Virtual Memory System (VMS)

CS-550-3: Operating Systems
Fall 2003

Rajesh Menon
December 04, 2003
<table>
<thead>
<tr>
<th>Table of Contents</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>1 The Process</td>
<td></td>
</tr>
<tr>
<td>1.1 Processor Modes</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Process in VMS</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Process States</td>
<td>5</td>
</tr>
<tr>
<td>1.4 Scheduling</td>
<td>6</td>
</tr>
<tr>
<td>1.5 Symmetric Multi-Processing</td>
<td>7</td>
</tr>
<tr>
<td>2 Memory Management in VMS</td>
<td>8</td>
</tr>
<tr>
<td>2.1 Demand paged virtual memory</td>
<td>8</td>
</tr>
<tr>
<td>2.2 The Memory Subsystem</td>
<td>9</td>
</tr>
<tr>
<td>3 VMS Data Structures</td>
<td>10</td>
</tr>
<tr>
<td>3.1 Process Data Structures</td>
<td>10</td>
</tr>
<tr>
<td>3.2 Memory Management Data Structures</td>
<td>11</td>
</tr>
<tr>
<td>3.3 Address Translation in VMS</td>
<td>12</td>
</tr>
<tr>
<td>4 Process Synchronization in VMS</td>
<td>13</td>
</tr>
<tr>
<td>4.1 Process synchronization between two processes</td>
<td>13</td>
</tr>
<tr>
<td>4.2 Mutual exclusion of Critical Regions.</td>
<td>13</td>
</tr>
<tr>
<td>Conclusion</td>
<td>14</td>
</tr>
</tbody>
</table>

**Appendices**

1 The history of VMS  15
2 The process life cycle  16
   The Interrupt-driven model of VMS
3 Early memory management scheme in VMS  20
   Figure on Page Management System
   Figure on Memory Management Algorithm
4 Scheduling in different kinds of VMS systems  23
   Table on scheduling characteristics of VMS systems
   Table on summary of scheduling algorithms in VMS systems
5 Figure on layout of the Process Control Block  25
   Figure on layout of the relationships between Process Data structures
   Figure on layout of the Job Information Block (JIB)
   Figure on layout of the Process Header (PHD)
6 Software representation of Semaphore  28
7 Layered Operating System Model  29

**Bibliography**  30
DEC VMS

Introduction

VMS (Virtual Management System) is the operating system that runs on the Digital Equipment Corporation’s VAX (Virtual Address Extension) line of computers. It is a multi-tasking, multi-user operating system that supports virtual memory. It was originally designed for VAX architecture in particular. VMS is regarded as a very efficient, robust and secure system even today.

VMS was built to be the leading-edge technology operating system targeted at leading-edge technology customers. VMS was one of the first major operating systems to support virtual memory, 32 bit address space, 128 bit floating point number precision, and a complex instruction set. The importance of VMS rose to increasing prominence from 1980 but the importance declined by 1990. The latest version of VMS is OpenVMS. Even though the primary platform of VMS was VAX, OpenVMS has been designed to extend beyond the VAX architecture to modern architectures like the DEC’s 64-bit Alpha architecture.

My study on VMS does not include its multi-user capabilities. The discussion on multi-processing capabilities is limited to a brief introduction into SMP. Some of the pertinent features of VMS are discussed below in the following chapters.

Chapter 1: The Process

(1.1) Processor Modes

The processor mode or the access mode is a level of privilege. The current access mode of the CPU depends on the type of the process being executed on it. Processor Status Longword (PSL)1 marks this value.

The various processor states2 are as follows.

1. Kernel Mode
2. Executive Mode
3. Supervisor Mode
4. User Mode

1. Kernel mode is the mode in which the critical functions of VMS execute. The entire instruction set of the VAX is available to the process executing in this mode. Kernel mode also allows access to most of the pages in memory. But even the kernel is bound by memory protection. Therefore even in a kernel mode a code would not be allowed to write to system memory where VMS resides.

2. Executive mode is for those processes that need more privileges than that of the user but lower privileges than that of the kernel. Mainly the Record Management Services (RMS)3 executes in this mode. RMS may perform many operations that require a higher privilege than those available to the user. In this mode the RMS data structure is protected from user access or changes.

3. Supervisor mode is more privileged than the user but lower to the kernel and executive modes. This mode is mainly used by the Command Line Interpreters (CLI) who must be able to manipulate the data structures in the control region of the process.

4. User mode is the mode in which all the user processes executes. They include compilers, editors, utilities, etc.

---

1 PSL is a register in the CPU that represents the current state of CPU like the access modes, flags, etc.
2 Please refer to Appendix 7 for diagrammatic representation of the layered VMS operating system.
3 RMS is the highest level of access to VMS file system. This is not covered in this report.
Example for the role of processor modes in memory access

Every page of the virtual memory space has a level of protection related to it. The three type of memory access are namely read-only, read/write, no access. If we consider the system memory space the access type would be mostly kernel-write, executive-read and user-no access.

(1.2) Process States

Figure 1: Process States. Original figure © by 1990 Professional Press, Inc

The basic categories of process states are as follows.

(i) Current process (CUR): The process that currently has the CPU and is executing now. There can be only one process with CUR state on a Uni-processor environment, but in a SMP system there can be one CUR process per processor.

(ii) Computable process (COM): The processes that are capable of being executed form computable processes. They remain waiting because of some other process that has the CPU. The COM processes of a particular priority are executed in a round-robin fashion.

(iii) Wait states: The wait states could be divided into the voluntary wait states and the involuntary wait states. Direct requests by a CUR process for hibernation or suspension place the process in voluntary WAIT states like HIB and SUSP respectively. Any $QIOW^4$ requests also places the process in a voluntary WAIT state like LEF. Indirect requests occur when there is a page fault$^5$ or contention for system resources.

(A) Voluntary wait state is comprised of the following:

(a) Hibernate (HIB): Certain processes like the Swapper remain in Hibernation until it is woken up by another process.

(b) Local event flag wait (LEF): A local event is an event unique to a process. When the process waits for a local even to occur, it sets a bit called the event flag to show this. Such a wait is LEF.

