upcoming discussion. It is the relationship between the user and the system manager that is the real concern of system management philosophy. So far we have talked of access, and how this might be handled by the two philosophies. But there has been one big, glaring, unanswered question. It's clear what the two philosophies are. But wherever did they come from?

It is my contention that they arose out of policies, not for system management, but for user management! Because the whole central issue of systems management philosophy is whether the system manager is serving the users or enslaving them. From whence comes this radical conclusion? I could find no direct corroborating evidence for my conclusion (indeed, my thesis!). In my research, I was unable to find any other work dealing with this issue. I must therefore present the original analysis, based on the data I have presented so far, which led me to this conclusion.

We clearly have two system management philosophies: one encouraging open, responsible access and the other based on an innate paranoia. Unix philosophy arose in the research environment; VMS in the business world. But why did they arise? It may at first seem obvious: Because in a research environment you need sharability and ease of access, while in a business environment you need high security
and reliability. This, however, is to describe the illness and not the prescription. Given those requirements, why the existing approaches? What is it that the two philosophies are trying to do?

My thesis is that they are concerned with user control; that is, the achievement of the respective goals for the two environments is determined by the placement of control over the system. An open, friendly environment is achieved by placing the majority of control over the system in the hands of the users. A secure, highly controlled environment is achieved by making the system manager the locus of power.

There's so much in this idea that it almost drowns in its own implications. The only thing to do here is to look at it in small chunks, then try and fit those pieces together to form a more comprehensive picture. To do this we will return to examining our two philosophies, this time in light of the foregoing knowledge, and see how well they support my thesis.

We'll look first at the VMS system, because that's the classic approach to systems management. Computers reached their heyday in business environments, and there, for the most part, they have remained. We've already examined the security issues a system manager must deal with. But the thing is, in order to implement that security, he must have
control over what the users can and cannot do. Even if the users want to be negligent and leave their files world readable, the system manager has another task entirely; namely, make the system secure. He must therefore prevent the users from compromising the system and, failing that, repair the damage.

The thing he is serving throughout all this, the entity to which he pays allegiance, is NOT the user community. This fact cannot be overemphasized. No, the system manager does homage to a much bigger concern: the needs of the institution which he serves. It is the institution's policies that he follows; in many cases, the users may complain about his actions, but that's none of his concern. He has a job to do... maintain the system. If part of that job is serving the users, so be it, but it is always (or should be) clear in his mind that his first priority is the carrying out of company policy.

This is it. This is the source, the wellspring, of VMS philosophy. All power to the system manager. The users must be guarded against, sanctioned, monitored. You see this EVERYWHERE in VAX/VMS design and development. Disk quotas. Access control lists. It is almost a divide and conquer operation. It was noted earlier that it is difficult to run VMS as an open system. Indeed, if it's a pain for the system manager to relinquish control (for that is what he is doing, make
no mistake about that!), then it must be nigh impossible for a user to circumvent controlling aspects of the system.

This was evident in resource management too. Don't let the users get away with printouts. Monitor all the print jobs. Be able to account for every dotted i of CPU time. Not, mind you, that this is a bad approach. I emphasize again... you NEED this in a business environment. But now, now we see where the VMS philosophy comes from. Control. That's the key. That's the issue around which the whole thing turns. Control... and where it's located.

And if it's the users who have the majority vote, why look what happens. If the system manager is serving THEM first, you have a complete turnaround! If the user's want to compromise the system... fine, let them compromise it. It is, after all, their system. If they want to bog down the CPU with inefficient word processors rather than upgrade, fine. Again, it's their system. And that's a distinction between the two philosophies that's so important it's staggering. If a user complains about limited batch queue time in a VMS-ish installation, chances are he'll be told that the policy is set to effect maximum throughput, not cater to his individual needs.

In a Unix-ish system, on the other hand, we would probably find something different. If the complaining user's problem is local to
himself, the system manager might give him extra CPU time or otherwise work to assist him. (This would happen in VMS philosophy only if the user required the resources, and even then it'd probably have to be approved. At my alma mater, it was literally impossible to get privileges under VAX/VMS, "because of security constraints"). If it's a general complaint from all the users, the system manager might go ahead and change the system, or if it's a problematic situation he might solicit user input. (When's the last time you saw user input solicited on a VMS-ish installation?)

I think by now you know what I'm talking about. If the philosophy split I'm talking about wasn't clear before, it should be now. Most Unix users will recognize the situation right away. It's not configuration of hardware and software that determines how the computer will be run. It's who really has the say in what is done. Because when you say 'computer system', really, you're talking about a lot more than the hardware and software. You're talking about people. In the end, they're the most important part of 'the system.' This is why I used 'Philosophy' in the title of this thesis. Because systems management isn't half as concerned with the computer as it is with the people who use it.

