

# An Implementation of an Extended File System for UNIX

*Clement T. Cole*  
*Perry B. Flinn*  
*Alan B. Atlas*

MASSCOMP†  
Software Engineering

## ABSTRACT

This paper describes the development of an extended file system for MASSCOMP's Real Time Unix (RTU) operating system. It discusses a mechanism by which users can perform file operations upon data physically residing in backing store on a remote computer. All operations are transparent to processes running on the local host. Migration of processes to the remote computer was not considered as a goal and no attempt was made at solving that problem. This system is client/server based and operates between two or more MASSCOMP computers on the same Ethernet. These machines run an enhanced version of the RTU kernel with an enhanced version of the network software. A new reliable datagram protocol (RDP) that supports multiple connections through a single endpoint has been developed to simplify the kernel's interface to the communication mechanism and to improve the throughput of remote file operations. All remote file operations are based on transactions. A global protection domain is used to simplify the concept of file ownership; that is, a single set of *user-ids* and *group-ids* is used on all machines.

## 1. Introduction

Given a network of computers running Real-Time UNIX (RTU), the MASSCOMP variant of the UNIXRit74a Time Sharing System, how can users of each machine on the network easily share databases? These databases might be anything from the system sources for a large programming project, to the telephone directory for an entire company. The key points are that the database is to be shared by many different programs and users throughout the network, and potentially could be *updated* by many different programs at different times. The programs used to update the shared database should be no different than those used to update a non-shared database. The goal of the MASSCOMP EFS project was to produce a new version of the operating system that would allow network-wide file operations without requiring modifications to existing user programs. Thus a user could easily and transparently share data between multiple machines.

---

† MASSCOMP and RTU are Trademarks of Massachusetts Computer Corporation.  
UNIX is a Trademark of AT&T Bell Laboratories.

## 1.1 What is EFS?

EFS is the *Extended File System* facility for RTU that allows users to access data residing on remote backing store. The remote backing store is connected to a normal MASSCOMP machine operating with the EFS environment. Except for a small network time delay, any valid access to the data retained on the remote backing store is *indistinguishable* from an access to data retained within the backing store on the local host. This is referred to as *network transparency*. Note, however, that a new set of error codes was introduced to cope with the new types of errors that arise with the EFS environment.

## 1.2 Constraints

The UNIX system call interface standard proposed and accepted by the /usr/groupBuc84a and the draft standard of the IEEE P1003 POSE working group(IEEd)a have left a seemingly indelible mark on UNIX systems manufactured by different vendors. The standard interface dictates calling conventions and semantics for all major system calls, including file I/O operations. Therefore any attempt to produce an *extended file system* (EFS) for UNIX must lie within the boundaries set by this interface. Furthermore, the authors consider it non-optimal to force users to recompile or relink a working program when the operating system for the same computer hardware is changed from being a simple version of UNIX, to one that supports an EFS. The MASSCOMP EFS works within the constraints described above. All file operations (*open(2)*, *close(2)*, *read(2)*, *write(2)*, *ioctl(2)*, etc.) continue to work as they did previously. Thus any program that works under a pre-EFS version of RTU continues to function correctly with an EFS version of RTU.

## 1.3 Why build an EFS?

In a timeshared system where all users perform their work on the same machine, data can easily be shared among users because it is all stored locally. Indeed, sharing is the norm. Most large systems go to great expense to allow users to share everything from files to in-core program images. In many cases, users may not even know they are working with objects shared by other users.

In the case of smaller workstations, however, data is stored on each of many machines. Since sharing is difficult, its occurrences become rare. In fact, data becomes replicated much more often than it is shared. Disks are copied either physically or via a network, but new versions of data from those disks are rarely sent to all sites that contain copies of it.

Consider the difficulty of keeping several workstations running the same version of system software, or worse yet, keeping many copies of some database up to date. An EFS allows workstations connected by a local area network to share data as though it were stored locally as it is in a large timeshared environment.

For a more concrete example, consider an application such as the design of VLSI circuitry for a central processing unit. Graphical editors, such as KIC-2Bil83a or HAWKKel84a are used by IC designers to layout the circuit geometrically. The circuits are then simulated with tools such as SPICECoh76a, Qua83a and are finally converted to an IC mask. Along the development path, different engineers work on different pieces of the problem with different tools. Each engineer works on his or her piece of the circuit, yet each may periodically need access to other parts, or to the entire circuit as a whole. In order to help the entire design team work together, the master image of the circuit and the "cell libraries" are usually stored on one central machine. Each designer need only concern himself with the files associated with his portion of the project.

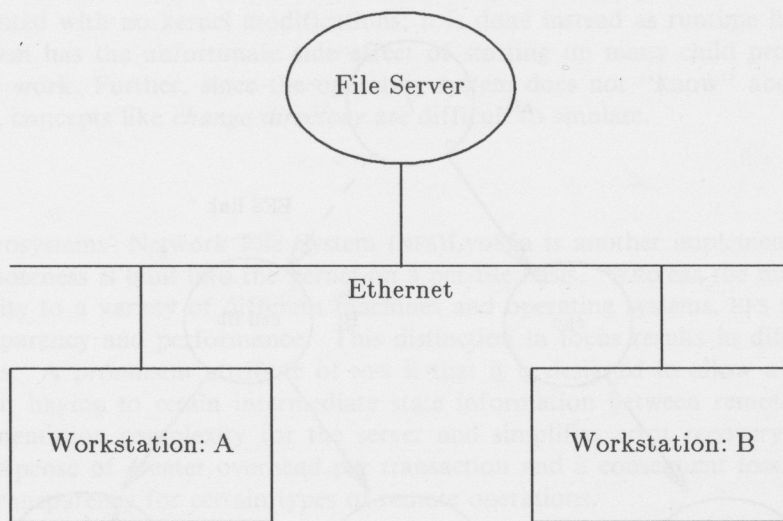


Figure: 1. Two design stations and a shared file server.