(c) Common event flag wait (CEF): Cooperating processes may wait for an occurrence of a common event. When this occurs a flag is set to show this. Such a wait is CEF.

(d) Suspended (SUSP): In order to enter a voluntary wait state the process must be current. The only exception is SUSP. One process can suspend another that is current.

(B) Involuntary wait state is comprised of the following:

---

$^4$ Input, Output or wait request by the process to the operating system.

$^5$ Need for paging in process image. Details are in the chapter 2 on Memory Management.
(a) **Free page wait (FPG):** It occurs when the free page\(^6\) list is empty.

(b) **Page fault wait (PFW):** In a demand paged virtual memory system page fault brings in the necessary pages of the process image for execution.

(c) **Resource waits such as AST wait (RWAST):** An AST is a component of VMS that lets the user processes be notified of external events like I/O completion or quantum (time interval) expiration.

(iv) **Out-swapped states (Computable and Wait out-swapped):** These are the representation of processes outside the memory in the system back-store or a disk.

**State Transitions:**

1. **Current (CUR) State.**

   (a) A process in the CUR state makes a transition into the COM state when preempted by a higher priority process or if the process utilizes its quantum of time it is assigned to the processor.

   (b) A CUR process makes a transition to the resident WAIT states by any direct or indirect request for any system service that cannot be completed immediately.

   (c) A CUR process can only be deleted because only then the process address space and PHD\(^7\) are available only when it is current.

2. **Computable (COM) and the Computable Out-swapped (COMO) states.**

   (a) A process that is ready to execute first enters the COMO state. The process image is stored initially in the system back store. The Swapper\(^8\) changes its state to the COM state by swapping it into the memory.

   (b) A process in the COM state does not wait for any resources or events. It is waiting to get control over the processor. A process changes from COM to the CUR state when selected by the Scheduler to run on the processor.

   (c) A COM process changes to the COMO state when it is out-swapped to the disk from the memory by the Swapper. This is mainly done to free more memory space for multiprogramming.

3. **WAIT and WAITO (WAIT out-swapped) states.**

   (a) A process that is not CUR or COM is in the WAIT state. The process is either waiting for an event to occur or for some system resource (I/O operation).

   (b) The process moves from the WAIT state to COM with the availability of the requested resource or the completion of the requested event.

   (c) The process moves from the WAIT to WAITO (Wait out-swapped) when the Swapper out-swaps it to free the memory resources.

   (d) The process transitions from WAITO to COMO if its request for resources or an event is satisfied.

**1.3 The Process in the VMS System**

Process is a combination of user’s program and the operating system during execution (Miller, 1997). The process is defined by six elements namely the **program** (the user code), the **data** for the program, the **input** and **output** of the program, the **CPU registers** and the **memory**.

In VMS a process is not equivalent to process image. The process creation deals with initialization of the process data structures such as the PCB, PHD and JIB\(^9\). But the image is formed when the LINK (a process

---

\(^6\) In demand paged virtual memory system the entire physical memory is divided into fixed regions called pages.

\(^7\) PHD is Process Header. Figure in Appendix 2 and chapter 3 on Data Structures.

\(^8\) SWAPPER is a process within the VMS.

\(^9\) Process Control Block (PCB), Process Header (PWD), Job Information Block (JIB).
image by itself) concatenates all the required subroutines referred by the machine readable object code\textsuperscript{10}. This process is called \textbf{binding}. Binding results in an executable code that acts as the process image.

\textbf{Stages in the process life time}\textsuperscript{11}

The process is created at LOGIN, after user authentication. The Command Line Interface (Digital Command Language) forms the interface between the user and the system. The process image is activated and the image is ready for the execution. After image has been successfully executed or terminated due to any exception, the image is rundown to free its memory for other processes. The process deletion is completed at LOGOUT. The \textit{interrupt driven model of VMS} is explained in detail in Appendix 2.

\textbf{(1.4) Scheduling}

A process that enters the system is queued before it gets the processor. It does not directly begin execution. The \textit{scheduler} is the part of the operating system that selects one of the processes in the ready state (COM state) to execute on the CPU.

\textbf{Process Context and Context Switching:} The process \textit{context} is all the information about a process. The context is represented by the general registers in the CPU, Process Status Longword (showing the process privilege, flags, etc), the Instruction Register (IR)\textsuperscript{12}, the base and length registers, and the AST level, to name the most important ones. \textit{Context Switching} occurs when the execution of one process is temporarily discontinued and another process resumes execution in its place. The process context of the former process is saved into the memory. When the process gets the processor back, it resumes execution by moving information from the memory to the PHD (Process header) where the process context is represented.

\textbf{VMS System is a Batch, Interactive and Real-time processes}\textsuperscript{13}: The VMS can accommodate batch, interactive and real-time processes. The processes are distinguished on the basis of the level of priority associated with them. The priority ranges from 0 (the least important) to 31 (the most important). The table explains the process type and priority level.

\textbf{Table 1: Process type and Priority level}

<table>
<thead>
<tr>
<th>Process Type</th>
<th>Priority Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch Process</td>
<td>3</td>
</tr>
<tr>
<td>Interactive Process</td>
<td>4 - 9</td>
</tr>
<tr>
<td>Real-time Process</td>
<td>16 and above</td>
</tr>
<tr>
<td>System Manager Discretion</td>
<td>0,1,2, 10, 15</td>
</tr>
</tbody>
</table>

\textbf{VMS Scheduling Algorithm}

In VMS scheduling is \textit{preemptive}. Hence the process with higher importance can take away the processor from the lower important processes.

The scheduler can employ the following algorithm for process selection.

(1) \textit{First-Come-Firs- Out (FIFO)}: The first process that comes to the system is executed to its completion before selecting the next process. The FIFO is applied for batch processes or for processes with identical priority. FIFO is also employed in the management of \textit{free list} and \textit{modified list} in the memory\textsuperscript{14}.

\textsuperscript{10} Compiler converts the source code into machine readable object code.
\textsuperscript{11} Appendix 2 explains the life cycle of a process in detail.
\textsuperscript{12} IR is also called PC or the Program Counter.
\textsuperscript{13} See Appendix 4.
\textsuperscript{14} They are dealt in chapter 2 on Memory Management.
(2) **Round Robin**: In this scheme of scheduling each process of a particular priority hold the CPU for a fixed quantum of time. Hence a more fair distribution of processor time is made between the processes ready to run.

