0

FIND SEGMENT ON AGE CHAIN

NEXT = AST_AGE_CHAIN (INDEX)

INDEX = NEXT

NEXT = ASTE #

LINK AST BEFORE ASTE # TO AST AFTER ASTE #

AST_AGE_CHAIN (INDEX) =
AST_AGE_CHAIN (ASTE #)
AST_AGE_CHAIN (ASTE #) = 0

CALL SWAPIN

Y

AST_STATUS (ASTE #)

< AST_STATUS MASK

= AST_UNINITIALIZED

N

SEGMENT INITIALIZED

CALL SWAPOUT

N

AST_ADDR (ASTE #) = 0

Y

SEGMENT NOT IN MAIN MEMORY

REMOVE FROM HASH TABLE OR HASH CHAIN
CALL PREHASH

HASH_VAL ASSOCIATED WITH SEGMENTS ASTE # FOUND

REMOVE IT HASH_TABLE (HASH_VAL) = AST. CHAIN (ASTE #)

AST IS HEAD OF CHAIN OF ASTE'S

HASH_TABLE (HASH_VAL) = ASTE #

SEARCH CHAIN HASH_VAL = HASH_TABLE (HASH_VAL)

NEXT = AST_CHAIN (HASH VAL)

HASH_VAL = NEXT

ADD TO FREE CHAIN
AST_CHAIN (ASTE #) = AST_CHAIN (0) = ASTE #

RETURN
DELETE
CALL WRITEDIR
RETURN ERR_FLAG
NO WRITE ACCESS
Y
ERR_FLAG
RETURN ERR_FLAG
NO SUCH ENTRY
Y
DIR_SIZE (OFFSET) = 0
N
DIR_TYPE (OFFSET) & DIR_TYPE_MASK ≠ DIR_TYPE_DIRECTORY
N
ENTRY IS A DIRECTORY SEGMENT EMPTY IT
ENTRY IS A DATA SEGMENT
Y
DELETSEG
SPASTE # ASTE #
OFFSET = OFFSET
CALL HASH
ASTE # = HASH OF DISH ADDRESS
CALL ACT
DIRECTORY INACTIVE
Y
SASTE ≠ 0
N
CALL GETDIR
SEGMENT DESCRIPTORS LOADED, SEARCH DIRECTORY FOR NON_ZERO ENTRIES
A
B C
DO TOFFSET = OFFSET_MIN TO OFFSET_MAX

DIR_SIZE (TOFFSET) ≠ 0

Y

DELETE ALL DIRECTORY ENTRIES

DELETE THIS SEGMENT

CALL DELETSEG

LAST ENTRY? TOFFSET = MAX?

N

OFFSET = OFFSET + 1

N

IS THIS A DIRECTORY?

Y

DELETE THIS SEGMENT

CALL DELETSEG

DIRECTORY EMPTY DELETE IT

CALL GETDIR

CALL DELETSEG

SPASTE # = ASTE #

RETURN OK
INDEX = ACL.CHAIN (INDEX)

REMOVE ELEMENTS ON ACL CHAIN

INDEX = ACL_CHAIN (INDEX)

PUT. ON HEAD OF FREE CHAIN
ACL_CHAIN (INDEX) = ACL_CHAIN (0)
ACL_CHAIN (0) = DIR_ACL.HEAD (OFFSET)
DIR_ACL.HEAD (OFFSET) = 0

CALL SOADD

SEGMENT REMOVED FROM WORKING SPACE OF ALL PROCESSES; IF ACTIVE DEACTIVATE

CALL HASH

OASTE # = HASH OF DISK ADDRESS

CLEAR CHANGE AND STATUS BIT
AST_CHANGE (OASTE #) = AST_CHANGE (OASTE #) & AST_UNCHANGED. MASK
AST_STATUS (OASTE #) = AST_STATUS (OASTE #) & AST_STATUS_NOT_MASK

DELETE SEG

INDEX = DIR_ACL.HEAD (OFFSET)

NO ELEMENTS ON ACL CHAIN

INDEX ≠ @

Y

REM OVE ELEMENTS ON ACL CHAIN

INDEX = ACL_CHAIN (INDEX)

ACL_CHAIN (INDEX) = ACL_CHAIN (0)
ACL_CHAIN (0) = DIR_ACL.HEAD (OFFSET)
DIR_ACL.HEAD (OFFSET) = 0

CALL SOADD

SEGMENT REMOVED FROM WORKING SPACE OF ALL PROCESSES; IF ACTIVE DEACTIVATE

CALL HASH

OASTE # = HASH OF DISK ADDRESS

CLEAR CHANGE AND STATUS BIT
AST_CHANGE (OASTE #) = AST_CHANGE (OASTE #) & AST_UNCHANGED. MASK
AST_STATUS (OASTE #) = AST_STATUS (OASTE #) & AST_STATUS_NOT_MASK