And so MANY installations, it seems, don't understand this!
From the system managers to the users. I've got a whole BOOK here
detailing massive computer mishaps [MCPF]. How did they happen?
HUMAN error!

Suppose a system manager discovers a lot of CPU time on her
VAX/VMS system is being eaten up by batch jobs, So, she decides to
lower the priority on the batch queue to let the interactive users, who
outnumber the "batchies", get better response time. This would seem to
be a good, adequate solution... in terms of the computer system alone.
But look what happens to the user community:

Suddenly, the people who were already complaining about
slow batch response are screaming because it's now twice as slow. The
system manager, of course, will inform them that this is to favor the
interactive users. But wait. Suddenly there are batch jobs in the queue
for twice as long as before. Suppose, on the old system, batch access
peaked around 3:00 and interactive peaked around 4:00. Now, however,
batch access will S-t-r-e-t-c-h over into the 4:00 time slot. And voila,
the system backplane turns to molasses.

Of course, the system manager should have seen this. But
that's my point. Too much system management takes place that focuses
on the computer rather than on the system. An alternative solution to the
above dilemma might be to actually increase the batch queue priority.
Sure, things slow down for a few moments, or even an hour or so. But the determining factor is, which are you benchmarking by: system performance, or the satisfaction of the user community?

This is a contrived example, I realize, but it is contrived to illustrate a point. The point of attention to the totality of the system. And what attention you give the totality, obviously, depends on what philosophy you are following. And that is determined by where control over the system is placed.

The issue of control may be centered on who the system manager is serving, but it is by no means irrelevant to how much control the individual users have over the system. The system manager by definition has maximum control. He can stop that sucker dead if he wants to. But for the users, access and capabilities on the system are of crucial importance. We’ve already looked at system access... the determination of which is the province of the system manager. And looking at how VMS and Unix philosophy operate, suddenly the real issue behind systems management becomes clear.

The big question of where the two philosophies came from has been answered. There’s still a smaller question, though, that is unanswered. I’ve shown that what philosophy you have is determined by where the control over the system lies. But now there’s the issue of
how that control is obtained. Access is obviously one method, but that's not really the whole picture, which was why I presented it as background rather than as a case in point for the exposition of the conclusions I make in this chapter.

Control over the system is obtained, and can be maintained, in only one way. And really, when you consider the fact that the system manager is god, it's an obvious way. Resource management. Once your philosophy is determined, it is the system manager's manipulation of available resources that will carry that philosophy out.

I'll take the time now to clarify something I have been doing throughout this paper. I've been detailing things in terms of Unix and VMS philosophy, black and white. I've been doing this to bring out the differences between the two philosophies, and establish a high contrast. But in reality, there need be no clear-cut distinction. You can have a 'mostly VMS-ish' installation or a 'mostly Unix-ish' installation. But there may be elements of the opposing philosophy in each. You can have an institution where tape access is controlled like nuclear strike capability, but the users can upload and download files from their micros all they want. Or, you might have an installation that encourages file sharing and cooperation of the users— but enforces stringent disk quotas due to limited storage space. There can be a blend, a comingling
of the two philosophies. They are not mutually exclusive by any means.

I have been using Unix and VMS environments to typify the two philosophies, but something that must be brought out again, lest it be obscured, is that the operating systems by those names have nothing to do with which philosophy is in force. I merely use the operating system names to designate the respective philosophies because the respective operating systems were designed with the respective philosophies in mind.

It is, albeit, difficult to run VMS in a non-secure fashion. However, as mentioned in Chapter I, Unix is quite a secure operating system when given half a chance, perhaps as much so as VMS. It provides, in many ways, a higher degree of control than VMS... and hence, more effective systems management is possible.

I've shown how each philosophy arises, and I've shown the method by which each is implemented. I'll turn now to the specific issue of resource management within the system, and show how this is the crucial step in the implementation of either philosophy.
"No matter who you are the basic truth is that your interests are nobody else's concern, your gain is inevitably someone else's loss, your failure somebody else's victory."

--Michael Korda, *Power*

**Chapter V**

**Mob Rules**

I said in the last chapter that the key to successful system management was effective resource control. It's through control of the system's resources that the system manager actualizes whichever management philosophy is extant. And, depending on whether control of these resources is centered on the users or the system manager, a Unix-ish or VMS-ish environment, respectively, will result.

The system manager obviously has full control over the system... this is a *sine qua non* of her position. It is therefore her granting and denying to the users, to various degrees, the resources available on the system that carries out her system management policy. There are three main categories of resource which the system manager must administrate: physical resources (tapes, disks, printers, etc.), operating system resources (memory, CPU time, etc.), and files. [VSM, p.5-1] Access to the system *could* be considered a resource, but while unquestionably a system management responsibility, it is not a direct
part of the system. Rather, it is the gateway to it. This was already covered in Chapter 3 to contrast and clarify the two philosophies.