(3) **Shortest Job First (SJF)**: This algorithm is applied to mainly the batch processes. The scheduler selects the process with the shortest job from the ready queue. In OpenVMS this algorithm is used in print queues.

(4) **Shortest Remaining Time (SRT)**: The algorithm is a variation of SJF. The SRT is calculated by subtracting the actual execution time from the estimated time of execution, as the process relinquishes the processor. The estimated time is then decremented for reevaluation by the scheduler.

(5) **Priority**: The highest priority process gets the processor first. As it relinquishes the processor the lower priority processes get the processor. The higher priority process preempts the lower priority processes. Within the same priority level, the *round robin* algorithm is usually applied by the scheduler. When all the process is at a particular level of priority has been exhausted then the lower priority queues are serviced.

(6) **Aging**: Sometimes the lowest priority process is never catered as higher priority processes continue to get hold of the process. This leads to a situation called *starvation*. Hence the priority of a process is raised in a particular time interval so that process gets the processor at one point. This can be guarantee a fixed *turnaround time*. 

When is scheduler invoked?

In the VMS system the real time processes pre-empt the time-shared processes. In the same call, the time-shared ones pre-empt the batch processes.

A scheduler is invoked under three circumstances.

(1) The scheduler is called when a system event occurs such as I/O completion, process creation, terminal I/O completion, etc. System events are not connected with the currently executing process. If we consider the completion of I/O request, for instance, the process state from WAIT to COM or it gets added to the ready queue. There is a need for priority re-evaluation based on this situation.

(2) The scheduler is called when the quantum counter in the Process header (PHD) that is being continuously decremented, becomes zero. Then the process has to relinquish the CPU.

(3) The Scheduler is also called to perform the wait service. For instance, a $QIOW (input, output or wait) or a $HIBER request (hibernation) made by the executing process, causes the scheduler to change the state of the process as requested.

The scheduler does three tasks when it is called. Based on the priority the new process, the process environment represented by the context is saved into the memory. Then the scheduler selects a new process to run from the ready queue. The process context of the new process is loaded into the processor.

(1.5) **Symmetric Multiprocessing** or SMP

SMP in VAX machine is a tightly coupled system as opposed to VAXcluster that is a loosely coupled system. Multiprocessing in VMS is called Symmetric Multiprocessing because each CPU can run both the operating system and the application programs. There is no master-slave relation among the different CPUs.

Each CPU has a private area for storage of the CPU-specific database. The database stores information like the process information, busy/wait indicator, pointer to the interrupt stack, etc.

---

15 Quantum may be .01 to .1 seconds.
16 A job refers to a process along with all its sub-processes as shown in the Job Information Block (JIB).
17 Amount of time a user should wait before the system can admit his job into the system.
18 Higher priority or equal priority (round robin substitution)
19 A system capable of executing more than one process simultaneously.
20 Tightly coupled system has multiple CPUs that share important system resources like the system bus, system clock, memory, peripheral devices and power supply.
Some Important terms with respect to SMP in VAX

(1) The *available set* is the set of CPUs available in the system. The *active set* is the set of CPUs currently running while the *idle set* is the set of CPUs ready to execute a process.

(2) Process Affinity: Certain process would prefer to run on a particular set of CPU. For instance, a process that executed on a CPU with a larger physical memory cache might use the data still cached in it, if it is placed back on that CPU. But they would also run on the others if there are none from the former set.

(3) Device Affinity: It occurs when a device has connection only to a particular CPU or the device can issue a hardware interrupt only to that particular CPU.

(4) Capability: Capability expresses a stronger bond between a process and CPU than affinity. If the process has a required capability then the CPU with that capability can only run that process. Hence it is hardware-based.

CHAPTER 2: Memory Management

(2.1) Demand paged virtual memory

VMS answer to the shortcomings of the partitioned memory was to develop the concept of **paged memory**. Every process that enters the system has a virtual address. The virtual address is divided into four parts. The S0 is the system region that may be shared. The P0 is the program region that exists as long as the process image and P1 is the control region for the process maintained by VMS. The fourth part is not used.

The process and the physical address spaces are divided into equal parts called *pages*. Hence only those pages that are currently being executed by the processor needs to be in the memory. Hence the problem of the size of the processes being limited by the size of physical memory was overcome.

The process that is scheduled for execution has its image created on the disk. The process image is divided into pages also called *virtual pages*. The memory is also divided into *page frames* or *physical pages*. A page frame and a virtual page are of the same size. On demand, the pages that need to be executed are loaded on to the memory frames.

Translation-not-valid Exception

Each page of the process image has data and instructions in it. As the instructions are executed consecutively, the Program Counter (PC) advances to the address of the next instruction to be executed. If the PC advances beyond the page frame of the memory to an address (concurrent or at a different location such as JUMP machine instruction) not currently on the memory an exception is generated. This exception is the translation-not-valid exception or more commonly called the page fault. Thus the services of the operating system is sought to read in the required pages from the disk to the page frame.

The Principle of Locality and the Working Set

The principle of locality states that the memory locations that have been recently referenced are likely to be referenced again in the near future (Miller, 1997). The pages that are currently in the page frames are called *active pages*. Active pages are the pages that the process requires for execution. The collection of active pages that are currently resident in the physical memory is called the *working set* of the process at a given time. Thus the operating system aims to allocate memory frames to the working set of a process.

The memory management subsystem controls the size of the working set by replacing the old inactive pages with the new pages that are faulted in. The important page replacement algorithm followed is the **Least Recently Used (LRU)** algorithm with a reference bit.

---

21 Please refer to Appendix 3 for shortcomings of partitioned memory.
**Least Recently Used (LRU):** The algorithm selects the frame that has not been referenced for a longest amount of time, for replacing the old page with the new page faulted in to the memory. The difficulty of the algorithm is the requirement of the operating system to timestamp each page that is brought into the memory. Since the frames in the working set are in no particular order the operating system has to search each page for the timestamp for every page fault.