A
DEACT

SEGMENT DEACTIVATED

CALL DFREE

DISK SPACE FREED

MARK DIRECTORY ENTRY FREE
\[ \text{DIR\_DISK (OFFSET)} = \emptyset \]
\[ \text{DIR\_SIZE (OFFSET)} = \emptyset \]

SET CHANGE BIT IN PARENT DIRECTORY
\[ \text{AST\_CHANGE} (\text{ASTE} \#) = \text{AST\_CHANGE} (\text{ASTE} \# \uparrow \text{AST\_CHANGED}) \]

RETURN
DISABLE

\[ \text{BLOCK\#} = \text{PS\_SAR (REG\#)} \]

\[ \text{Y} \]

\[ \text{BLOCK\#} = \emptyset \]

\[ \text{N} \]

EXECUTE 2 ARITHMETIC SHIFT RIGHT COMMANDS TO OBTAIN STORAGE ADDRESS OF SEGMENT

\[ \text{ASTE\#} = \text{MBT (BLOCK\#)} \]

\[ \text{Y} \]

\[ \text{BLOCK\#} < \text{END\_BLOCK} \]

\[ \text{N} \]

AST MIN TO AST MAX

\[ \text{AST\_ADR (ASTE\#)} = \text{ASTE\#} \]

\[ \text{ASH (BLOCK\#)} \]

\[ \text{Y} \]

SET PRIORITY LEVEL HIGH

DESTROY DESCRIPTORS

\[ \text{PS\_SDR (REG\#)} = 0 \]

\[ \text{PS\_SAR (REG\#)} = 0 \]

\[ \text{PS\_CURRENT\_PROCESS} = \text{THE\_CURRENT\_PROCESS} \]

\[ \text{REG\#} = \text{REG\#} + \text{REG\_CONSTANT} \]

C

B

A

287
CHANGE BIT SET
AST_CHANGE (ASTE#) = AST_CHANGE (ASTE#) + AST_CHANGED

CLEAR SEGMENTATION REGISTERS
SDR (REG#) = 0
SAR (REG#) = 0

SET PRIORITY LEVEL LOW

DECREMENT DESCRIPTOR COUNT
AST_DES_COUNT (ASTE#) = AST_DES_COUNT (ASTE#) - 1

NOT ELIGIBLE FOR SWAPPING OUT
AST_CPL (ASTE#) & WIRED_DOWN
MASK = 0

DESCRIPTORS REMAIN
AST_DES_COUNT (ASTE#) = 0

NO DESCRIPTORS LEFT: ADD TO HEAD OF SWAP CHAIN
AST_SWAP_CHAIN (ASTE#) = AST_SWAP_CHAIN(0)
AST_SWAP_CHAIN(0) = ASTE#
UNLOCK
AST_UNLOCK (ASTE #) =
AST_UNLOCK (ASTE #) | 
AST_UNLOCK_FLAG

ADJUST MEMORY QUOTA
PS_MEM_QUOTA =
PS_MEM_QUOTA +
AST_SIZE_ASTE #

RETURN
ELEMENT FOUND; TEST FOR CORRECT MODE

RETURN ERROR Y

ACL_MODE INDEX) 
& ACL_MODE_MASK 
= NO_ACCESS

RETURN ERROR Y

(MODE= WRITE & READ 
& EXECUTE_ACCESS) 
& 
(ACL_MODE (INDEX)) 
& 
ACL_MODE_MASK ≠
WRITE & READ & EXECUTE_ACCESS

RETURN OK
P_REG^ = REG# + REG_CONSTANT

P_REG^ = REG#

REG# > CROSS_REG #

PS_SAR (REG#)

RETURN "V

REGISTER NOT FREE

REGISTER FREE

RETURN ERROR

NO SPACE IN USERS MEMORY

SPACE AVAILABLE

CALL SWAPIN

SEGMENT NOT ON SWAP CHAIN

AST_ADDR (ASTE #)

AST_UNLOCK (ASTE #)

FIND SEGMENT ON SWAP CHAIN

INDEX = 0

AST_SIZE (ASTE #) <= PS_MEM_QUOTA

= WIRED_DOWN MASK

= WIRED_DOWN

index

SEGMENT NOT ON SWAP CHAIN

INDEX = 0

B

A
NEXT = AST_SWAP_CHAIN (INDEX)

INDEX = NEXT

NEXT = ASTE #

REMOVE FROM SWAP CHAIN.
AST_SWAP_CHAIN (INDEX) =
AST_SWAP_CHAIN (ASTE #)
AST_DES_COUNT (ASTE) = 0
AST_UNLOCK (ASTE #) &
AST_UNLOCK (ASTE #) &
AST_LOCK_MASK