Physical resources, as previously mentioned, are things like disk drives, tape drives, printers, etc. In short, by and large physical devices. Aspects of system management pertaining to access to these devices was covered in Chapter 3. I'm going to turn now to the more involved side: management of these resources. To do so, I will once again contrast the tools and capabilities of the two operating systems, thus bringing out the different philosophical approaches.

VMS is probably the leader in terms of standard operating system features for control of physical resources. The MOUNT utility handles the mounting of tapes and disks for access. It has a gaggle of command qualifiers, or options. Disks and tapes can be mounted by anyone with access to the device. However, they can be mounted shareable or nonshareable, system or nonsystem (only a user with a "system ID" can dismount them), and different protections for the various access fields (discussed more thoroughly below under files) may be specified as well [VDCL, MOUNT]. Such attention to various access modes for these devices reveals the recognition on the part of Digital of the varying degrees of resource control users may need.

Note that a volume (tape or disk) that is mounted system and
set for system manager access only is about as exclusive as you can get. Note similarly that with each step down from this level of protection, the system manager surrenders some of his control over the volume, and by extension over its use. If she allows world reading of the volume, users may copy data from it. If she allows world writing, users may change data on the volume. There is regulation to some extent by privileges (see below), to prevent the rampant manipulation of any portion of the volume by any user. Nonetheless, at the bottom level of a disk being open to reading, writing, or dismounting by anyone, it can be seen that the system manager has very little control over what happens to the disk.

Before our discussion of VMS resource management can go any further, I'm going to have to explain two very important capabilities of the VMS operating system-- two capabilities that have helped to make this OS an industry leader. The first is privileges; these were mentioned briefly in Chapter 2, but it is time now to look at them in some detail.

"Privileges restrict the performance of certain system activities to certain users. These restrictions protect the integrity of the operating system’s performance and thus the integrity of service provided to users. You should grant privileges to each user on the basis of two factors"
This description is a fairly accurate one as far as the function of privileges in the VAX/VMS environment is concerned. The two criteria for privilege assignation are: "Whether the user has the skill and experience to use the privilege without disrupting the system," and, "Whether the user has a legitimate need for the privilege" [VSM, p. 6-7]. Clearly, then, according to VMS philosophy, least access rules in terms of enabling privileged users, to say nothing of resource management in general.

The thirty-odd privileges under VMS determine what a process can and cannot do, both to the operating system itself and to other users. Internally, they are stored in a bit vector located in various data structures, including the process header and the user’s UAF record entry [VVI, p. 488]. Enabling of privileges may be accomplished with the DCL command SET PROCESS/PRIVILEGES [VDCL, SET] or by an image (i.e. program) calling the system service SYS$SETPRV [VSH, p. 348]. A process may only enable privileges it is authorized to use; authorization is determined by the user’s UAF record, which is, in turn, configured by the system manager.

Privileges are very important to the operation of VMS, as they are intended to provide a graduated access to the system. Though, as I mentioned above in Chapter I, this looks better on paper than in
practice. It is important to understand privileges under VMS, because they really constitute a resource by themselves.

The second operating system feature that needs explaining is the Access Control List, or ACL. This is the VMS version of a concept that has been around for some time. Essentially, an ACL is a list of identifiers and the permitted accesses for each identifier. An identifier may be, and usually is, a user ID (actually, UIC). It may also, however, be part of the system rights list database, which is maintained by the system manager. Clearly, some explaining needs to be done here.

First, what an ACL does. It provides a means of granting or denying access to system resources on a per-user basis. This is a VERY powerful feature, as it overcomes the handicap of having to deal with groups of users. Moreover, should the opposite be desired, it allows the arbitrary grouping of users together. All this is accomplished with the concept of identifiers. An identifier is, simply put, either a user ID (UIC) or an entry in the system rights list. The system rights list is simply a database which contains the names of all users on the system and what identifiers can be associated with them. With this setup, ACLs may be assigned not only to files (where they see the most use), but also to devices and other physical resources. Some examples should
make all this a little clearer.

Let's say that a system manager doesn't want one of her users to have access to a certain tape drive (for whatever reason). She then places an ACL on the tape drive with the SET DEVICE/ACL command [VDCL, SET]. She can then restrict his use in one of three ways: she may permit reading to the drive, reading and writing to the drive, or control over the ACL on the drive. These capabilities are called, in ACL parlance, READ, WRITE, and CONTROL.