Hence LRU algorithm was used along with a single bit, called the **reference bit**, which is set by the VAX hardware. Whenever a page is referenced the bit is set (set to one). Any page with reference bit set to zero is selected for replacement. At regular intervals such as a clock interrupt the reference bit is cleared. A reference counter may be also used. The counter is incremented for each reference of the page. The counter is also periodically cleared.

**Size of the working set:** The size of the working set allocated to a process depends on its need. If the process faults beyond the highest threshold for all process, then the size is increased. If there process faults below the lowest threshold then the size is reduced. Between the two thresholds the working set is not disturbed.

*(2.2) The memory management subsystem of VMS*

The memory management subsystem of VMS is comprises of the following.

(1) **Page fault handler:** The page fault handler of the operating system moves a required page of the process image from the disk to the working set. The page fault handler controls the working set size for all the processes on the system so that there is a fair allocation of memory between them. The handler implements the **local replacement** policy where in the pages of other processes are not allocated to a thrashing process.

(2) **Page management**:22

The page management can be explained using the following illustration.

(i) Initially, when the system boots up, the memory holds only the resident routines23 and their data.
(ii) The remaining frames are for allocating the pages of the process image which exists as the free frames represented by the **free list**.
(iii) As processes are added on to the system by the user, each process would have a working set list with its active pages.
(iv) During process execution, if the working set becomes too big, the frames that have not been used recently are removed from it.
(v) These removed frames go to the **free-list** or to the **modified-list** if there has been some modification to the original page that was faulted in to the memory.
(vi) When the process is deleted the entire working set returns to the free-list.

All the information about a page is kept by the hardware in the **Page Table Entry (PTE)**24 associated with each page.

**Tackling a page fault:** **Hard fault** or **Soft fault**25

**Hard fault** occurs when the requested page is on the disk. If the requested page is either in the free-list or the modified list then the requested page needed not faulted in from the disk. Here there is a **soft fault** where such a page resident in the memory can be made part of the working set. Thus with the cost of a little overhead in maintaining these lists, a lot of unnecessary disk access is eliminated.

22 Please refer to Appendix 3 for diagrammatic representation of page management.
23 They are those processes or subroutines that are always resident in the memory.
24 Please refer to chapter on Data Structures.
25 Please refer to Appendix 3 for figure.
Both the free-list and the modified-list is governed by the First-In-First-Out (FIFO) algorithm. The frame that is replaced with the new page is the one that is on the list for the longest amount of time. Hence a processes oldest page that has been removed from its working set is retained in memory for the longest possible time. This increases the possibility for a soft fault.

(3) SWAPPER: The reduction or the lack of physical memory for new processes causes the operating system to remove inactive processes from the memory. This is called out-swapping. Conversely, the in-swapping operation brings the processes back to memory from the disk. These operations are carried out by a process called SWAPPER. The SWAPPER is not a procedure like the rest of VMS elements. The SWAPPER remains in the HIB state until it is woken up by the SCH$SWPWAKE procedure. This procedure determines the need for the SWAPPER in situations like an out-swapped process is ready for execution, etc. This procedure then changes the SWAPPER state from HIB to COM.

SWAPPER functions:

(i) Copying into Paging File on the disk: When the modified-list becomes too large or the free-list becomes too small, the SWAPPER copies the inactive pages to the Paging File that holds all the out-swapped modified pages from the memory. The modified-list is then attached to the free-list increasing its size.

(ii) Shrinking of Working Sets: The SWAPPER can carry out a uniform reduction of all working sets in the memory with the hope of accommodating newer processes on the memory.

(iii) Copying into the Swapping File: As a last resort when there is more need for physical memory for new processes, the SWAPPER may out-swap an entire inactive process into a Swapping File by copying its working set into the file. The process is now in the HIBO state until the SCH$SWPWAKE decides to make it into the COM state.

(iv) Faulting into memory pages as clusters: The SWAPPER faults in the pages in the Paging File as group of adjacent pages in keeping with the principle of locality of reference. These group of pages are called clusters.

Chapter 3: VMS Data Structures

(3.1) Process Data Structure

The process context\(^{26}\) is represented by three elements.

1. The software context: The software context of the process includes the data structures that represent the different characteristics of the process. They are as follows.

(i) The software Process Control Block (PCB)\(^{27}\) or simply the PCB

PCB is the fundamental data structure of a software process is PCB. It specifies the name, state, priority, data values, event flags and quota limits of the process. The forward link and the backward link are pointers that connect the PCB to the PCB double-link list. The housekeeping field comprises of an integer identifier for the block type and the length of the block. The AST Control Block (ACB) in also in the PCB. The PCB has also pointers to the JIB and the PHD.

(ii) The Job Information Block (JIB)\(^{28}\)

JIB: In VMS, jobs refer to the aggregate of parent process and its sub-processes (Miller, 1997). The shared quota of memory space controls the maximum number of sub-processes allowed. Logically the parent and the sub-processes form a tree structure with the parent at the top most level and they point to the same JIB.

\(^{26}\) Process environment in system represented by various data structures

\(^{27}\) Please refer to Appendix 5 for PCB diagram.

\(^{28}\) Please refer to Appendix 5 from JIB diagram.
But each sub-process has a different PCB noted in the PCB vector. The account name and user name are common to the family.

2. The **hardware context**: The hardware context is represented in the **Process Header (PHD)**\(^\text{29}\).

The PHD includes the working set\(^\text{10}\) list of the process, P0 and P1 page tables\(^\text{31}\), etc. The **hardware PCB** is a part of PHD. It includes the stack pointers, general registers, and the AST\(^\text{32}\) level register. The base and length registers for the program and control regions are also part of the hardware PCB. These four registers define the virtual memory space of a process.

3. The **virtual address space**\(^\text{33}\): In the VMS no process is directly allocated physical memory space. The programs are written to use the virtual addresses. The process of mapping the virtual address to physical memory address is called **address translation**.

The PCB, the JIB and the PHD are all dynamically allocated from the **data pool**\(^\text{34}\). The PCB and JIB remain in the memory for the lifetime of the process. But the PHD can be written to the disk when the process is out-swapped from the memory. PHD is required only when the process is current. Fixed memory locations where the PHD is stored are called **balance slots**. PHD is stored in one of the balance slots when it is resident. The processes in the balance slots are called the **balance set**.