DETERMINE THE MODE OF ACCESS PERMITTED

MODE = 0

MODE = SDR_WRITE_ACCESS

MODE = SDR_READ_ACCESS

SET PRIORITY LEVEL HIGH

CALL LSD

DESCRIPTORS LOADED IN
PS.CURRENT.PROCESS = THE.CURRENT.PROCESS

Y

LOAD HARDWARE REGISTERS

SDR (P_REG #) = PS_SDR (REG #)
SAR (P_REG #) = PS_SAR (REG #)

SET PRIORITY LEVEL LOW

INCREMENT DESCRIPTORS
AST_DES_COUNT (ASTE #) =
AST_DES_COUNT (ASTE #) + 1

N

SEGMENT IS NOT IN USERS MEMORY SPACE

AST_CPL (ASTE #) &
WIRED_DOWN MASK =
WIRED_DOWN

Y

ADJUST USERS MEMORY QUOTA
PS_MEM_QUOTA = PS_MEM QUOTA
MINUS AST_SIZE (ASTE #)

RETURN OK

N
GATE
TRAP
KERNEL_ENTRY
GENERAL REGISTERS SAVED ON STACK

N
PREVIOUS
MODE=
SUPERVISOR

Y
KS\text{DR}3=SDRO
KS\text{AR}3=SARO
CALL PCHECK
KS\text{DR}3=SDRO
KS\text{AR}3=SARO
PLACE RETURN CODE ON SUPERVISOR STACK
KRC=RC
KERNEL_EXIT
GENERAL REGISTERS RESTORED
RETURN
CALL SWAP N

SEGMENT NOT IN MAIN MEMORY

SEGMENT SWAPPED IN

GETDIR

AST. ADR (ASTE #) = 0 ?

CALL LSD

SEG DESCRIPTORS IN DIR.KSR.ADR ACCESS SDR.WRITE.ACCESS

RETURN
GETR

CALL DSEARCH

USER HAS READ & EXECUTE ACCESS

RETURN ERROR

PROPERTY NOT PRESERVED

PS_CUR_CLASS < (DIR_CLASS (OFFSET) & DIR_CLASS_MASK)

PROPERTY NOT PRESERVED

PS_CUR_CAT ≠ (DIR_CAT (OFFSET))

CALL CONNECT

PROCESS CONNECTED TO CPL OF THE ASTE

RETURN SEG #
GETW
CALL DSEARCH
USER HAS WRITE & READ B EXECUTE ACCESS TO THE ACL
RETURN

TRUSTED PROCESS * PROPERTY NEED NOT BE PRESERVED ONLY SECURITY

RETURN ERROR

CALL CONNECT
PROCESS CONNECTED TO CPL AND WAL OF THE ASTE
RETURN SEG *
GIVE
CALL WRITEDIR

RETURN ERROR
SEGMENT DOES NOT EXIST

DIR_DISK FOR NEW SEG
= Ø

SEARCH ACL_CHAIN FROM DIR_ACL_HEAD TO END FOR ELEMENT =ACL_USER & ACL_PROJECT

RETURN ERROR
DUPLICATE ELEMENT FOUND

ALLOCATE ACLE # = ACL_CHAIN (Ø)

RETURN ERROR
NO FREE ELEMENTS

ICLE # = 0

RESET ACL_CHAIN (Ø) =ACL_CHAIN (ACLE #)

FIND CORRECT POSITION FOR NEW ELEMENT;
POSITION = Ø

CHAIN EMPTY POSITION = Ø

DIR_ACL_HEAD = Ø

B

A

299
ENTER ELEMENT AT BEGINNING OF LIST:
ACL_CHAIN (ACLE #)
DIR_ACL_HEAD (OFFSET)
DIR_AFL_HEAD (OFFSET) =ACLE

Y
POSITION = Ø

N

INSERT ELEMENT AFTER ACL ELEMENT IDENTIFIED BY POSITION:
ACL_CHAIN (ACLE #)
ACL_CHAIN (POSITION)
ACL_CHAIN (POSITION) =ACLE #

MODE=
WRITE & READ & EXECUTE

CALL SOADD

RETURN OK
INITH

RETURN

N

ERROR

Y

CALLED BY

TRUSTED

PROCESS?