Another example. Let's say a system manager wishes to give several secretaries access to a disk drive. He can create a separate ACL entry on the disk drive for each one, or, he can create a rights identifier in the rights database. The identifier would "belong to" each secretary. Then, the ACL for the disk drive would merely specify the access, for that identifier. This way, access for a group of people can be manipulated through one system entity. It is this powerful feature of VMS, introduced with release 4.0, that has helped push the operating system to the top of the market.

Now to bring all this back round to resource management. In addition to the options available with the MOUNT command, then, additional device control is available through ACLs and privileges. I've shown how ACLs would limit device access. Let me show how
privileges might work, to supplement the earlier mention of disk access.

Suppose a system manager has an LN03 laserprinter hooked up to the system. However, he only wishes faculty to be able to print on it. He creates an identifier for faculty, then places an ACL for write access on the queue. Fine, the faculty and the system manager now have access, but no-one else does. Suppose furthermore that he does not want the faculty changing queue characteristics, but that only operators should be able to do this.

VMS provides a built-in privilege for operators, called OPER. Each system operator’s account has this privilege, and it allows them to answer MOUNT requests, respond to user questions, and, among other things, manipulate print queues. So the queue will be ACLed for faculty writeability, while the system operators will have OPER privilege to allow them to change queue characteristics as needed.

With these tools at hand, then, it is possible for a system manager to maintain a high degree of control over the system. And as we have seen, this is a necessary component for a successful VMS philosophy. In Unix environments there is a much different situation.

Unix has no such granularity of control associated with it. All devices are always available to the system. Filesystems, which are
"pieces" of disk drives, are mounted and unmounted; however, this is normally not a frequent occurrence save for a removable media disk drive. Tape drives under Unix are not treated similarly to disks at all, but rather as different devices in their own right. There are a number of utilities for manipulating tapes; tar, cpio, dump, etc. There are also commands for controlling filesystems: df, umount, newfs, etc. [UPRM]. However, individual user control under Unix is MUCH harder than under VMS. Now, to understand how it can be accomplished for devices, an explanation of "groups" is necessary.

In Unix, there is a file called /etc/group. This file contains a list of "group names" followed by people in those groups. It is, in this respect, rather like the rights database under VMS. A user may be in several groups. Some versions of Unix require explicit switching between groups; in most, whichever program you are using does it for you. In any event, it is possible for a program to find out, at any time, a user's default group (the one specified for him in /etc/passwd) and his current group. All files have a "group" protection field; various access types based on this field may be specified (see below). [USS, p. 32]

In VMS, all the system manager has to do is put the user in the rights database and slap an ACL on whichever device she wants restricted. Alas, no such quick fix for Unix! Instead, she must place all
users who are to have/not to have (depending on the degree to which she wishes to limit access) read/write/or whatever access to a given device in one group, say disk2. Then, she must either make the filesystem disk2 accessible only to members of group disk2, or, she must modify each and every program that will access disk2 so it checks to see if the user is in group disk2 or not. Alternatively, she could modify the command interpreter, but the best method would probably be to create an SUID or SGID program (see below) to check the user's permissions, then perform the access. Note that these methods may involve patching the operating system kernel, not an uncommon practice in the Unix world. [USS, p. 136-137]

Regardless, it should be apparent that controlling device access under Unix can be something of a pain. The process is even worse for print queues. Clever users can easily circumvent the line printer daemon, and jockey queue entries (theirs AND others!) to their heart's content (usually scrambling the queues to the point of necessary resetting, though it should be noted much of this has been fixed under System V and 4.3BSD). The tape drive access fault has already been mentioned; when left open for access by anyone, if one user is using the tape drive, another user can just toss his commands in alongside, undoubtedly creating havoc. There is no lock manager as in VMS to
solve this problem.

This whole situation, however, is just what we would expect in the respective operating systems. VMS provides a battery of controls for the system manager... in addition to the security controls, there are also numerous accounting utilities to keep track of system resource usage. Unix, on the other hand, makes scant provision for highly controlled access. It is simply assumed that the users can take care of themselves. As Dennis Ritchie (he wrote Unix, remember?) says, "The first fact to face is that Unix was not developed with security, in any realistic sense, in mind; this fact alone guarantees a vast number of holes." [USM, p. 17-1] Not, mind you, that whatever VMS does Unix can't do. Unix can do anything VMS does... the question is simply one of how much effort you are going to expend patching the system into the shape you desire. Here we see it quite clearly: VMS design focuses on giving control to the system manager, while the innate Unix design distributes system control to the user community (virtually in spite of the system manager!).

In addition to physical devices, though, there are still operating system resources. The major ones in this category are memory and CPU time. And, as we would expect, VMS is on top of the situation with extensive facilities for determining not only how much a
user can use, but for tracking his use as well.