**(3.2) Data Structures for Memory Management.**

**VAX address Format**

The pages of the process image are scattered all over the physical memory frames. Hence there is the need to **map** or translate the virtual address into the physical address.

**Figure 2**: VAX address format. Original figure ©1997 by Butterworth-Hienemann

<table>
<thead>
<tr>
<th>Region</th>
<th>Page</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

The 32-bit address VAX architecture is divided into three fields.

(i) The region  
(ii) The page  
(iii) The offset

**Region field:**

The region field can take four values. 0 for process 0 or P0, 1 for process 1 or P1, 2 for SYSTEM or S0, 3 for some VAX architecture. The P0 region contains the major part of instructions in the process image. P1 contains the user data.

**Page field:**

The page field is a 21-bit field that contains the page number from 0 to 2,097,151 (value of \(2^{21} - 1\)) that totals to 2 mega-pages.

---

\(^{29}\) Please refer to Appendix 5 from PHD diagram.  
\(^{30}\) Please refer to chapter 2 on Memory Management.  
\(^{31}\) Please refer to chapter 2 on Memory management.  
\(^{32}\) Asynchronous Trap that let process know of completion of I/O or quantum expiry  
\(^{33}\) Handled in chapter 2 on Memory Management  
\(^{34}\) Areas in the Virtual Memory Address where VMS allocates and de-allocates data structures
Offset field

The offset field is 9 bit long that have values from 0 to 511 totaling to 512 values that equals the size of a block in a disk. It shows the offset into the page where the page table entry is located.

Page Table Entry (PTE)

Figure 3: VMS page table entry format. Original figure ©1997 by Butterworth-Hienemann

Figure 3: VMS page table entry format.

The PTE is an important data structure that resolves the virtual page number to the page frame on the physical memory. PTE entry is a 32-bit entry. It is important to note that there are two page tables, the P0 table and the P1 table, for each process. The other page table is the system region (SYS) page table that all the processes share. The three important fields that are relevant here are as follows.

(i) The valid-bit or V-bit which is set to one if the page is in the memory.
(ii) The modify-bit or the M-bit, which is set to one if the page has been changed after it, was faulted in.
(iii) The frame number in the memory where the page that is faulted in is allocated.
(iv) The bits 26 and 22 shows whether the page that is not in the working set (V-bit is set to zero) is in the free-list or the modify-list in the memory.

(3.3) Address Translation

Figure 4: Translation Model. Original figure ©1997 by Butterworth-Hienemann

Virtual address

Address

Page Table

Instruction Read/write mode

Validate and convert virtual address to physical address by the hardware

Physical

Page Fault

Address invalid exception

The hardware performs the virtual address to physical address translation. The primary components for the translation are the virtual address and the page table. The Instruction mode tells whether the address is to be used to perform a read function or a write function. The processor mode is the security function that shows the mode of execution (kernel, executive, supervisor or user mode).

Validation of addresses: The VAX hardware has 6 registers for the purpose of validation. A pair or registers cater to the P0, P1 and SYS page tables. One of the pair is the base-register and the other is the length-register. The validation and conversion of the virtual address to physical address is called VAX address translation.

---

35 When M=1 and V=0 then the page has to be in the modified list.
36 VAX address translation is not covered in this paper.
Steps in address translation: A simple example

(1) An instruction is resident in the memory. As execution proceeds, the PC points to the address of an instruction in a different page.

(2) To see if the instruction is in the physical memory, the page number part of the virtual address is used as an index into the PTE.

(3) If we assume that the V-bit of the PTE for this index is set to 0, a page fault occurs.

(4) The page fault handler examines the 22 and 26 bit to determine the location of the virtual page. Again we assume the page is on the disk for simplicity.

(5) The handler selects a new frame from the free-list to allocate to the new page. The new page is read in and the PTE is updated (V-bit set to 1).

(6) The instruction is re-executed.

(7) The frame number from the PTE is combined with the virtual address offset to form the physical address of the required instruction.

CHAPTER 4: Process Synchronization

(4.1) Process Synchronization between two processes

Process synchronization achieves operational privacy for processes through inter-process communication. It allows processes to cooperate for system resources and share data between themselves for communication while preventing accidental interference by one process on the functioning of the other. In modern VMS or the OpenVMS this is achieved in three ways.

(1) Hibernate-Wake: Two system services SHIBER (to sleep) and SWAKE (to wake up) are used in this method. When two processes depend on one another for execution such that one has to execute after the other, the first process executes while the second hibernates or sleeps. When the first process completes it ‘wakes’ the second process and goes to hibernation. Only one bit of data is passed between the processes. It is best suited for a pair of processes.

(2) Event-Flag: Two or more processes can use this method to synchronize their execution. The flag is a two bit flag with three options- Set Event Flag, Clear Event Flag or Wait for Event Flag. Even in this method only a single bit it transferred between the processes.

(3) Mailbox: It is a special form of I/O that takes place between two processes. The data that is transferred between two processes is more than one bit. One of the process writes to the mailbox and the other process reads from it.

(4.2) Mutual Exclusion of Critical Regions

When multiple processes execute in parallel and they execute on shared regions of code, the resource sharing needs to be carefully resolved between all competing processes. The portion of the code that accesses the shared variables is called the critical section. Mutual Exclusion is the method by which the critical section can be accessed only in a serial manner.

The one of the important methods is by the implementation of a semaphore for mutual exclusion.

Functioning of Lock and Unlock services

Lock and unlock are system services used to enforce mutual exclusion on any resource. They are available from any mode. An executing process reaches the shared section of code. It executes a LOCK before entering its critical region. Lock gives the process exclusive access to the shared variable. When another process tries to execute the LOCK it will be put on the wait state. When the first process exits the critical region it executes the UNLOCK which changes the state of the waiting process back to executable.

