THE DEC VMS OPERATING SYSTEM

(Draft)

John Lynch

Operating Systems CS 345-60

April 4, 2000

The DEC VMS operating system was developed in 1975 by the Digital Equipment Corporation. It was developed to run on Digital Equipment Corporation's VAX computer system (Virtual Address Extension). The new VAX system was designed to overcome memory limitations that had existed on earlier computers by use of virtual memory.

Virtual memory creates the illusion of an extremely large memory. It was began in 1969, and had been implemented by IBM, Honeywell, and Sperry in large mainframe systems by 1972. A technique called demand-paged memory management was used for Virtual memory.(Bynon 1-7)
In virtual memory a virtual address space is created for the machine. As an example, if the VAX needed to process data and instructions whose addresses are scattered throughout a four giga-byte address space, it doesn't need four gigabytes of physical mem-ory. The term virtual memory refers to the fact that something that is virtual appears to be present but really isn't.
Bynon 1-8

The goal the Virtual Address Extension (VAX) architecture was intended for a family of computers that would span several decades. (Bynon 1-7)
VMS was developed concurrently with DEC's VAX system. The software development—code named Starlet—began in June of 1975. The project plan was to create a new operating system for the Star family of

processors, but also could be extended to support many different environments. The hardware for the VAX system was designed to be culturally compatible with the PDP-11. The software was designed to parallel the hardware compatibility by providing compatibility with the existing operating system, RSX-11M. While the development of both hardware and software was an effort of many people, some individuals should be noted. Roger Gourd led the project and software engineers Dave Cutler, Dick Hustvedt, and Peter Lipman were technical project leaders, each was responsible for a different part of the operating system. They simplified the earlier designs and and made key modifications to the highly complex memory management design and the process scheduling of the system. (DECLit p12, 14)

The design and structure of DEC VMS.

The VMS (Virtual Memory System) was developed simultaneously with the VAX hardware. VMS is a layered system, with the layers as concentric rings. These rings provide different levels of access privileges, with center ring having the most privileges, the outer ring, least. The kernel lies at the center and is the heart of the operating system. (Bynon 2-2)

The Kernel is comprised of three components that perform the majority of the operating system's resource-oriented tasks. The first is the input/output (I/O) subsystem, second is the job scheduler, third is memory management.

The I/O subsystem consists of device drivers (low level, hardware specific programs) and several key system services. The most notable system service is $QIO, it reads to and writes from physical devices on behalf of software requests. All outer layers of the operating system access the device driver through the $QIO system service. Servicing device interrupts and logging device timeouts and errors are chores also handled by the I/O subystem.

The job scheduler is responsible for selecting processes for execution or processing. Since it synchronizes processes for execution by continuously checking process state transitions and process priorities, it is kept constantly busy and performs many duties at timed intervals, such as waking hibernating processes and switching in a new process to be executed when another’s quantum of time has expired.

Memory management is comprised of the swapper and the page fault handler. This is the most complex mechanism of the VMS kernel and has four basic responsibilities:

- To give each process its share of physical memory
- To translate virtual memory addresses to physical addresses.
- To allow selective sharing of data among different processes.
- To shield process data (memory) from other processes.

The swapper’s main responsibility is to help the system completely use the available physical memory, while the page fault handler implements the software virtual memory support. (Bynon 2-3)

Also in the kernel are system services. System services are in fact a group of procedures that perform tasks such as controlling I/O, maintaining resources for processes, and allowing interprocess communication. While most system services are used by VMS are for the use of processes, there are also many system services for use in general programming.

The Record Management Services resides in a layer around the kernel known as the Executive mode. Tasks such as reading, writing, creating, and deleting files are handled by record management services. Like system services, record management is a group of procedures that that hide lower level tasks from the user. These procedures provided at two distinct levels; RMS routines themselves and through disk/tape ancillary control processes (ACPS)Ancillary control processes are separate processes running on the system. They perform the actual mass storage transfer. This helps to avoid access conflicts between processes. (levy 275)

The outermost layer of VMS is the user interface. Known as the command line interpreter (CLI). Although other CLI's exist, the most popular is the Digital Command Language (DCL). The CLI provides a user interface with the operating system and its facilities. Services provided by the CLI call system services, RMS, or external images, including VMS utilities and userwritten programs. As in UNIX other user interfaces (shells) can be developed. (Bynon

2-4)

VMS process control and scheduling issues.