In figure 1, we see two different design stations sharing a common file server.<sup>1</sup> By letting each workstation remotely graft some part of the server's file system as part of its own directory hierarchy, any process running on a workstation, (such as the graphics editor) can access data on the remote file server *as though the data were stored locally*. The designer can then share common pieces, such as the cell libraries, without having to copy and store them locally. When the designer needs to use a cell, the editor may simply "read it in" from its master location on the file server.

Figure 2 shows two file systems "virtually linked" together so that each process on workstation A views the remote directory hierarchy as part of its own. Cell libraries are contained on the remote file system, and the local system has a local copy of some part of the total circuit being designed.

1. Note, that you do not have to specify a "file server" as such. Any machine on the network that is participating in EFS can act as either a client or a server.

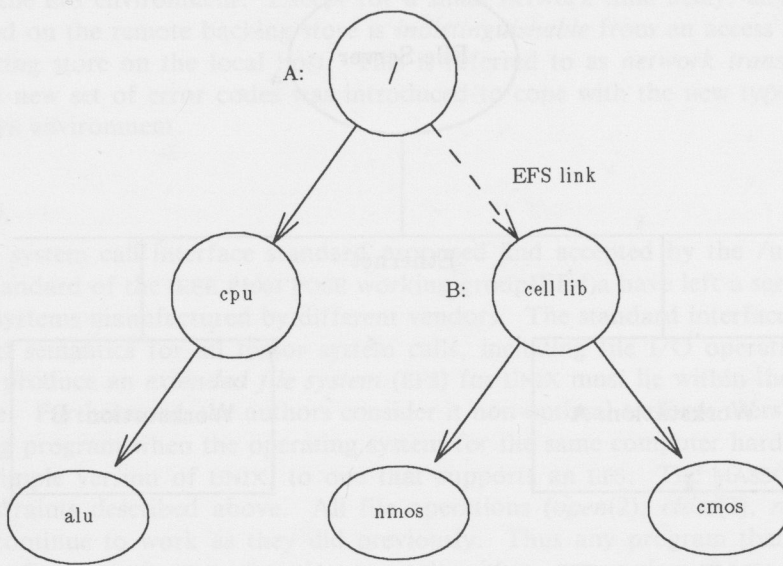


Figure: 2. Machine A is sharing part of Machine B's File System.

## 2. Previous Work

### 2.1 Version 8 File System

The goals of the MASSCOMP EFS project are similar to goals of the Version 8 File System described in the *Proceedings of the Summer 1984 USENIX conference* Wei84a and developed at Bell Laboratories. The Version 8 File System is proprietary to Bell Laboratories, and thus could not be used as a starting point for our efforts. The Version 8 File System, like EFS, is built around the "inode" concept.

### 2.2 Remote Virtual Disk

EFS differs from other network file system implementations in that the kernel has been modified to route the standard UNIX system calls to another host when necessary. It differs from the *remote virtual disk* (RVD) approach, as developed by Mike Greenwald and Larry Allen Gre83a for the MIT Laboratory for Computer Science's multiple VAX/750 UNIX systems running as CPU servers. The MIT approach is similar to the LucasFilm Duf82a method. The major advantage of RVD is that under it, as under EFS, no user code must be modified. Unlike EFS however, RVD does not allow a given object-file or file system to be shared simultaneously by several machines. Rather it allows a single physical disk to be divided up into smaller chunks and parceled out to remote machines.

## 2.3 The Newcastle Connection

The work done by the University of Newcastle-upon-Tyne in EnglandBro82a was another approach that we chose not to follow. The Newcastle system is simple and has the advantage of being implemented with no kernel modifications; it is done instead as runtime library calls. The Newcastle system has the unfortunate side effect of starting up many child processes to do the network server work. Further, since the operating system does not “know” about the “remoteness” of a file, concepts like *change directory* are difficult to emulate.

## 2.4 NFS

Sun Microsystems' Network File System (NFS)Lyo85a is another implementation where the concept of remoteness is built into the kernel on a per-file basis. Whereas the main thrust behind NFS is portability to a variety of different machines and operating systems, EFS is aimed more at complete transparency and performance. This distinction in focus results in differences between the two designs. A prominent attribute of NFS is that it is designed to allow a server system to operate without having to retain intermediate state information between remote requests. This reduces implementation complexity for the server and simplifies error recovery. However, this comes at the expense of greater overhead per transaction and a consequent loss in performance, and a loss of transparency for certain types of remote operations.

## 3. The Design Space

The “design space” for our development effort imposed the following constraints.

- 1.) All kernel code must be written to operate in a multiprocessor environment.
- 2.) EFS development must not impede any other operating system development.
- 3.) The design must operate using a “Client - Server” model.
- 4.) All operations revolve around *inodes*.
- 5.) Calls must be provided to enable and disable remote access.
- 6.) No single client process should be able block an EFS server from serving other client processes.

The next paragraphs describe these constraints and their impact on the design.

### 3.1 Multiprocessor Safety

Unlike other UNIX variants, notably AT&T's System V and the University of California's 4.2 BSD, RTU runs on both dual-processor and full multi-processor (MP) computer architectures. This constraint means that any new code added to the RTU kernel must not hinder the multiprocessor nature of the system.

### 3.2 Impact on other Development

At the time of the EFS development, three large kernel projects were underway, each of which entailed rewriting major sections of the MASSCOMP RTU kernel. The group working on EFS has become adept at folding changes into other versions of the kernel from other development groups. Overall, we have been reasonably successful, but that story is the subject for a different forum.



