---

37 The order of execution is known.
38 Physical Resources
39 One at a time
40 Please refer Appendix 6 for semaphore implementation.
Conclusion

Advantages of VAX/VMS

(1) Despite the wide range in size and power of the different models, all the models have similar architecture (i.e. the hardware design as it occurs to the software).
(2) All models can run the same Operating system namely the VMS. Truly, DEC strived to adhere to their policy, “One architecture, one operating system”.
(3) VAXs work well together in a peer-to-peer relationship. This helps multiple computers to carry out tasks in cooperation and such a unit is called VAXcluster. VAXcluster provides easy sharing of resources and also redundancy in the case of malfunctioning by any of the participating systems.
(4) Since DEC developed VMS and VAX together the designers claim that the VAX is a “balanced architecture” as they were able to compromise between the demands of the hardware and those of the software, thereby increasing the efficiency of the computer. In the case of portability issues between other operating system versions, the burden is on the operating system to include the lowest common denominator of all the versions.

Disadvantages of VAX/VMS

(i) DEC’s total dependence on the complex instruction set (CISC) for VAX hurt it economically because a new family of simple instruction sets called the RISC became popular and widely used.
(ii) VMS was written in VAX assembly language. This made translating the millions of line of code to another assembly language a daunting task.
(iii) The architecture design of VMS had a major flaw with respect to implementation of Interrupt Priority Levels (IPL)\(^1\) to get the effect of locking. IPL was a highly non-standard feature. Every lock was highly overloaded in meaning. This made VMS more complex and non-scalable.
(iv) The total picture of process synchronization and management of deadlock was poorly understood as it had been implemented in the VAX architecture. Hence the scope of evolution of VAX architecture was severely limited.

Future of VMS

DEC’s 64 bit Alpha architecture is the successor of VAX. It is a very fast 64-bit RISC processor architecture. But Alpha/VMS could not completely eliminate the shortcomings of the VAX/VMS. Hence it did not become a sure-shot market success. The principal operating systems being used with Alpha are OpenVMS, Tru64 UNIX and Windows NT (older versions), and the various GNU/Linux variants.

In the early 1990s, DEC renamed the VMS (version 5.5) operating system as the OpenVMS. OpenVMS embraced new, more open standards like POSIX\(^2\) compatibility of some sort and using of Motif\(^3\), among others. Presently, only VAX and Alpha are able to run OpenVMS. But Hewlett-Packard’s decision to port the system for Intel's new Itanium processor may give OpenVMS a new lease of life.

\(^1\) IPL is maintained by the hardware. Each interrupt was assigned an IPL. Thus it represented which interrupt was allowed to occur at a particular time. The higher the IPL the more time-critical was the event.
\(^2\) POSIX is the acronym for Portable Operating System Interface for UNIX, a set of IEEE and ISO standards that define an interface between programs and operating systems. By designing their programs to conform to POSIX, developers have some assurance that their software can be easily ported to POSIX-compliant operating systems. (webopedia.com)
\(^3\) MOTIF is a set of user interface guidelines created by the Open Software Foundation (www.opengroup.org) that specify how an application should look and feel, covering issues like the placement of title bars and menus. It is used on more than 200 hardware and software platforms and has become the standard graphical user interface (GUI) for UNIX.
Appendix 1

History

The evolution of various phases of DEC operating systems is detailed below. The major features of each operating system are also detailed. It is important to note that DEC developed their hardware architecture and the software that was to be supported by it together. They believed in the philosophy that hardware and software two sides of the same coin. In a way the VMS dependence on VAX architecture is responsible for lack of popularity by 1990.

Table 1: Evolution of VMS

<table>
<thead>
<tr>
<th>DEC Operating Systems (OS)</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>First Generation Computer OS</strong></td>
<td></td>
</tr>
<tr>
<td>OS in Programmable Digital Processor (PDP-1)</td>
<td>Time sharing OS, Static partition of the available physical memory.</td>
</tr>
<tr>
<td><strong>Major Disadvantages:</strong></td>
<td></td>
</tr>
<tr>
<td>Job-to-partition size had to match.</td>
<td></td>
</tr>
<tr>
<td>OS in later generations of Programmable Digital Processor (PDP-6, PDP-8)</td>
<td>Time sharing OS, Dynamic partition of the available physical memory.</td>
</tr>
<tr>
<td><strong>Major Disadvantages:</strong></td>
<td></td>
</tr>
<tr>
<td>External Fragmentation due to the lack of contiguous memory for process allocation .</td>
<td></td>
</tr>
<tr>
<td>OS in PDP-10</td>
<td><strong>Re-locatable partition allocation:</strong> Hardware solution to problem of fragmentation in the form of base register and bounds register.</td>
</tr>
<tr>
<td><strong>Mapped memory:</strong> memory address space seen by the program need not be the physical address space used.</td>
<td></td>
</tr>
<tr>
<td><strong>Second Generation Computer OS</strong></td>
<td></td>
</tr>
<tr>
<td>RSTS OS on PDP-11</td>
<td>General Purpose, time sharing, Multi-user OS.</td>
</tr>
<tr>
<td><strong>Third Generation Computer OS</strong></td>
<td></td>
</tr>
<tr>
<td>VMS (Virtual Management System) on VAX (Virtual Address Extension) computers</td>
<td>Demand paged virtual memory.</td>
</tr>
<tr>
<td><strong>Modern Age Computer OS</strong></td>
<td></td>
</tr>
<tr>
<td>OpenVMS</td>
<td>SMP , Clustering.</td>
</tr>
</tbody>
</table>

---

44 Partitions of various sizes were allocated and de-allocated leading to fragmentation.
45 SMP or Symmetric Multiprocessing is dealt in the first chapter.
Appendix 2

The Process

(1) The process Life Cycle

Figure 1: Process Life Time. Original figure © by 1997 Butterworth-Heinemann

(1) Login and Process Creation:

(1) The terminal driver \(^{46}\) detects key strokes from a terminal device such as a keyboard. If there is no other process performing a READ operation \(^{47}\) on that terminal, the terminal driver alerts the system process called JOB_CONTROL. The JOB_CONTROL validates the key strokes for security reasons. A particular combination of strokes notifies an attempt to log-in by the user.

(2) The JOB_CONTROL executes a system service routine called the $CREPRC (Create Process). The $CREPRC creates a dummy process that eventually becomes the user’s process after validation of the username and password by LOGINOUT, an image \(^{48}\) of the dummy process that was newly created. If the login fails \(^{49}\) the LOGINOUT destroys the dummy process not allowing the user to login.

