Open Software - UNIX/OSF/UI,Etc.

Thomas W. Doeppner Jr.

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-92-60
December 1992

Open Software — UNIX/OSF/UI, Etc.†

Thomas W. Doeppner Jr.

Department of Computer Science
Brown University
Providence, RI 02912–1910
USA

We discuss the support of *open software* by the computer industry, particularly the UNIX-oriented computer industry. We look at the modern UNIX operating system and its competing implementations, OSF/1 and SVR4. A major theme in the area of open software is support for distributed computing. We discuss distributed file systems and then cover the more general topic of software architectures for distributed computing, concentrating on OSF's DCE. Finally, we discuss the future of UNIX and its competition.

1. Introduction

Single-vendor computer installations are a thing of the past. It is a rare institution whose needs can be met by a single equipment or software vendor. Heterogeneity is now the norm. *Open software* is software that can run on most vendors' equipment, software that can interact with other software running on other platforms, software that lets one exploit whatever special features a particular vendor's platform might provide.

If the computer industry could agree upon a single, standard operating system to be used on all computers and supported by all vendors, then coping with heterogeneous hardware would be simplified (though not simple!). However, this is not likely to happen. A number of computer vendors have decided to support UNIX on their hardware, but, in spite of the work by numerous POSIX committees to produce standards, differences still exist among the various implementations of UNIX that are in common (and hoped-to-become-common) use today. It is still not trivial to port applications from one vendor's UNIX platform to another.

Fortunately, the industry does appear to be moving in the direction of industry-wide standards for various types of software. Though there are not one, but two industry-standard implementations of UNIX, *OSF/1* from Open Software Foundation (OSF) and *SVR4* from UNIX System Laboratories (USL), the differences between the two are being minimized with upcoming releases. OSF's *Motif* is gaining wide acceptance as a windowing environment. OSF's *Distributed Computing Environment* (*DCE*), though not yet supported as a production-quality product by any vendor, appears to be gaining widespread attention and is likely to be supported by a large number of vendors in the next few years. While OSF and USL continue to produce new technology, UNIX International (UI) appears to be taking on the role of coordinating industry involvement in developing higher-level architectural plans, such as their *Atlas* architecture for distributed computing.

In this paper we examine some of the issues and some of the current work affecting the support of open software. We start with the issues affecting individual computers (i.e. operating systems and file systems), and then discuss those relevant to collections of computers.

2. Individual Computers

UNIX is currently the de facto standard operating system. Though it has been in existence for over twenty years, it has changed with the times and now supports most of the features expected of a

†This work was partially supported by DARPA order 8225, ONR grant N00014–91–J–4052

modern operating system. We first describe what these features are and how they are dealt with in modern UNIX implementations, and then discuss the special case of file systems.

2.1. The Modern UNIX Operating System

Today's UNIX operating system, though inheriting a great deal from the first operating system to be called UNIX, has a number of new features. (We hasten to point out that, for the most part, these are new features for the UNIX world, not new features for the field of operating systems; many of them made their first appearance years, if not decades ago in proprietary and research operating systems.) Some of what one is found in a modern UNIX operating system is summarized in Figure 1.