Like UNIX, a program in execution is called a processes. These are selected by the scheduler for execution. They form the body from which all useful work is accomplished. Also as in UNIX a process can create other processes in VMS the parent process is called the creator, and its subprocesses are called descendants. Collectiively, the creator, descendants and its subprocesses, are known as a job. The programs that runs within the context of a process to are referred to as executable images. (Bynon 2-4)

Like processes in UNIX, VMS processes have a state which defines the location in physical memory where instructions and data are stored, as well as the program counter, and contents of the hard-ware registers of the process.

The layers, or rings of protection roughly correspond to access modes. These four modes, known as Kernel, Executive, Supervisor, and User, give the VAX processor its basic protection. Instructions may be executed by aprocess in one of these modes. There are different levels of privileges a processes can have for accessing memory and the executing certain types of instructions depending on its mode. As an example, the instruction to halt the processor can be executed only in the Kernel mode. (Levy p274)

VMS Process scheduling

In the VMS operating system, the scheduler maintains a data structure called the software process control block (PCB). The software PCB is part of the context for every process. It describes the condition of a process at any point in time. It contains the user's privileges, accumulated resource usage information, user name, and so on. The scheduler maintains a record of the condition of processes by using queues of the software PCBs arranged by the proces state and priority. All processes have their software PCB linked onto one of the scheduler's queues. (Levy p303)

VMS uses priority scheduling to decide when processes run. Each process is assigned a priority between 0 and 31. These priorities are defined by the operating system. When a process is ready to run, it is placed on one of the 32 corresponding runnable priority queues maintained by the scheduler for executable processes. When choosing a new process to run, the scheduler always takes the process from the head of the highest nonempty priority queue. The software PCB of each process contains a pointer to its hardware PCB, which is used in performing the context switch.

VMS avoids the problem of highest-priority processes executing forever by changing process priorities dynamically. A process is given a base priority when it is created. The base priority assigned to the user by the system manager. A processes that is in execution can have its priority raised or lowered, depending on its activity.   If a process computes until its quantum expires, its priority is reduced, and it is placed at the tail of a lower priority queue. A process is never lowered below its initial priority. A waiting process's priority is raised, if for example there is an I/O request. Then it would have a greater chance of computing. The events that cause a priority change include:

1. execution (priority is decreases when a process is placed into execution)
2. quantum expiration (priority is decreased)
3. terminal-input or terminal-output completion (priority is increased)
4. other input or output completion (priority is increased)
5. resource availability, wake, resume, or deletion (priority is increased)
6. acquisition of a lock (priority is increased)

As events occur a process' priority gets changed. Interactive users waiting are favored by raising a process's priority following an interaction (terminal or I/O completion) and reducing its priority as the process becomes computational. The largest priority increase happens after the completion of a terminal-input operation.

Only the processes whose base priorities are between 0 and 15 can have their priorities altered. These processes are known as time-sharing processes. Their priorities are never reduced below their base priority or raised above 15. This is in contrast to real-time processes. Their base priorities are between 16 and 31. Because real-time processes deal with time-critical events, they can only be run by suitably privileged users. Real-time processes run until they reach completion, suspend themselves to wait for an I/O operation or until a higher_priority real-time process becomes runnable. They are not held to quantum limits as timesharing processes are.

The scheduler also manages process-state translations. This is done by maintaining 32 queues for processes that are in the executable state but are currently out of memory. This is a separate set of queues from the ones that keep track of processes in the executable and in-memory state. The swapper is responsible for moving process between memory and the disk under command of the scheduler.