I mentioned earlier that the UAF record for each user is a very important data structure in VMS. It is undoubtedly THE key data structure, so much so that a system without a UAF cannot function and will not boot properly. [VSH, p.437] The UAF specifies, for each user, how much of the various operating system resources a user is allowed to consume. There is a plethora of things to specify, too: Direct I/O limit, Non-paged pool, flag clusters (semaphores), mailboxes, open files, and, of course, CPU time and virtual and physical memory, and the list could easily go on (But wait, there’s more...!). Each of these can be honed and fine-tuned for each user. They can also be honed and fine-tuned for the system as a whole through use of the SYSGEN utility. This utility allows the specification of certain system defaults, most effective upon boot time but some effective immediately. The SYSGEN utility functions similarly to the Unix configuration files. Unfortunately, a thorough discussion of system initialization is beyond the scope of this paper (maybe for my Master’s thesis...). In any event, suffice it to say that CPU time and all other operating system resources are under complete control of the VMS system manager.

I am not going to belabor this situation further, though much could be said about it. The reason I am not is because I am contrasting
VMS and Unix philosophy, and what I am about to say is more than enough to knock your socks off. VMS has such extensive user controls that half the internals orbit around them [VVI, p. 22].

Unix has none.

That's right, NONE! Here, here we finally can see the difference between VMS and Unix philosophy in such black and white clarity it is dazzling. There are NO resource controls in Unix! You can consume memory, consume files, consume I/O, all to your heart’s content. One user can bog the system down into the LaBrea tar pits if she chooses. The only limitations are those imposed at system configuration and boot time, and those aren’t much! [USM, Config] In Unix, the system manager has no control over operating system resource consumption. "The problem here is... in lack of checks for consumption of resources. Most notably, there is no limit on the amount of disk storage space used..." [USM, p. 17-2]

Oh, the operating system can be patched to include controls (some chaps in Australia did it for disk quotas, a system that is still buggy today by virtue of this piggybacking), but then as I mentioned, you can patch Unix to do almost anything. The thing to see is that this is certain proof of my thesis: there is simply no intention for high-level system manager control in Unix. (It may well be for this reason that
Unix has taken so long to penetrate the business market, despite its advanced features.)

Don't, however, get me wrong. Unix may be shy on controls. It is not shy on accounting. In fact, the Unix resource consumption accounting is actually better than that found under VMS! This is due, however, to the large array of "Unix tools" available; these can be combined and recombined to into a truly formidable system accounting package. As distributed, Unix sa (system accounting) is slightly less user-friendly than the VMS ACCOUNTING utility. However, you cannot shuffle accounting data under VMS with nearly the dexterity possible under Unix.

The last area of resource management is files. I saved this for last because it is by far the biggest. There are exactly two types of files I need to talk about: programs and data (Unix special files class as devices and were covered above). In order to present a full discussion, I will first flesh out the context in which files live on each system, then go on to discuss the various access methods and protection capacities for each system.

Taking VMS first again, this time because it is simplest, disks in VMS are formatted volumes capable of supporting a directory hierarchy. The top level, or root directory, [000000], usually contains
the bitmap for the disk, a list of bad sectors, a list of unreferenced files, etc. In addition, a disk may contain a bootblock if it is a system disk, and a quota file if disk quotas are in effect for that volume. The directory system, FILES-11, uses a linked block structure, and supports multiple file links as well. [VSH, p.372]

Files in VMS may be of various types. Associated with each file is sundry information such as creation date, file owner, time of last modification, size, and of course the filename. The owner of a file is designated by a User Identification Code, or UIC, a pair of octal numbers, which usually corresponds to that user's identifier in the rights database. The first octal number is the user's *group number* which as you may suspect is nearly identical in conception to a group under Unix (which was around long before VMS...). The second octal number is the user's *member number* which indicates who she is in her group. The UIC has one important function: it is used on all file access checks for files that do not have ACLs associated with them. Before showing how an ACL is associated with a file, I'm going to explain UIC protection codes, since these are important to Unix too.

A file has four classes of possible access: System, Owner, Group, and World. The System class refers to anyone having an octal group number less than 011, or any user having system privilege
(SYSPRV). The Owner class applies to the user who’s UIC matches that of the UIC recorded in the owner attribute of the file. The Group class applies to all users with the same group UIC as that of the owner of the file. The World class applies to any user.

For each class, then, there are four protection-attributes: read, write, execute, and delete. Read access implies the ability to read the contents of the file, regardless of what they are, and thus gives the ability to copy the file. Write access allows changes to the file, though not deletion. Delete access allows deletion of the file, then, while Execute access allows the running of the file if it is a program image or DCL command procedure. As a side note, read and execute work differently for a directory: read allows listing of the directory files, while execute allows only access to a specific, already known, file. Note that additional access checks will then be applied against the referenced files; the directory is merely a buffer between the accessing program and the target file.