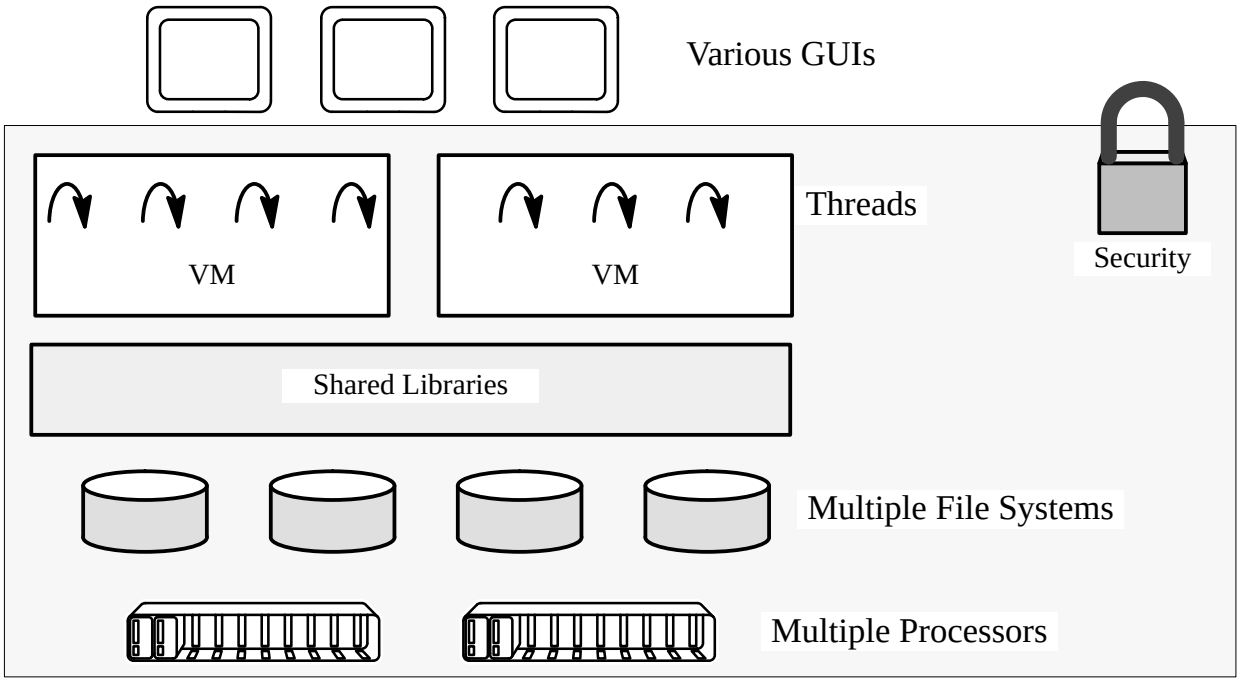

Figure 1. Some Features of a Modern UNIX System

The modern UNIX implementation can support a variety of graphical user interfaces (*GUI*s). Each UNIX process runs in its own private (but sharable) address space: this address space is potentially quite large (a significant fraction of the address space provided by the hardware addressing modes, which is typically four gigabytes on a 32-bit machine, but significantly larger on a 64-bit machine). Within this address space one can map files, and, as a special case, the code for procedures (otherwise known as shared libraries). Any number of threads of control can execute concurrently within each address space. The operating system supports a number of different file systems (where "number" is typically two or three—S5, UFS and NFS), and can run on, and take advantage of, a shared-memory multiprocessor. This means that not only can multiple UNIX processes run simultaneously on different processors, but a single, *multithreaded* process can take advantage of this parallelism. Traditional UNIX security is provided but, as an option, much tighter security can be provided, in adherence to such requirements as the "Orange Book" model of the US Government.

There are currently two "industry-standard" versions of UNIX: OSF/1 and SVR4. (We use the term "industry standard" to mean a standard supported by a large segment of the computer industry. OSF/1 and SVR4 are competing industry standards.) In terms of functionality supported, OSF/1 and SVR4 are roughly equal. At the time of writing, there are no companies shipping production versions of OSF/1, though production versions of SVR4 can be obtained from a number of vendors. Except for such issues as stability and performance of a particular implementation of OSF/1 and SVR4, there are few, if any, significant differences for most users between OSF/1 and SVR4. Assuming the continued development of both OSF/1 and SVR4, both systems will continue to evolve, both will have certain short-term advantages over the other, and both will eventually adopt functionality that is found to be important. Of course, some vendors (e.g. Sun in Solaris 2, which is derived from SVR4) are adding additional functionality to their implementations, that could be very important to certain users.