---

\(^{46}\) Terminal Driver is the interface of the VMS that communicates with the terminal I/O devices like the keyboard.

\(^{47}\) If any process is performing read operation at a terminal then login has already occurred and the interrupt from the terminal devices is for the READ operation.

\(^{48}\) Image refers to an executable file.

\(^{49}\) Login fails due to bad username, password, or both.
(3) The SCREPRC also creates the three data structures for the process namely the Process Control Block (PCB), the Job Information Block (JIB) and the Process Header (PHD) in the kernel area called the **Scheduler Database**.

(4) A significant image that is activated by the LOGINOUT on successful login is the Command Line Interpreter (CLI). CLI is that part of the operating system that reads, interprets and executes commands entered by the user. The CLI is a level of abstraction around the OpenVMS. It acts as a shell around OpenVMS that makes the system easier for the user to access (Miller, 1997).

(2) Image Formation, Activation, and Rundown.

It is pertinent to note that in VMS a process is not synonymous with the process image. The process creation deals with initialization of the scheduler database. But the image is formed when the LINK (process image by itself) concatenates all the required subroutines referred by the machine readable object code. This process is called **binding**. Binding results in an executable code that acts as the process image.

**Image Formation**

**Figure 2:** Image file format. Original table © 1997 by Butterworth-Heinemann

<table>
<thead>
<tr>
<th>Image Header</th>
<th>Image Body</th>
<th>Fix-up Information</th>
<th>Debug</th>
</tr>
</thead>
</table>

The process image consists of four parts.

The **image header** contains preliminary information about the image such as the memory requirements and disk location of the code. The goal is to avoid the need for reading the entire image on to the memory during execution.

The **image body** contains the machine instructions for the VAX machine.

The **fix-up information** provides the locations that must be considered for binding.

The **debug** area contains the names and locations for the variables used in the program.

**Image Activation**

Activation of image deals with the following:

(1) From the user’s point image activation occurs with the command RUN.

(2) The activator verifies the availability of the file and privileges of the user.

(3) The most important function of the activator is to calculate the memory requirements for the image and store this information in the PHD.

---

50 Kernel is that part of VMS that is continuously resident in the memory.

51 More in the report on data structures of VMS.

52 Digital Command Language is the most common CLI in OpenVMS system.

53 Process is the environment created and maintained by the operating system for the user program.

54 Compiler converts the source code into machine readable object code.

55 Privileges are not covered in the report.
(4) The activator also calculates the memory requirement for the stacks that are used by calling subroutines and for storing data.
(5) The activator makes the bindings for the DEBUG (specified by the user) and TRACEBACK (if not suppressed by the user) (Miller, 1997).

Image Rundown

Image rundown occurs when the image has finished execution or it encounters a fatal trap that terminates it. The major function is to enable the release of image-specific memory and also to cancel any outstanding I/O requests.

(3) LOGOUT and Image Destruction

The signal to delete a process is LOGOUT. The deletion algorithm is the inverse of the creation algorithm. The JIB and the PCB are returned to the data block pool. The PHD is freed for other processes.

The steps of destruction are as follows:

(1) The CLI recognizes the command LOGOUT and calls the system service $DELPRC (Delete Procedure).
(2) The procedure runs down the process image and then it runs down the CLI itself restoring the memory used to the free memory pool.
(3) The procedure also prohibits any further scheduling to avoid the use of the inconsistent and incomplete database.
(4) Once the rundown is complete the procedure call the scheduler to schedule a new process to execute.

(2) Logical Model of VMS\textsuperscript{57}: Interrupt Driven Model

(1) Consider a user process being executed on the CPU.
(2) An interrupt from an external I/O device or an exception\textsuperscript{58} triggered by the hardware (peripheral devices) transfers control to the operating system. These interrupts or exceptions are associated with the appropriate interrupt handlers.
(3) Step 2 discontinues the process execution so that the processor can respond to the interrupt or exception. The process context\textsuperscript{59} is saved so that it can be restarted at the exact state in which it was discontinued.
(4) The exception causes the terminal driver\textsuperscript{60} to determine if any I/O request has been completed.
(5) A $QIOW request is a system service that a process uses to request I/O operations. Hence the request is validated and queued to the I/O queue of that device.
(6) In the case of the exception, if the terminal driver indicates that there is no more data to be transferred then the process is unblocked and the process state changed from WAIT to COM by the scheduler.
(7) Conversely, if there is more data to be transferred then the execution of the interrupted process\textsuperscript{61} is resumed.

\textsuperscript{56} Process Header, see report on Data Structures.
\textsuperscript{57} Refer to Appendix 1 for Diagram of the Operating system model.
\textsuperscript{58} The exception differs from interrupt in that it is synchronous with the flow of the process (Miller, 1997).
\textsuperscript{59} The process context refers to the state of the process shown by the state of the process data structures at a particular instant of its life on the system.
\textsuperscript{60} Terminal Driver is the interface of the VMS that communicates with the terminal I/O devices like the keyboard.
\textsuperscript{61} This is not the blocked process that issued the request for I/O. The interrupted process continues execution as in the case of Direct Memory Access (DMA) for high speed devices.
Figure 3: Operating System Model. Original Figure © 1997 by Butterworth-Heinemann.
Appendix 3

The Memory Management

Early memory management Scheme in VMS

The common approach to the implementation of multiprogramming, until VMS, was to assign physical memory to the processes as they log in. This kind of allocation is called contiguous allocation, which is synonymous with the term partitioned memory allocation. In such a system, the address translation\(^{62}\) can occur either at link time or run time. Many well know algorithms were used for such allocation namely the first-fit, best-fit and the worst-fit algorithms. The methods differed on how the processes were allocated to the contiguous memory spaces.

Disadvantages:

(1) The major problem of such an approach was external fragmentation where the memory had many chunks of free space that were distributed between processes but they were too small to be allocated to any process.
(2) There was also the problem of returning the allocated space to the free memory pool. The general concept of free memory pool management is to combine smaller spaces to a larger memory block to accommodate a process.
(3) The most serious problem was that the system could not execute any process that was bigger than the physical memory. To some extend this problem was catered by the principle of overlays. The principle is applied when a process that could be divided into functions is executed as branches. Only the executing function is loaded into the memory. The function that is not always needed, such as the interrupt handler, is loaded only when it is required. This is achieved by overlaying the function that called the handler in the memory with the interrupt handler.