CALL WRITEDIR

RETURN

N

ERROR

OK_FLAG

Y

RETURN

N

ERROR

DIR_SIZE

(OFFSET)=0

COPY AST ATTRIBUTES

INTO DIRECTORY

DIR_CLASS

(OFFSET)=AST_CLASS

(ASTE #)

DIR_CAT

(OFFSET)=AST_CAT

(ASTE #)

DIR_DISK

(OFFSET)=AST_DISK

(ASTE #)

DIR_SIZE

(OFFSET)=AST_SIZE

(ASTE #)

DIR_ACL_HEAD

(OFFSET)=0

RETURN OK
INITSEG

AST_ADR (ASTE #) = BLOCK #

CALL LSD

DESCRIPTORS LOADED

PERFORM INITIALIZATION ASSUMING ALL SEGMENTS ARE 1K BYTES
FOR I = 0 TO 511
ZERO ARRAY (I) = 0

DATA SEGMENT: SET TO ALL ZEROS
DIRECTORY SEGMENT: ALL ENTRIES MARKED FREE

AST_TYPE (ASTE #) &
AST_TYPE
MASK =
AST_TYPE.DIRECTORY

PUT ALL ACLES ON FREE CHAIN: FOR I = 0 TO ACL_MAX

ACL_CHAIN (I) = I + 1

ACL_CHAIN (ACL_MAX) = 0

UPDATE AST
AST_CHANGE (ASTE #) = AST_STATUS
(ASTE #) & AST_STATUS.NOT MASK
AST_CHANGE (ASTE #) = AST_CHANGE
(ASTE #) & AST_CHANGED

RETURN
NOTE: ALL INTERRUPT HANDLERS LISTED IN GATE WORK THE SAME WAY.

1. INTERRUPT
2. KERNEL ENTRY
3. V THE DEVICE SEMAPHORE TO NOTIFY THE WAITING PROCESS
4. CALL V
5. KERNEL EXIT
6. RETURN
IPCRCV

ANYTHING THERE?
PT_IPC_QUEUE_HEAD (PS_CURRENT PROCESS) = 0

NO MESSAGES
PT_IPC_QUEUE_HEAD (PS_CURRENT_PROCESS) = IPC_WAIT
PT_FLAGS (PS_CURRENT_PROCESS) = BLOCKED

CALL V
CALL SLEEP
CALL P

INDEX = PT_IPC_QUEUE_HEAD (PS_CURRENT_PROCESS)
PT_IPC_QUEUE_HEAD (PS_CURRENT_PROCESS) = IPC_LINK (INDEX)

RETURN INFORMATION
RC = IPC.PROCESS# (INDEX)
KRC2 = IPC.DATA (INDEX)

PUT ELEMENT ON FREE CHAIN AND INCREMENT QUOTA
IPC_LINK (INDEX) = IPC_LINK (0)
IPC_LINK (0) = INDEX
PT_IPC_QUOTA (PS_CURRENT_PROCESS) = PT_IPC_QUOTA (PS_CURRENT_PROCESS) + 1

RETURN
RETURN

TRUSTED PROCESS

RETURN

NO FREE ELEMENTS

RECEIVING PROCESS HAS FREE ELEMENTS ALLOCATE ONE

INDEX=IPC_LINK (0)
IPC_LINK (0) = IPC_LINK (INDEX)
IPC_LINK (INDEX) = 0

FILL IN ALLOCATED ELEMENT
IPC_PROCESS # (INDEX) = PS_CURRENT_PROCESS.DOMAIN
IPC_DATA (INDEX) = MESSAGE

PROCESS IS BLOCKED WAITING A MESSAGE UNBLOCK IT

PT_IPC_QUEUE_HEAD (PROCESS #) = INDEX
PT_FLAGS (PROCESS #) = READY

PT_IPC_QUEUE_HEAD (PROCESS #) = INDEX
PROCESS'S QUEUE HEAD EMPTY

APPEND NEW MESSAGE TO PROCESS' QUEUE

PT_IPC_QUEUE_HEAD (PROCESS #) = 0

FIND END OF THE QUEUE AND ADD ELEMENT

INDEX 2 = PT_IPC_QUEUE_HEAD (PROCESS #)

INDEX 2 = IPC_LINK (INDEX 2)