2.2. File Systems

Two standard file systems have been supported by UNIX systems: *S5* (for "System 5," it was the only local file system supported in UNIX System 5 before SVR4 (system 5 release 4)), the original UNIX file system, and *UFS* (for UNIX File System, otherwise known as the "Berkeley file system" or the "fast file system"), which was developed at the University of California at Berkeley. Of the two, UFS is preferred, being generally always faster than S5, at the expense of a considerably more complicated implementation. The complexity comes from the work done to optimize the layout of files on disk, which is of two forms: time optimization to minimize the time to read a file sequentially, and space optimization to minimize the amount of disk space lost to fragmentation.

A major problem with both UFS and S5 is that lengthy consistency checking is required after a crash. The reason for this consistency checking is not, as is commonly believed, because UNIX buffers all user operations to disk in an internal buffer cache, but because a number of system data structures, describing the file system layout, are maintained on disk. Updates to these data structures require a number of operations. If the system crashes, it is quite likely that the crash occurred in the middle of such a sequence of operations (or in the middle of several such sequences of such operations). When the system comes up, these system data structures (often called *meta-data structures*) are in an inconsistent state, and in order to identify and repair these inconsistencies, it is necessary to examine the entire file system. The amount of time required for this is, at best, proportional to the size of the disk—30 minutes may well be required to check and, if necessary, repair a file system on a gigabyte disk.

The current trend in UNIX file system technology is in the direction of *log-based* or *journalled* file systems. These are file systems in which operations on meta-data structures are treated as transactions in a transaction system—changes are logged. In the event of a crash, the system replays the log and "backs out of" uncommitted operations and completes committed operations. Examples of such systems are *Episode* from Transarc (this product is part of OSF's DCE, where it is called *LFS*, for "local file system"), *JFS* (for "journalled file system") from IBM, and *VxFS* (for "Veritas file system") from Veritas and USL.

With these new file systems, lengthy consistency checking after a crash is not required. Instead, the log is "played back" after a crash. The time required for this is proportional to the size of the log, which, in the case of Episode, contains no more than the last thirty seconds worth of changes to meta-data structures; this process takes far less time than does full consistency checking.

3. Distributed Computing

Computers running UNIX have been used in distributed environments for at least the past decade. The developers of Berkeley UNIX (starting with a version known as *4.1a BSD*) were instrumental in producing software that was widely used and that provided full connectivity among UNIX systems via the Internet. 4.1a BSD begat 4.1c BSD, which begat 4.2 BSD, which begat 4.3 BSD, and so forth. This work was incorporated into both OSF/1 and SVR4. However, the distributed computing paradigm developed at Berkeley was that of a collection of individual computers communicating with one another. Little was done to foster an environment in which it was easy to write distributed applications, especially in heterogeneous environments.

3.1. Distributed File Systems

A necessary component of a distributed environment that received early attention was the distributed file system. Sun's *NFS*, though not the first distributed file system, nor even the first distributed file system for UNIX, was the first distributed file system for UNIX to achieve widespread commercial success.

Though NFS works very well, it is not without limitations. Its first limitation is in scalability. It works well on moderate-size networks, but tends to produce fairly heavy network loads and to overburden servers in large networks. Though this problem is reasonably well dealt with by adjusting one's network topology and improving server technology, NFS still suffers because NFS clients use the network and hence the servers too frequently.

A technology that appears to be solving these problems is the use of *client caching*. Clients of the file system hold onto significant portions of the files they obtain from their servers in a local disk. A few requests are sent to the server to obtain the files needed by a client, the client stores these files (or large pieces of them) on its local disk, and thus most file accesses are dealt with locally and do not involve the network or the server.

This technology was developed in *AFS* (for Andrew File System), a file system initially produced at *Carnegie Mellon University* (the name "Andrew" is used because it was the first name of both Mr. Carnegie and Mr. Mellon). More recent development and commercialization has been done by Transarc.