Hence the modern VMS system uses the demand paged virtual memory system.

\(^{62}\) Please refer to chapter on Data Structures.
(2) Representation for page management in VMS

Figure 4: Page management system of VMS. Copyright © 1997 by Butterworth-Heinemann
(3) Page management algorithm

**Figure 5:** Detailed page management algorithm of VMS. Copyright © 1997 by Butterworth-Heinemann
Appendix 4

(1) Scheduling in different VMS systems

VMS systems may be grouped into three kinds.

(1) **Batch Systems:** A batch system collects a sequence of activities called a job from the user. The system executes the job at a time it considers as optimal. Hence the execution may not be immediate. Execution in a batch mode can be for a single program at a time (Serial processing) or it may be for more than one job (parallel processing). In a batch system, the scheduler is called a **Long-term scheduler** which comes into play occasionally when a process relinquishes the processor voluntarily.

(2) **Interactive Systems:** Interactive systems are also called timesharing systems. The system obtains user-input at various steps in the execution of the process and proceeds accordingly. Such systems are user-adaptive and also focused at problem solving. In such systems a process gets the CPU for a fixed time slice called a quantum. When the quantum for one of the processes is used up, another process gets the processor for the same quantum. Hence all the processes get the processor for some fixed time period giving the impression that the system responds to all the processes equally. The tradeoff is a slight delay in the execution time for a single process when compared to its execution time on a batch system, where it has complete hold over the processor when it executes. In the interactive systems the scheduler may be called the **Short-term scheduler** which decides which process gets the CPU.

(3) **Real-Time Systems:** The real-time operating system takes input from different transducers or sensors (temperature, velocity etc) and makes computations and decisions on the data so obtained. The decisions have to be made as soon as the inputs are available. Hence the Short-term scheduler schedules a process to the processor to obtain output without any delay and waits for any other process that needs to be scheduled to the processor.

(2) **Scheduling Characteristics of the three types of Systems:**

<table>
<thead>
<tr>
<th>System Type</th>
<th>Time-shared</th>
<th>Preemptive</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch</td>
<td>No</td>
<td>No</td>
<td>optional</td>
</tr>
<tr>
<td>Interactive</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
</tr>
<tr>
<td>Real-time</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

---

63 Uni-programming or Multi-programming or time sharing environment.
64 This refers to Symmetric Multi Processing.
**Summary of Scheduling Algorithms on VMS systems**

Table 3: Original table © 1997 by Butterworth-Heinemann

<table>
<thead>
<tr>
<th>System Type</th>
<th>Algorithm</th>
<th>Round robin</th>
<th>Shortest job First</th>
<th>Shortest remaining time</th>
<th>Priority</th>
<th>Aging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>Interactive</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Usually</td>
<td>Usually</td>
</tr>
<tr>
<td>Real-time</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Appendix 5

Process Data Structures

(1) Layout of the Process Control Block

Figure 6: PCB. Original figure © 1997 by Butterworth-Heinemann
(2) Layout of the relationships between Process Data structures

**Figure 7:** Process Data Structures. Original figure © 1997 by Butterworth-Heinemann

(3) Layout of the Job Information Block (JIB)

**Figure 8:** JIB. Original figure © 1997 by Butterworth-Heinemann

<table>
<thead>
<tr>
<th>Forward Link</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Backward Link</td>
<td></td>
</tr>
<tr>
<td>Housekeeping</td>
<td></td>
</tr>
<tr>
<td>User Name</td>
<td></td>
</tr>
<tr>
<td>Account Name</td>
<td></td>
</tr>
<tr>
<td>Shared Quota</td>
<td></td>
</tr>
</tbody>
</table>
### Layout of the Process Header (PHD)

**Figure 9:** PHD. Original figure © 1992 by Van Nostrand Reinhold

<table>
<thead>
<tr>
<th>Fixed Portion (PHD Structure)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working Set List (expands toward PST)</td>
</tr>
<tr>
<td>Expansion Area for Either WSL or PST</td>
</tr>
<tr>
<td>Process Section Table (expands toward WSL)</td>
</tr>
<tr>
<td>Empty Pages</td>
</tr>
<tr>
<td>Four Arrays for Process Header Pages</td>
</tr>
<tr>
<td>P0 Page Table (expands toward P1 table)</td>
</tr>
<tr>
<td>Expansion Area for Either Page Table</td>
</tr>
<tr>
<td>P1 Page Table (expands toward P0 table)</td>
</tr>
</tbody>
</table>
Appendix 6

Semaphore

A semaphore is a operating system provided synchronization tool. Semaphore has a single integer variable that can be modified using a system call. The semaphore can be accessed only using two operations- the \texttt{wait()} and the \texttt{signal()}. When a process executes a \texttt{wait()} and finds that the semaphore value is negative it issues a blocking call and places itself in the wait queue. As a process exits the critical section it executes \texttt{signal()} that increases the semaphore value. If the value is zero or more then the process on the wait queue is awakened are it is ready for executing in the critical region.

Software equivalent of operation of semaphore: Class Notes © 2003 by Charles Abzug

typedef struct
{
    int value;
    struct process *linked_list;
} semaphore;

Wait operation

\texttt{Wait(speedemaphore S)}
{
    S.value--;
    if (S.value < 0)
    {
        add this process to S.linked_List;
        block();
    }
}

Signal Operation

\texttt{signal(speedemaphore S)}
{
    S.value++;
    if (S.value <= 0)
    {
        remove process P from the S.linked_List;
        Awaken(P);
    }
}

It is to be noted that the explanation of the semaphore servers to give a general idea about the semaphore only. The actual implementation in VMS is different.
Appendix 7

Interrupt-driven VMS operating system

Figure 10: Representation of layered structure of VMS operating system. Original figure © 1992 by Van Nostrand Reinhold.
Bibliography


ABZUG, CHARLES (2003). *Process Synchronization and Deadlock*, class notes; Department of Computer Science, James Madison University.

Other good materials on this subject