Some of the remaining states are nonexecutable processes that are waiting for event occurences, such as page-faults, completion of an I/O operations or the availability resources. These processes are linked onto one of the scheduler's wait queues. As the events that a process may be waiting for occur, the software PCB is moved between queues to reflect the state of the process. Putting processes into different queues according to status helps reduce the work of locating a waiting process when an event occurs that affects it. For example, when a page fault completes, the scheduler can find the waiting process in the Page Fault Wait queue. Its software PCB can then be linked to the appropriate executable queue, based on its priority and whether or not it is in memory. Figure 1 shows the possible states and transitions for a process in the VMS system. The arrows in Figure 1 show the events that cause a process to move from one state to another.

VMS attempts to provide equal time to users by preempting long-running jobs and by changing process priorities to allow other processes to execute. This favors interactive users while also maintaining some real-time system features. (Levy p303)

VMS memory management issues.

The operating system manages the sharing of memory among several programs. It has to maintain the physical separation of memory among the programs. Since programs address specific memory locations, the operating system and the hardware must make each program believe that it is the only program in memory. The system allows the program to operate as if it had been loaded into contiguous physical memory.

This environment allows every program to be written as if it had a contiguous, zero-based physical address space. Time sharing systems must run several programs at the same time to achieve maximum use of the cpu The operating system must provide the logical environment that the program expects.

There are several techniques for hiding from the program its physical location in memory. The simplest technique for allowing several programs to coexist in memory is relocation through the use of a hardware base register. With this technique, each program is loaded into contiguous physical memory.

The CPU hardware contains two special registers: a base register and a length register. By knowing the length of a program, and its base physical address the operating system tricks the program into think it is really at physical address zero. Before a program is started, the base register is loaded with the base physical address. The length register is loaded with the length of the program. To ensure that the program accesses only its memory the hardware checks the length register. The actual physical address is found by adding the value of the base register to the program generated address. this allows for several programs to be stored in memory at the same time. The work of adding the adding the values in the base register and length register is handled by the hardware. Since they refer to an addresses that aren't actual physical memory locations, program-generated addresses are known as virtual addresses.

The concept of virtual addresses assumes a contiguous memory space. However, users write programs and subprograms that can be seen as segments into which a program has been subdivided. Program segments in the system are written as if they were loaded into a contiguous memory space that starts at memory location zero. In order to make segmentation possible the hardware must include a base register and a length register for every segment. Since every segment must have base register and a length register, the hardware must keep track of the appropriate registers for each segment. Since each segment is mapped separately, it can be located anywhere in the memory.

Relocation and segmentation schemes can create a problem for the operating system by increasing the complexity of memory management. Holes are created in memory when programs are removed, either because they have finished processing, or are waiting for an event to finish. This leaves odd-sized holes (fragmentation) in memory. Programs that need to be loaded into memory may not be able to fit in these holes. The operating system must solve this problem by moving segments around so it can compact all free segments into a large contiguous space. The operation of moving segments, and consolidating the memory space is expensive and time consuming.

When segments get moved around in the memory space, their corresponding base registers must be changed so the proper virtual-to-physical translation can take place. To make this change, running programs must be stopped which results in less useful work getting accomplished.

One solution to this problem is paging. Paging is a strategy where the programs and physical memory are divided into equal-sized blocks called pages. This fixes the problem of odd sized memory holes left by varying sized segments. It also removes the need for the entire program to be in physical memory and the problems of fragmentation and compaction. Because both programs and physical memory are of equal size, there is no problem swapping pieces of a program into memory.

Programs are usually larger than a page. So most programs are many pages in length. Paging solves the problem of memory fragmentation, but some fragmentation does occur since the last page of a program may be only half filled. This is known as internal fragmentation, as opposed to the external fragmentation caused by segmentation.