ADJUST RECEIVING PROCESS'S QUOTA

PT_IPC_QUOTA (PROCESS #) = PT_IPC_QUOTA (PROCESS #) - C

RETURN
OUTERP

RETURN ERROR

NO WRITE ACCESS Y

AST_WAL(ASTE #) & PS_PROCESS_MASK = 0

N

SET PRIORITY LEVEL HIGH

RETURN ERROR

OUT_OF_BOUNDS Y

SMFR_COUNT(ASTE #) = MINUS 128

N

CALL P

RETURN OK
RETURN ERROR

NO WRITE ACCESS

AST_WAL(ASTE #)
& PS_PROCESS_MASK = 0

SET PRIORITY LEVEL HIGH

OUT OF BOUNDS

SMFR_COUNT(ASTE #) = 127

CALL V

RETURN OK
P

SET PRIORITY LEVEL HIGH

SMFR_COUNT (SMFR #) =
SMFR_COUNT (SMFR #) MINUS 1

N

SMFR_COUNT (SMFR #) < \emptyset

Y

ADD TO QUEUE OF PROCESSES BLOCKED ON SEMAPHORE

PT_FLAGS (THE_CURRENT_PROCESS) = SMFR
POINTER (SMFR #) BLOCKED
SMFR_POINTER (SMFR #) = THE_CURRENT_PROCESS

START NEW PROCESS RUNNING

CALL SLEEP

N

SMFR #< KERNEL SMFR

Y

CALL V

V PERFORMED ON KERNEL_SMFR

CALL SLEEP

THIS PROCESS CONTINUES

CALL P

P PERFORMED ON KERNEL_SMFR

SET PRIORITY LEVEL LOW

RETURN

\{ THIS LETS OTHER PROCESSES EXECUTE WITHIN THE KERNEL \}

\{ THIS ALLOWS OTHER PROCESSES TO RUN \}

\{ THIS LOCKS UP THE KERNEL AGAIN \}
FUNCTION_CODE = FUNCTION_CODE_APARM

CALL P

P PERFORMED ON KERNEL_SMF

PARM_FLAGS = FUNCTION_ARRAY(FUNCTION_CODE)

CHECK REQUIRED PARAMETERS

PARM_FLAGS & SEG #_FLAG ≠ 0

SEG_PARM = SEG_APARM

RETURN SEVERE ERROR

SEG_PARM < SEG #_MIN OR SEG_PARM > SEG #_MAX

ASTE #_PARM = PS_SEG(SEG #_PARM)

RETURN SEVERE ERROR

ASTE #_PARM & SEG_FLAG ≠ 0

SEGMENT IN WS OF PROCESS
PARM_FLAGS.

OFFSET_FLAG

OFFSET

PARM< RETURN SEVERE ERROR

OFFSET_MIN

OFFSET_MAX

OFFSET PARM WITHIN BOUNDS

CLASSIFICATION PARM WITHIN BOUNDS

NO CHECKING REQUIRED FOR CAT, SEG_TYPE OR SIZE
SET VALUES IF REQUIRED

CLASS_PARM=CLASS_APARM

SEG_TYPE_PARM=SEG_TYPE_APARM & DIR_TYPE_MASK

SIZE_PARM=SIZE_APARM

B
PARM_FLAGS

MODE_FLAG ≠ Ø

MODE_PARM = MODE_APARM

RETURN SEVERE ERROR

MODE_PARM ≠ (WRITE & READ & EXECUTE_ACCESS)

MODE_PARM ≠ (READ & EXECUTE_ACCESS)

MODE_PARM ≠ NO_ACCESS

MODE_PARM WITHIN BOUNDS

NO CHECKING OF USER AND PROJECT PARMS: SET VALUES IF REQUIRED

USER_PARM = USER APARM & ACL

USER MASK

PROJECT_PARM = PROJECT APARM

N PARM_FLAGS & REG_FLAGS

REG PARM ≠ REG APARM

RETURN SEVERE ERROR

REG ≠ REG APARM

REG_MIN OR REG APARM

REG MAX

REG APARM WITHIN BOUNDS
C

N

PARM_FLAGS ≠ PROCESS_FLAGS

Y

PROCESS #_PARM > PROCESS #_APARM

RETURN SEVERE ERROR

Y

PROCESS #_PARM < PROCESS #_MIN OR PROCESS #_PARM > PROCESS #_MAX

N

PROCESS #_PARM WITHIN BOUNDS

NO CHECKING OF MESSAGE PARM
SET VALUE IF REQUIRED

MESSAGE_PARM = MESSAGE_APARM

CALL FUNCTION GIVEN BY FUNCTION_CODE

CALL V

V PERFORMED ON KERNEL_SMFR

RETURN RV
ERROR
SEGMENT NOT A DIRECTORY

CALL
SWAPIN

DIRECTORY SEGMENT IN MAIN MEMORY

CALL
LSD

SEGMENT DESCRIPTORS LOADED
ACCESS MODE = READ ACCESS

SAVE SEGMENT ATTRIBUTES
CLASS = DIR_CLASS (OFFSET)
CAT = DIR_CAT (OFFSET)
SEG_TYPE = DIR_TYPE (OFFSET)
SIZE = DIR_SIZE (OFFSET)

REGAIN ACCESS TO USER
STACK STATUS REGISTER
INSERT DATA THAT HAS THE SAME SECURITY LEVEL AS THE DIRECTORY

RETURN OK
RESCIND
CALL WRITEDIR

RETURN ERROR
N
IS DIRECTORY ON PROCESSES WAL

SEARCH FOR SPECIFIED ACLE
INDEX = DIR_ACL_HEAD (OFFSET)

RETURN ERROR
N
INDEX = Ø
Y

USER = ACL_USER (INDEX)
& ACL_USER_MASK
& PROJECT = ACL_PROJECT (INDEX)