Putting this all together, then, as an example the default protection on a VMS system might look something like:

S:RWED,O:RWED,G:RE,W:RE

Which means that system class has read, write, execute and delete
access to file, the file's owner has the same, but group and world classes are limited to read and execute operations. Default protection refers to the protection assigned a file at its creation; it is set according to a predetermined value, but may be changed by the user. It is fairly obvious from this, then, how UIC protection works under a VMS system. What can be done to a file is determined by the access set for the various classes.

Note, however, that there are some major problems with this setup. If a user wants to create a data file accessible only to users X, Y, and Z, he cannot do so easily because setting G:W would allow any other group member access also, and setting W:W would be inviting trouble from any user at all. S:W would be useless, too, unless the other users were all privileged users. Worse, it turns out that you cannot just set W access; you must set RW so the VAX I/O routines can read the indexes into the file and thus know where to look on the disk for the data.

Another problem is that D access for a file will not by itself allow its deletion by that category. The deleting user must also have D access to the file's parent directory; this may add some security, but it can also be a real pain.

ACLs have changed all this, however. In an ACL, specific
user identifiers may be indicated, with appropriate access for each. If there is no ACL entry for a user, protection defaults to UIC-based access. In addition to RWED, which must in ACLs be spelled out, there is also CONTROL access, which allows altering of the ACL entry, and should obviously not be given lightly. There are various other options as well. (See the VAX/VMS ACL Editor Reference Manual for more information on ACL use.)

The last VMS file control we need to consider is the lock manager. Considerable time and effort have been invested in this system service by the Digital design force, so it is somewhat surprising that it can often be such a pain. The lock manager is designed to enforce exclusive access to a non-shareable VMS file. (Files are by default non-shareable on creation.) As long as the file is opened for access, it will prevent access by another user.

Unfortunately, there are some minor bugs here too. Opening a file with the DCL OPEN command [VDCL, OPEN] will call down the lock manager on it. Then a command such as

```
DELETE FILE.A
```

will result in a message like:

```
%RMS-F-FLK, file locked by another user
```

which is what would be expected. In case a file is accidentally left open
or improperly closed, such as by failure to deassign a logical name or failed DECnet access, VMS provides the UNLOCK command [VDCL, UNLOCK]. However, if UNLOCK is issued immediately following the above sequence of commands:

```
%RMS-F-NOTLOCKED, file was not locked
```
or a similar message will result. I am not aware of the reason for this, and have seen no articles on it, but while an error in such a case may be legitimate, nonetheless either the error message and/or the manual entry should be more explicit about why an error is issued or what UNLOCK may and may not be used for. It is possible to disable lock manager action on a file by specifying it shareable, though this can often involve some jockeying with the OS when dealing with shareable executable libraries.

We can see from these various features that VMS provides the user and the system manager with a number of good file access and file management controls. Users may be assigned disk quotas with the DISKQUOTA utility; there is an entry for each user on each disk, which limits the amount of file space the user may consume (thus overcoming the problem in the original Unix system...). However, there is a small and annoying bug here too; though a user has 40,000 blocks of quota (block=512 bytes), this applies only to the disk where her login directory resides (default disk, specified in the UAF). If she is to create
files on another disk volume, she must have a quota entry in the disk quota file there, too. This results in a situation where a user has legitimate write access to a directory on a disk different from his default disk, but cannot create files there unless he has quota on that disk.

We can see from this, and from some of the other bugs in the file control capabilities of VMS, that again VMS philosophy is at work. The entire system is oriented toward restricting access, and limiting resource consumption. Attempting to provide for an open environment is difficult, and in many ways requires much more effort on the part of the system manager than simply leaving everything secure and difficult to work with (assuming an insecure environment is desired).

There is one particular area where VMS suffers greatly: If a user has a data file which he wants to be accessible only from a given program, so that any other users can only read and/or write the file while running that program, he is in trouble. There is, under the current version of VMS (4.5), no way to do this. He must leave the file W:RW or G:RW, so that the program has access to it when run by other users, but this allows a user to use a program other than the legitimate one to manipulate or corrupt the file. There is no solution such as ACL= (IDENTIFIER= [PROGRAM], FILE= DATAFILE, ACCESS= READ+WRITE), though the ability to ACL a file for access by a given
program would be one possible solution.