Every program's pages are kept track of by a list of mapping registers called a page table. The items in the page table are page table entries (PTEs), each has the base physical address for one page in the program. Each PTE also has one bit, called a Valid bit, shows if the page is in physical memory. The Valid bit makes it possible to have only parts of a program in memory. If a page is needed that is not in physical memory the hardware calls an operating system procedure that loads the page from disk. With enough address space, programs can be written for as large contiguous address space really existed. The operating system handles all the low level work. (Levy 268-272)

Figure 2 shows an diagram of the physical address computation for a paging machine with N-bit addresses and M-bit pages. The virtual address can be seen as being comprised of two components, a virtual page number and a byte offset for that page. The virtual page number, i, selects the page table entry for the ith page. The table entry contains the actual physical address for that page. The byte offset is used with the physical page address to form the physical memory address of the referenced page.

A distinction that could be made between segmentation and paging is that segmentation makes work easier for the user, while paging makes work easier for the operating system. Segmentation is involves the logical portion of the program, where as paging is deals with the physical portion of the program. Just as a page table entry has a Valid bit, it is possible for segment register to also use a Valid bit. When an attempt is made to use a segment not loaded in primary memory, a segment fault would be generated. With many segment registers, a segment table and many segment table entries are likely to be used. The program can be separated into many segments that can be scattered throughout physical memory. Should the segments be constrained to an equal size, segmentation becomes paging.

Paging and segmentation are not mutually independent concepts. There are some systems that offer both segmentation and paging. In these systems, each segment is comprised of a number of pages.

This makes it easier for the operating system to share and swap space for different users. In a segmentation scheme with paging the virtual address can be comprised of three parts: a segment number, page number, and byte offset. The segment number points to an entry in the segment table. The segment table entry points to the address of the page table for that segment. The page number of the virtual address points to a page table entry in the selected page table. The actual physical address is formed by concatenating the physical page address in the page table entry with the byte offset from the virtual address.

While it seems the extra level of indirection provided by segmented paging would make things more complicated for the operating system. It actually makes the sharing of code segments between programs simpler. In a one-level page table structure each program needs entries in its own page table to point to a shared section of code or data. In a segmented paging structure, a page table could be shared between two programs. Each program's segment table would have an entry that points to a common page table for the shared segment. A shared physical page could be mapped by only one page table entry. Since there is only one descriptor, it is easier for the operating system to keep track of the page. (levy p268-272)

Networking Issues

Networking capability is provided by a family of software and hardware communication products known as DECnet. DECnet allows Digital computers, and computers of some other manufacturers, to communicate through a network. DECnet supports three networking protocols: Ethernet, Digital Data Communications Message Protocol (DDCMP), the original DECnet protocol, and X.25, the standard public network protocol.

A DECnet network can start with two nodes, it can grow to more than 64,000 nodes. The physical link among the nodes is of no consequence to DECnet. It can be linked by Ethernet, fiber optic, satellite, leased line, or another available communication medium. The connection method is invisible to the network users. The nodes in different parts of the world appear to be next door.

Nodes in a DECnet network have a peer relationship: Any nodes in the network can communicate with each other without consulting a controlling node. DECnet control is fully distributed. Distributed control allows each node can be equally attentive to user requests. Since information doesn't have to go through a master node, network efficiency is increased.

The nodes in a DECnet network can operate under many different operating systems and platforms. DECnet, operating on any system, provides the same outward interface: the Digital Network Architecture (DNA). DECnet is patterned after the International Standards Organization's Open Systems Interconnect (OSI) model. The International Standards Organization is the group chartered to recommend industry standards.

Both DNA and OSI models both specify a network architecture in terms of layers. Each layer has a specific function. Other nodes have the same corresponding layers. Two important intercomponent relationships are defined by the layered network architecture: protocols and interfaces.

The differences between the DNA and OSI model are mostly syntactical. The seven layers of the OSI model are:

Physical layer (hardware) - Defines the characteristics for the physical connection, i.e., controllers and device drivers.

Data Link layer (hardware) - Defines a bit- or byte-oriented protocol for information exchange, acknowledgment, error-detection, and retransmission on the data link.

Network layer (software) - Defines a higher-level protocol to provide the multiplexing functions required to route messages between the local and a remote node through the data link. This module also provides network congestion control.