SPECIFIED ACLE FOUND
REMOVE FROM CHAIN

REMOVE ELEMENT FROM CHAIN
ACL_CHAIN (SAVE_LAST) =
ACL_CHAIN (INDEX)

REMOVE ELEMENT FROM CHAIN
DIR_ACL_HEAD (OFFSET) =
ACL_CHAIN (INDEX)

PUT ELEMENT AT HEAD OF CHAIN
ACL_CHAIN (INDEX) = ACL_CHAIN (Ø)
ACL_CHAIN (Ø) = INDEX

CALL SQADD

RETURN OK
RUN

SET PRIORITY LEVEL HIGH

LOOK FOR CHANGES
REG # = 0 TO REG_MAX

X = REG #

REG # >

X = REG # * REG_CONSTANT

CHANGE BIT NOT SET

SDR(X) B

SDR.CHANGE_-

Mask =

SBR.CHANGED

PERFORM 2 ARITHMETIC SHIFTS RIGHT
VAR = MAIN MEMORY ADDRESS

CHANGE BIT IN
ASTE NOT SET

VAR < END

BLOCK

VAR = MBT_ASTE.(VAR)

AST_CHANGE (VAR) * AST_CHANGE (VAR)

1 AST_CHANGED

SAVE KSR1

PT_KSAR1 (THE_CURRENT_PROCESS) = KSAR1

PT_KSDR1 (THE_CURRENT_PROCESS) = KSDR1

CALL LSD

SAVE KSR1

PT_KSAR1 (THE_CURRENT_PROCESS) = KSAR1

PT_KSDR1 (THE_CURRENT_PROCESS) = KSDR1

CALL LSD
DESCRIPTORS FOR NEXT PROCESS LOADED
CALL SWAP
CONTINUE THIS PROCESS
SET PRIORITY LEVEL LOW
RETURN
SLEEP

SET PRIORITY LEVEL LOW

NEXT PROCESS = THE_CURRENT_PROCESS + 1

NEXT_PROCESS > MAX

Y

NEXT_PROCESS = PROCESS # _MIN

N

PT_FLAGS(NEXT_PROCESS) & PT_FLAGS_.MASK = READY

Y

FOUND A READY PROCESS

RETURN

NEXT_PROCESS >= THE_CURRENT_PROCESS

N

CALL RUN

RETURN
NO PROCESSES HAVE DESCRIPTORS FOR THE SEGMENT

LOOP THRU ALL PROCESSES
PROCESS # = PROCESS # - MIN
TO PROCESS # - MAX

PROCESS INACTIVE
NO NEED TO CHECK FURTHER

DETERMINE MODE OF ACCESS

PLACE NOT CONNECTED TO SEGMENT

MODE = READ & EXECUTE_ACCESS

MODE = WRITE & READ & EXECUTE ACCESS
MODE OF ACCESS 
STILL PERMITTED

K

FIND SEG #
SEG# = SEG#_MIN TO SEG#_MAX

SEG# = SEG# + 1

PS_SEG (SEG #) = ASTE #

Y

DCONNECT

SEGMENT RELEASED FROM PROCESS'S WORKING SPACE

N

PROCESS # = PROCESS #_MAX

Y

CALL LSD

DESCRIPTORS FOR THE CURRENT_PROCESS RESTORED IN KSR2

RETURN
PROCESS IS TRUSTED AND IS FREE

CALL LSD

SEGMENT DESCRIPTORS OF NEW PROCESS PS LOADED

INITIALIZE PROCESS'S PS
PS_CURRENT_PROCESS = PROCESS #
PS_USER_ID = USER
PS_PROJECT_ID = PROJECT
PS_CUR_CLASS = CLASS
PT_CUR_CLASS (PROCESS #) = CLASS
PS_CUR_CAT = CAT
PT_CUR_CAT (PROCESS #) = CAT
PS_MEM_QUOTA = MEM_QUOTA
PT_IPC_QUOTA (PROCESS #) = IPC_QUOTA

INLINE CODE IS USED TO FIND PS_PROCESS_MASK AND PS_PROCESS_NOT_MASK

A
MOVE THE PROCESS # INTO TEMPORARY REGISTER # 3 (TEMP # 3)

DECREMENT AND TAKE THE NEGATIVE OF THE PROCESS # (TEMP # 3)