AFS is currently in widespread use (though there are certainly not as many AFS sites as there are NFS sites). One of its most important applications is for geographically distributed file systems. It has a single, logical file system that covers (at least) three continents—a user sitting in the United States can access files in Europe or Japan as easily as he or she can access files in the local file server. Though this technology was developed as part of AFS, there is no reason why it cannot also be incorporated into NFS. I.e., an NFS client could take advantage of client caching with few, if any changes to the NFS protocol.

Another limitation of NFS is that the semantics of UNIX system calls made to files accessed via NFS are different from those of system calls made to files accessed on the local file system. The differences are subtle—they come up when two different users (on different machines in the NFS case) access the same file. UNIX users expect that when one reads a file, one will see the effect of the most recent write. This is not the case with NFS, in which it may take many seconds for the effect of a *write* issued by one client computer to be noticed on another. This could cause conflicts if the other computer is also modifying the file.

The most recent version of AFS, which is being incorporated into OSF's DCE under the name *DFS* (distributed file system), deals with this semantics issue: a DFS server distributes *tokens* to clients

giving them permission to perform operations on a file. A client reading a file must have a *read token*. If a client desires to write to a file, it requests a *write token* from the server, which grants the request only after it has revoked all extant read tokens. These token-passing operations are not noticed by the user of DFS, whose interface to DFS is just the normal operating-system-supported I/O calls.

3.2. Distributed Computing Architectures

Distributed file systems are an important component of an overall plan for distributed computing. Two such plans are currently being promulgated, one by OSF and the other by UI. A high-level view of these plans is shown in Figure 2.

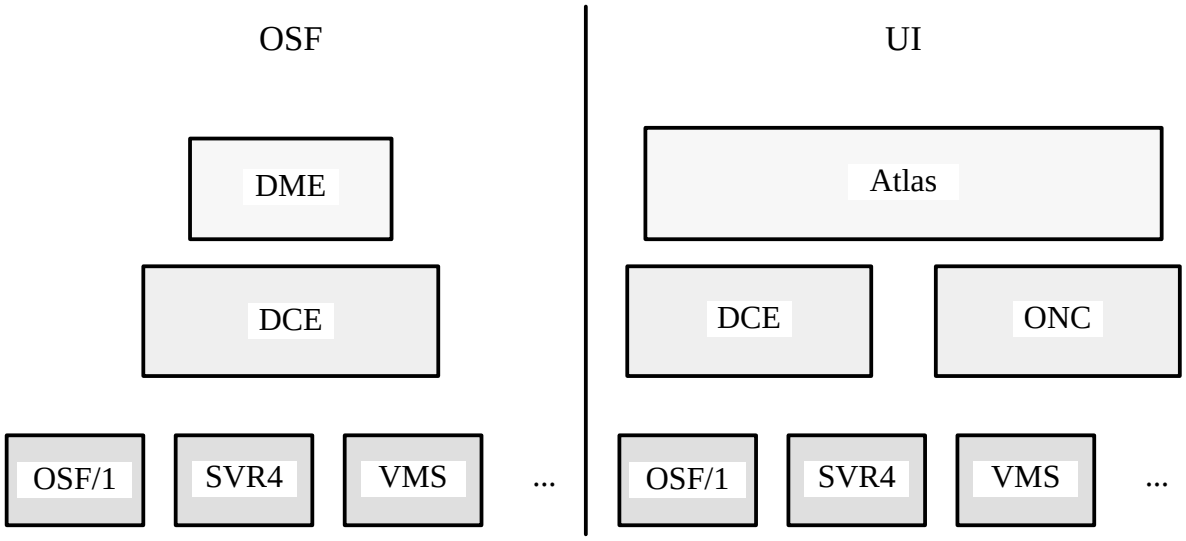

Figure 2. Competing Distributed Architecture Plans

UNIX International (UI), with the aid of (a portion of) the computer industry, has devised an overall framework for an industry-standard distributing computing architecture. The Open Software Foundation (OSF), in the meantime, has been developing an actual distributed computing architecture, known as DCE (for distributed computing environment). Fortunately, as it turns out, OSF's DCE fits within the UI Atlas view of the world. However, Sun's *ONC* (Open Network Computing) also fits within this scheme. The intent of both groups is that, whatever distributed architecture is adopted, it will support (or be supportable by) most existing operating systems, not just UNIX.