Transport layer (software) - Creates logical to physical links and ensures integrity of the data transfer, including all required error recovery not handled at lower levels.

Session layer (software) - Handles establishment and termination of the virtual connections between two nodes.

Presentation layer (software) - Handles data format translation of the user's data format and the network's data format for such functions as remote file access, file transfer, and virtual terminal functions.

Application layer (software) - Allows users and applications to communicate with (i.e., use) the network. (Bynon 3.9 - 3.11)

Physical links are dealt with at the lower layers, one through four. Collectively they provide a transport service, dealing with the technicalities of how communication between nodes happens. The Transport layer connects the the physical and logical links. All functions above the Transport layer reference logical links instead of physical ones.

Communication between layers of local and remote nodes is handled through DEC's network services protocol (NSP). The creation of logical links, delivery of data, and control messages, breaking up normal data messages that can be broken up and transmitted, and managing the movement of messages is the responsibility of NSP.

NSP can send or receive three types of messages: control, data, and acknowledgment. Logical links are established, maintained, and destroyed by the network services protocol. It does this by exchanging control messages with other NSP modules, or even itself if the programs are on the same node.

Since the links that are made are logical, programs running on a DECnet system see other nodes as direct links. When two cooperating programs make calls to DECnet facilities to initiate a handshake procedure a logical link is made. DECnet calls are sent between the programs that establish an agreement to communicate. An aspect of logical links is important to system security. This involves the fact that both programs must agree to form the logical link. There is a specific order to a handshake procedure. First, the source program makes a call to request communication with a target program. The target node is sent a message, CONNECT INITIATE. The process on the target node can either reject or accept the request. If the request is accepted, NSP returns a CONNECT CONFIRM message. After the link is established, the NSPS can send and receive data and acknowledge messages for the cooperating program, until one of the programs send a request that breaks the link.

The first layer in the OSI model considered to be system dependent is the Session layer. Process to process communication functions are provided by software in the session layer. This type of communication is called DEcnet as task-to task communication. This bridges the gap between the Presentation layer and logical links.

Some task-to-task functions are: receiving connection requests for processes, requesting logical links for processes, sending and receiving logical link data, disconnecting and aborting logical links, monitoring logical links, identifying processes, creating processes for incoming connect requests, and mapping node names to node addresses.

Users can access files that reside on different nodes. User-written programs can create and delete remote files, open and close remote files, and read from and write to records of remote files. Remote file access requires the interaction of two DECnet programs: a source and a target. The operating system running the source program reformats the data to conform to the local file system. Communication is done through the data access protocol (DAP).

Users can connect a terminal to a remote node, and run interactively on the remote node. This allows users on one node to run programs that reside on another. (Bynon p3-23 - 3-27)

Current state of the VMS operating system

VMS became OpenVMS in 1992. Digital changed the name of VMS to OpenVMS to reflect the portability and openness of this operating system. This is not the same as open source operating systems such as Linux. OpenVMS is supported on Compaq Computer Corporation's Alpha series computers, VAX, MicroVAX, VAXstation, and VAXserver series computers. OpenVMS now supports POSIX standards of the IEEE, real-time applications and transaction processing. (DECLit p44-50)

Some new capabilities that make have kept VMS a viable operating system include integration between OpenVMS and Windows NT, Netscape Web Server, Java Development Kit.

The Digital Command Language (DCL) provides users access the OpenVMS software. DCL commands take the form of a command name followed by parameters and qualifiers. Processes can be connected using PIPE. Since Open VMS runs on Alpha, and VAX series computers, you would have to be in just such an environment to use VMS.

It is surprising to find out that in a world where Windows NT, and Unix run the internet, that VMS is also being used to support web sites. Northern Light is one example.

Bibliography:

Bynon, David W.   Mastering VMS Pennsylvania: Professional Press Inc., 1990

Levy, Henry M., and Richard H. Eckhouse, Jr. Computer Programming and Architecture: The VAX Second Ed. Digital Press, 1989

Kathie Peck VAX Open VMS at 20 Digital Equipment Corporation 1997