MOVE THE NUMBER "4000" INTO TEMP # 0

MOVE THE NUMBER "BFFF" INTO TEMP # 1

IF TEMP # POSITIVE LEFT SHIFT TEMP # 0, TEMP # 3 TIMES, OTHERWISE RIGHT SHIFT TEMP # 3 TIMES

LEFT OR RIGHT SHIFT TEMP # 1 TEMP # 3 TIMES IF TEMP # 3 POSITIVE OR NEGATIVE, RESPECTIVELY

PUT THE CONTENTS OF TEMP # 0 INTO PS_PROCESS_MASK

PUT THE CONTENTS OF TEMP # 1 INTO PS_PROCESS_NOT_MASK
ALL 15 PS_SAR'S AND PS_SDR'S = 0

ALL SEG # FROM 0 TO SEG_MAX PUT ON FREE CHAIN

PS_SEG (SEG #_MAX) = SEG_FLAG
MARKS END OF FREE CHAIN

CONNECT NEW PROCESS TO ROOT
PS_SEG (0) = PS_SEG (1)
PS_SEG (1) = ROOT ASTE #
AST CPL (ROOT ASTE #) =
AST CPL (ROOT ASTE #)
PS_PROCESS_MASK

GAIN ACCESS TO KERNEL AND USER STACKS

CALL GETR

READ ACCESS GAINED TO PROCESS DIRECTORY
PDD_SEG # = SEG # RETURNED

CALL GETR

READ ACCESS GAINED TO EXEC PROCESS DIRECTORY
PD_SEG # = SEG # RETURNED
GAIN ACCESS TO USERS STACK AND ENABLE IT

CALL GETW

WRITE ACCESS GAINED SS_SEG# = SEG# RETURNED

CALL ENABLE

NOW KERNEL STACK

CALL LSD

SEGMENT DESCRIPTORS OF PROCESS DIRECTORY LOADED

CALL HASH

KS_ASTE# = ASTE# RETURNED

HALT

FATAL ERROR THIS SHOULD NEVER HAPPEN

KS_ASTE# \neq 0

CALL LSD

SEGMENT DESCRIPTORS IDENTIFIED BY KS_ASTE# LOADED
AST_DES_COUNT (KS_ASTE#) =
AST_DES_COUNT (KS_ASTE#) + 1
PT_KS_ASTE# (PROCESS #) = KS_ASTE#

CALL DCONNECT

PROCESS DIRECTORY DIRECTORY AND
PROCESS DIRECTORY RELEASED FROM
WORKING SPACE

CALL GETR

READ ACCESS TO CODE DIRECTORY,
IDENTIFIED BY ROOT_ASTE# AND CD_<
OFFSET HAS BEEN GAINED CD SEG# =
SEG# RETURNED

GETR

READ ACCESS GAINED TO PROC DIRECTORY
PROC_SEG# = SEG# RETURNED

CALL ENABLE

SEGMENT IDENTIFIED BY PROC_SEG# PLACED IN WORKING SPACE

CALL DCONNECT

CD_SEG# RELEASED

E

326
CALL LSD

SEGMENT DESCRIPTORS OF EXEC PROCESS RELOADED

CALL LSD

SEGMENT DESCRIPTORS OF NEW SEGMENT LOADED

EXEC NOW HAS WRITE, WRITE ACCESS TO NEW PROCESS'S PROCESS SEGMENT

PT_R5 (PROCESS #) = 0
PT_FLAGS (PROCESS #) = READY
PT_IPC_QUEUE_HEAD = 0

INITIALIZATION OF NEW PROCESS COMPLETE

RETURN OK
STOPP

DO I=SEG #_MIN TO SEG #_MAX

ASTE #=PS_SEG(I)

N  ASTE # & SEG_FLAG = ∅

Y  CALL DCONNECT

SEGMENTS TO WHICH PROCESS HAS ACCESS ARE RELEASED

N  I=SEG #_MAX

Y  CLEAR OUT IPC QUEUE

I=PT_IPC_QUEUE_HEAD (PS_CURRENT_PROCESS)

N  I ≠ ∅

I=IPC_LINK(I)

Y

N  IPC_LINK(I) ≠ ∅

Y  INSERT QUEUE AT HEAD OF FREE CHAIN

A

328
The diagram illustrates the process of removing a process's kernel stack and informing the executive process of the current process's termination. The steps are as follows:

1. ** IPC_LINK(I) = IPC_LINK(0) **
2. ** IPC_LINK(0) = PT_IPC_QUEUE_HEAD **
3. ** PT_IPC_QUEUE_HEAD = (PS_CURRENT_PROCESS) = 0 **

** REMOVE PROCESS'S KERNEL STACK **

- ** ASTE # = PT_KS ASTE # (PS_CURRENT_PROCESS) **
- ** AST_SWAP_CHAIN (ASTE #) = AST_SWAP_CHAIN (0) **
- ** AST_SWAP_CHAIN (0) = ASTE # **
- ** AST_UNLOCK (ASTE) = AST_UNLOCK (ASTE #) **
- ** AST_UNLOCK_FLAG **