DCE is currently well into development. Two initial, "functionality" versions of it have been released, and a production-quality release is expected within a year. As a separate project, OSF is working on *DME* (distributed management environment), whose concern is the management of services within a distributed environment.

We have already discussed two components of DCE, LFS and DFS. We now sketch some of its additional functionality.

3.2.1. DCE Security

As became quite clear to the world after the infamous "Internet Worm" incident in 1988, the distributed system comprised of all the computers on the Internet was not very secure. Fortunately, technology is available to help with these problems.

The general approach is to use encryption techniques to provide security. In DCE, at the user's option, one of three levels may be chosen: *authentication*, meaning that client and server learn, reliably, who the other is, *integrity*, meaning that no third party can modify data transmitted between client and server, and *privacy*, meaning that no third party can learn what information is being transmitted between client and server. Authentication is fairly inexpensive (i.e. non-time-consuming) to provide, integrity is less so, and privacy is fairly expensive.

DCE Security requires that both parties, client and server, possess a "shared secret," an encryption key. How they obtain this shared secret is not dictated by the architecture, but the current implementation provides a technique, based on *Kerberos*, developed at MIT's Project Athena.

DCE Security is integrated into DCE's RPC (remote procedure call) package, so that all communication between clients and servers is secure at the desired level.

3.2.2. DCE Directory Services

In a distributed system providing many services and perhaps multiple instances of many of the services, it is necessary to have a convenient technique for locating these services. DCE provides two such techniques. One, known as *CDS* (Cell Directory Service), is used to find services either in the local (distributed) environment or in some other environment (one running DCE). The other directory service, *GDS* (Global Directory Service), uses X.500 to access data bases that may exist outside of the DCE environment.

4. The Future

For the near term, it appears that the computing community is will profit from the recognition by the computer industry that heterogeneity is extremely important and that efforts such as DCE should be supported and adopted. Though DCE will not be ready for widespread use for at least a year (this sentence is being typed into the computer in early November, 1992), it is of such importance and is so far along that its acceptance seems inevitable.

But what about the future of UNIX? It may be that UNIX has completed its gradual evolution. Upcoming changes could be major. Of much current interest is the notion of *microkernels*. This is the operating-system design concept in which only a minimum of operating-system functionality resides in the kernel (i.e. running in privileged mode); most of the functionality is provided by user-level servers. Such an approach makes development of systems software easier than before—one can use the full set of programming tools designed for user-level applications when working on the operating system! The approach also (potentially) allows the easier extension of the operating system—different servers can be provided to extend the functionality of the traditional operating system. Such servers can take advantage of the low-level interfaces provided by the microkernel, and thus are not limited (and slowed down) by the use of the standard operating-system interface.

There are two well known projects based on this idea. One is the *Mach* operating system of Carnegie Mellon University. Its use as a microkernel for the support of OSF/1 is being investigated by OSF as part their *OSF/1 MK* project. The other project is *Chorus*, being developed by Chorus Systèmes. It has been licensed by USL as a possible basis for a microkernel version of SVR4.

Finally, what about non-UNIX operating systems? Two major new non-UNIX systems are being developed by major companies—*OS/2* by IBM and *NT* by Microsoft. Both provide all the functionality of modern UNIX systems. Both will have tremendous marketing organizations behind them. Both come from the *PC* (personal computer) marketplace, while UNIX, most recently, is a product of the workstation marketplace. But, given the ever-increasing power of PCs, how much longer will there be a difference between the workstation and PC marketplaces? Will the concept of a workstation be of any relevance in the medium-term future? Or, putting it another way, how much longer will the workstation market exist?

In a workstationless market, there may or may not be a UNIX (there certainly will be support for UNIX applications), but the advances made in the area of open software support for distributed computing will still be very important. Your PC, if not your notebook computer, will run DCE.