The reason this presents such difficulty is that when a user runs someone else’s program, it still runs under her user-id. Thus, she can access all her own files, but none of the files of the program’s author unless she herself has access to those anyway. There is one way around this situation; the program must be installed with privileges. This means that when the program is run, it acquires privileges associated with it. A program is installed by the system manager with the INSTALL utility; it is logged in the file VMSIMAGES.DAT in the system manager’s directory. Thus, installing a program with READALL or BYPASS privilege, either of which allows access to any file on the system, would solve the problem. [VSM, p. 8-1,3]

Of course, this opens up a security hole big enough to float a battleship through. How many institutions will permit a system manager to install a user-written utility that, through a trojan horse, could corrupt or subvert the entire operating system? Of course, the user-written code could be checked, but that would take time... Indeed, the system manager himself has problems, because any privileged software he writes must be thoroughly debugged, lest it irreparably corrupt the system, and this applies even to a relatively trivial task.

We can clearly see then that a VMS philosophy is strongly
present in the VAX/VMS file resource controls and capabilities. But it is with some surprise that we find out that while Unix was not designed with security in mind, in many ways it is far more flexible than VMS.

Files in Unix all live in a filesystem, which is simply a *partition* on a disk. Partitions are logical rather than physical, though where and how they reside can be controlled with reasonable granularity. This grouping information is all contained in block 1 of the disk, called the *super block*. Also, *immediately* following the super-block is the i-list, which contains the list of *i-nodes* for the filesystem. The i-node is a table that contains the file attributes for a given file; where it is, when it was created, who owns it, its size, etc. The file list, which describes the location of the blocks on disk for a given file, allows indirection to provide a theoretical maximum file size of 16,842,762 blocks, though luckily Unix imposes other limits long before this. A directory under Unix is somewhat different than VMS; it contains a list of file-names followed by the i-node number for the i-node associated with that file. A multiple link to a file is easily created by assigning two or more filenames the same i-node. [USS, p. 104,6]

File protection under Unix is similar to that found under VMS. Each user has a user-id and a group-id, the group-id being determined in the file /etc/group. For each file, there are three
permission classes, not four, and three possible permissions for each class. The classes are u (user), g (group), and o (others), which correspond to the VMS classes Owner, Group, and World. The file protections for each class are r (read), w (write), and x (execute). Unlike VMS, w access for Unix allows deletion of the file. Execute access works the same, both for normal files and for directories. Read access for a directory allows 'directory search', discussed above for VMS. File protections are changed with the chmod command [UPRM, chmod] and the default is set with the umask command [UPRM, umask]. The access permissions are compared against a process' effective user-id and effective group-id to determine valid access. Normally, the euid and egid are the same as the process' real uid and real gid, as determined in /etc/passwd and /etc/group.

There are no ACLs in Unix. Instead, a very subtle and terribly powerful method was developed for allowing special access. This is the famous SUID/SGID system (set user-id/set group-id), now patented by AT&T. A file that is set SUID (applicable only to executable files) will, when run, cause the process running it to have the effective user-id of the program's owner NOT the user running the program. The effects of this are might and manifold. [USS, p. 20,1]

If a program A, owned by user 1, is set SUID, then any other
user running the program will have user 1’s euid and corresponding access to all of user 1’s files. At first, this may seem an insane security compromise. Closer examination, however, reveals that there is a subtle control involved: program A does only what user 1 tells it to. It will therefore access only those files user 1 wishes modified. And, the only way another user can get to a file with access limited to user 1 is through program A! This method overcomes ALL the difficulties of the VMS privileged software situation. A similar situation applies for SGID, only the group rather than the user id is involved, and the protection check is made against the group access field for a file.

The SUID/SGID concept drives all of Unix. Privileged system software is written with SUID root, though various SGIDs may be used. Device access may be limited in this manner, by assigning appropriate protections to special files and running SUID/SGID programs for those files. Unlike VMS, Unix uses no privileges. Instead, user root has access to everything, can delete any file, change any aspect of the system. [USM, p. 17-2]

SUID/SGID opens the doors for a number of trojan horse schemes, but luckily most of these have been defused. In addition, the mount program, which mounts filesystems, runs SUID root but performs no special checks on the filesystem being mounted. [UPRM, mount].
As a result, a user could place SUID root files on the filesystem to be mounted, mount them, then run them... effectively becoming the super-user! The only easy way around this is to forbid non-privileged users to use mount, but this is against Unix philosophy. [USM, p. 17-3]

Disk quotas, as mentioned previously, are available under Unix as an additional patch. However, there are a number of problems, chief amongst them being the issue of files with multiple links. If the original owner of a file deletes it, and it has multiple links, the link is removed, but the file with its links remains, and the quota for the extant file is still missing from the original user's quota allocation.