** INFORM EXEC PROCESS OF CURRENT PROCESS'S TERMINATION **

- ** CALL IPC SEND **

** PT_FLAGS (PS_CURRENT_PROCESS) = INACTIVE **

- ** CALL V **

** V PERFORMED ON KERNEL_SMFR **

- ** CALL SLEEP **

** PROCESSOR ALLOCATED TO NEXT READY PROCESS **

- ** EXIT **
CALL INITSEG

SWAPIN

LOOK FOR FREE BLOCK
BLOCK # = MBT_CHAIN (0) & MBT_CHAIN_MASK

FREE BLOCK EXISTS

BLOCK #
END_BLOCK

Y

NONE FREE LOOK ON SWAP
CHAIN: I = 0

NEXT = AST_SWAP_CHAIN (0)

I = NEXT

N

AST_SWAP_CHAIN(NEXT) ≠ 0

Y

BLOCK # = AST_ADR_(NEXT)

CALL SWAPOUT

NEXT REMOVED FROM SWAP CHAIN
AND PUT ON FREE CHAIN

REMOVE BLOCK FROM FREE CHAIN AND
MARK IT ALLOCATED
MBT_CHAIN(0) = MBT_CHAIN(BLOCK #)
MBT_FLAGS(BLOCK #) = ALLOCATED

CALL INITSEG

INITIALIZE IT

Y

AST_STATUS(ASTE #)
AST_STATUS MASK
AST_UNINITIALIZED

N

A
CALL DISKIO

P IS PERFORMED ON DISK
SMFR WHILE SEGMENT IS
READ FROM DISK

UPDATE AST
AST_ADR (ASTE#) = BLOCK #
MBT: ASTE (BLOCK #) = ASTE#
I ALLOCATED

NO DESCRIPTORS FOR
SEGMENT: PUT IT AT HEAD
OF SWAP CHAIN
AST_SWAP_CHAIN (ASTE#) =
AST_SWAP_CHAIN (0)
AST_SWAP_CHAIN (0) = ASTE#
AST_UNLOCK (ASTE#) =
AST_UNLOCK (ASTE#) I
AST_UNLOCK_FLAG

RETURN
SWAPOUT

BLOCK # = ASTADR(ASTE #)
ASTADR(ASTE #) = ∅

INDEX = ∅

NEXT = AST_SWAP_CHAIN(INDEX)

INDEX = NEXT

NEXT = AST #

Y

REMOVE IT AND RESET UNLOCKED BIT
AST_SWAP_CHAIN(INDEX) = AST_SWAP_CHAIN(ASTE #)
AST_SWAP_CHAIN(ASTE #) = ∅
AST_UNLOCK(ASTE #) = AST_UNLOCK(ASTE #)
& AST_UNLOCKED_MASK

N

AST_CHANGE(ASTE #)
& AST_CHANGE_MASK = AST_CHANGED

Y

SEGMENT HAS CHANGED
PERFORM DISK WRITE

CALL DISKIO

RESET CHANGE BIT AND WAIT FOR DISK WRITE TO COMPLETE
AST_CHANGE(ASTE #) = AST_CHANGE(ASTE #)
& AST_UNCHANGED_MASK

CALL P

FREE MEMORY ALLOCATED TO SEGMENT INDEX = 0
INDEX = MBT_CHAIN (INDEX) & MBT_CHAIN_MASK

MBT_CHAIN (INDEX) & MBT_CHAIN_MASK > BLOCK #

N

PUT ON FREE CHAIN
MBT_CHAIN (BLOCK #) = MBT_CHAIN (INDEX)
MBT_CHAIN (INDEX) = BLOCK # | FREE_MEM

RETURN

Y
SET PRIORITY LEVEL HIGH

SMFR_COUNT (SMFR #) = SMFR_COUNT (SMFR #) + 1

N

SMFR_COUNT (SMFR #) ≤ 0

Y

UNQUEUE MOST RECENT PROCESS

PROCESS_A = SMFR_POINTER (SMFR #)

SMFR_POINTER (SMFR #) = 0

N

SMFR_COUNT (SMFR #) ≠ 0

Y

PT_LINK (PROCESS_A) = 0

N

PT_LINK (PROCESS_B) = 0

Y

PROCESS_A BECOMES READY

PT_FLAGS (PROCESS_A) = READY

SET PRIORITY LEVEL LOW

RETURN
ERROR: SEG IS NOT A DIRECTORY

AST_TYPE (ASTE #) & AST_TYPE_MASK=
DIRECTORY

AST_WAL (ASTE #) & PS_PROCESS_MASK=0

SEG IN MAIN MEMORY

AST_ADR (ASTE #)=0

CALL SWAPIN

SEG IN MAIN MEMORY

CALL LSD

SEG DESCRIPITORS LOADED IN DIR_KSD_ADR
ACCESS MODE=SDR_WRITE_ACCESS

CHANGE BIT CHANGED TO INSURE DISK WRITE AT SWAP OUT TIME

RETURN OK