It's pretty obvious from all of this that Unix philosophy is in full force for Unix systems as well. Under VMS, strict attention is given to providing high control for the system manager over privileged software. In Unix, however, there is little such attention, to the degree that some early bugs in the Unix system were pretty severe. In Unix, much control is given to the users, allowing them to write SUID and SGID programs, and even not originally providing for user file consumption. Furthermore, file protection shows that little difference was made between writing and deleting, and the existence of only three access classes reveals the ease and sweeping generalities of Unix, with the focus being on individual user customization.
"Confusion now hath made his masterpiece!"
--Macbeth 2.3

Chapter VI
Shutdown

Time for concluding remarks at last! (I don’t at all relish the thought of fighting troff macros to get this printed...) I’ve tried in the preceding pages to give a good overview of both VAX/VMS and Unix operating systems, comparing and contrasting both in their various features. More importantly, though, I’ve shown that there are two radically different system management philosophies. Each philosophy is determined by the placement of control in a given system; with the users or with the system manager. The first, a user-oriented philosophy, is typified by the Unix operating system, which was designed with such placement of control in mind. The second, system-management centered philosophy, is typified by the VAX/VMS operating system, which was designed with total control given to the system manager and relatively little to the user community. I then chose to call each philosophy Unix or VMS, though this says nothing about either the operating systems or which philosophy is in operation.

I showed how Unix philosophy is not very concerned with
security issues, and is in many ways loose in terms of access to the system. Further, I showed how an installation operating under Unix philosophy will be generous with any computing resources it has. I then went on to exemplify how the Unix philosophy pervades the design of the operating system. Much control is given to the users, including the writing of semi-privileged programs, even though this often poses some risk to the system. The operating system is targeted largely at research and some academic environments, where file sharing is encouraged and security and reliability are not critical.

At the same time I was doing all this, I showed how VMS is designed with reliability and security as the two main design goals. I showed how an installation operating under VMS philosophy will concentrate on controlling access to the system, and accounting for all resources consumed, in addition to keeping checks on that consumption. I detailed how VMS provides extensive capabilities for scaled access, thus achieving a high granularity of security to meet diverse computing needs. The operating system is targeted mainly at a business computing environment, where security is a prime consideration and reliability is essential. As a result, it is often less flexible than Unix.

There were a number of issues I was unable to cover, such as the question of hacking and operating system security considerations.
Also, there are a number of other things which make VMS different from Unix, all springing from which philosophy governed the operating system design. Such things as file formats (VMS has a gaggle, while Unix uses stream files), network access (Unix has a lot of bugs here, but VMS DECnet is very reliable and, in many ways, an industry standard), memory protection (the Unix kernel is usually readable; the VMS kernel is definitely NOT), and terminal control (VMS limits itself to a specific number of terminals unless extensive programming is performed, while Unix allows user specification of terminal characteristics), and general user friendliness in the environment (VMS is very user-friendly, save perhaps for the error messages; Unix is not, and has few explicatory error messages at all).

In conclusion then we can see that which philosophy is dominant at an installation is a prime consideration of systems management. Both philosophies have benefits and deficiencies, but both require a reasonably strict following along the chosen path to insure success and contiguity in systems management. To this end, it should be obvious that mere technical training for a system manager is not sufficient. She must also understand the consequences of both views, and their impact on her system management, not to mention how the consequences of her actions affect the actualization of the respective
philosophies.

Unix is a rising star in the operating system market... but this may prove a mixed blessing. There are numerous faults in the system intimately connected with the design philosophy. A Unix environment run under a VMS philosophy may present continual surprises for the system manager if he does not understand the reasons behind relative insecurity in the Unix system, an understanding which is a key to the effective administration of Unix.

VMS on the other hand is clearly an operating system of choice for the business environment, or indeed any situation where security and reliability are major considerations. Its use in academic settings or other places where much file sharing and program development must take place is problematic. The difficulty in overcoming innate VMS strengths in the occasions when they prove hindrances is something for thought, to be sure.

In the end, though, policy and the effective philosophy come from management. It is as important for upper management to understand the issues of the two philosophies, and their consequences, as it is for system managers. I hope I have been able to shed some light on a topic previously left to the dark and shadows, and that systems managers everywhere may have a better understanding of just how what
they do affects the user community. Or vice-versa. :-)}
"If you copy from one author, it's plagiarism. If you copy from two, it's research."

--Wilson Mizner

Appendix A
References


References


About the Author

Paul Frommeyer is a recent graduate of Ball State University in Muncie, IN, where he worked as an undergraduate teaching assistant and acted as a Unix and VMS system manager for the Computer Science Department. He holds a BS with honors in Computer Science and Philosophy, desperately wants to work for Digital Equipment Corp., and currently makes his residence in Cincinnati until he can find a position in VMS or Unix system management.

He also enjoys posting frequently to Usenet, drinking Guinness beer, and submitting late theses.