

Fermi National Accelerator Laboratory

FERMILAB-Conf-90/88

Use of UNIX in Large Online Processor Farms*

Joseph R. Biel
Fermi National Accelerator Laboratory
P.O. Box 500
Batavia, Illinois 60510

April 1990

* Presented at the 1990 Conference on Computing in High Energy Physics, Santa Fe, New Mexico, April 9-13, 1990.



Operated by Universities Research Association Inc. under contract with the United States Department of Energy

USE OF UNIX IN LARGE ONLINE PROCESSOR FARMS

Joseph R. Biel
Fermilab, Batavia, Illinois 60510

ABSTRACT

There has been a recent rapid increase in the power of RISC computers running the UNIX operating system. Fermilab has begun to make use of these computers in the next generation of offline computer farms. It is also planning to use such computers in online computer farms. Issues involved in constructing online UNIX farms are discussed.

INTRODUCTION

There is an increasing trend in high energy physics experiments in the direction of using online farms of general purpose computers. "General purpose" as it is used here refers to a computer that can be programmed in a high level language (i.e. FORTRAN) with essentially the same ease as an offline analysis computer. These online farms are used as a final filtering step to reduce the rate at which data is written to tape. The purpose of this paper is to discuss some of the issues involved in running the UNIX operating system in these general purpose computers. I will first discuss some reasons for using general purpose computers in online farms. Next I will give the reasons for running UNIX on such farms. Finally, I will discuss some recent work performed at Fermilab that relates to online UNIX farms.

ONLINE FARMS OF GENERAL PURPOSE COMPUTERS

The main argument for using general purpose computers online is the ease of use they offer in the preparation of the online program. A general purpose computer allows the use of a well known high level language, such as FORTRAN, for writing the online algorithms. This makes it possible for online algorithms to be discussed with the entire experimental collaboration in terms of FORTRAN code instead of some arcane language, such as microcode. Online farms are usually targeted to be used as a final filtering step. In this situation, the filtering program running in each online computer has access to the entire set of data for each event. Because the entire event data is available, the online filtering program operates with essentially the same information that the offline analysis program has. If the online computers are general purpose computers, the program they run can be written in a manner similar to that used to write the offline program. The division between what is

done online verses what is done offline can be made very flexible. An experiment can start its run with little or no event filtering being done online. After filtering algorithms have been fully tested offline, the filtering code can then be moved online. As offline filtering techniques are improved, an improved form of online filtering can be performed. The transfer of code between an offline computer and an online farm is especially easy if the online farms have been built using the same processor chip that is used in the offline analysis computers. This is not a requirement, but it should increase the confidence that any online filtering is executing the same algorithm that has been tested offline.

UNIX ONLINE FARMS

In the above discussion, I have defined a general purpose computer in part as one that can be programmed "with essentially the same ease as an offline analysis computer". A significant contribution to the ease of programming a computer is due to the operating system environment that the computer provides. In order to meet the goal of an online farm of fully general purpose computers, a full operating system should be provided. In particular, the programs should have the usual FORTRAN access to disk files and terminal input/output. This allows program development to be done for an online farm the same way it is done for an offline computer. Initialization data files can be read from ordinary disk files using FORTRAN OPEN and READ statements. Programs can be debugged on the actual farm hardware by running a normal terminal session with a symbolic debugger. Virtual memory paging should be supported so that an occasional need for a large amount of memory does not hit a restrictive physical memory limit. If a program crashes, the operating system can make its usual crash dump file. The crash dump file can then be examined later with a crash dump analysis program. The farm hardware and software should allow peripheral connections to be made over a network. Serial connections can be established with a network utility such as Telnet, and disk connections can be established with network mechanisms such as NFS and ftp. For processors that share the same high speed bus (e.g. VME) the network connection can be made directly over that bus. For processors that do not share such a bus, a network connection can be established over Ethernet.

There are some potential disadvantages to running a full operating system on the online processors. First, the processor modules will be somewhat more expensive because they may need more hardware features to run the operating system. These features range from the simple (i.e. perhaps an onboard time-of-day clock chip) to the complex (e.g. a disk controller and Ethernet controller). Second, more memory will be needed to hold the operating system. Third, the booting of the

farm is likely to be more complex with a full operating system. Fourth, the operating system itself must be "ported" to the processor board. This includes getting the appropriate license for the operating system.

FERMILAB UNIX WORK

Recently the Fermilab Computer Research and Development Department (formerly the ACP group) has done some work that demonstrates how hard it would be to construct an online farm of RISC computers running UNIX. Most of the work reported below has been done by Computer R&D Department members Mark Fischler and Mike Isely. The work was done in the process of developing support software for a VME computer module that was being developed by the department. This module, called the ACP/R3000, is based on the MIPS Computer Systems R3000 RISC microprocessor. The work done falls into four categories. First, UNIX was ported to the ACP/R3000. Second, interprocessor communication mechanisms were implemented. Third, a special version of UNIX, called "diskless UNIX", was prepared that allows construction of farms of mostly "diskless" ACP/R3000 modules. Fourth, some tools were developed to support online use of UNIX on the ACP/R3000.

UNIX Port: The first step was to port UNIX to the module. This was greatly aided by the fact that a version of UNIX for the R3000 microprocessor had already been developed by MIPS. A source license for that version of UNIX was purchased and the necessary changes made to it. The changes involved were primarily due to the different VME access mechanism used by the ACP/R3000. Changes were made to approximately 40 of the approximately 1000 source files that make up UNIX. This took about six months time for two (very good) people to complete. This included the time for them to learn about UNIX, both from reading books about the subject and from studying the UNIX source files.

Interprocessor Communication: The next step was to implement an interprocessor communication mechanism. This was needed to provide network connections with the processor and as a preliminary step toward the implementation of "diskless" UNIX described below. Interprocessor communication (Figure 1) was implemented first by constructing a device driver that could transfer a block of data to or from a range of VME addresses. Because the memory of each ACP/R3000 is accessible over VME, this device driver allows one processor to read or write the memory of another processor. This device driver was then used to allow "IP" packets (i.e. the lower level part of the TCP/IP protocol) to be transferred between processors. The result was an implementation of standard network mechanisms over the VME bus between ACP/R3000

modules. Thus, for example, NFS disk references could then be made between processors over VME.

Diskless UNIX: The next step was to allow farms of mostly "diskless" processors to be constructed. The idea here was to allow an entire crate of ACP/R3000 modules to boot UNIX from a single disk. Ordinarily, each VME processor module running UNIX would boot from

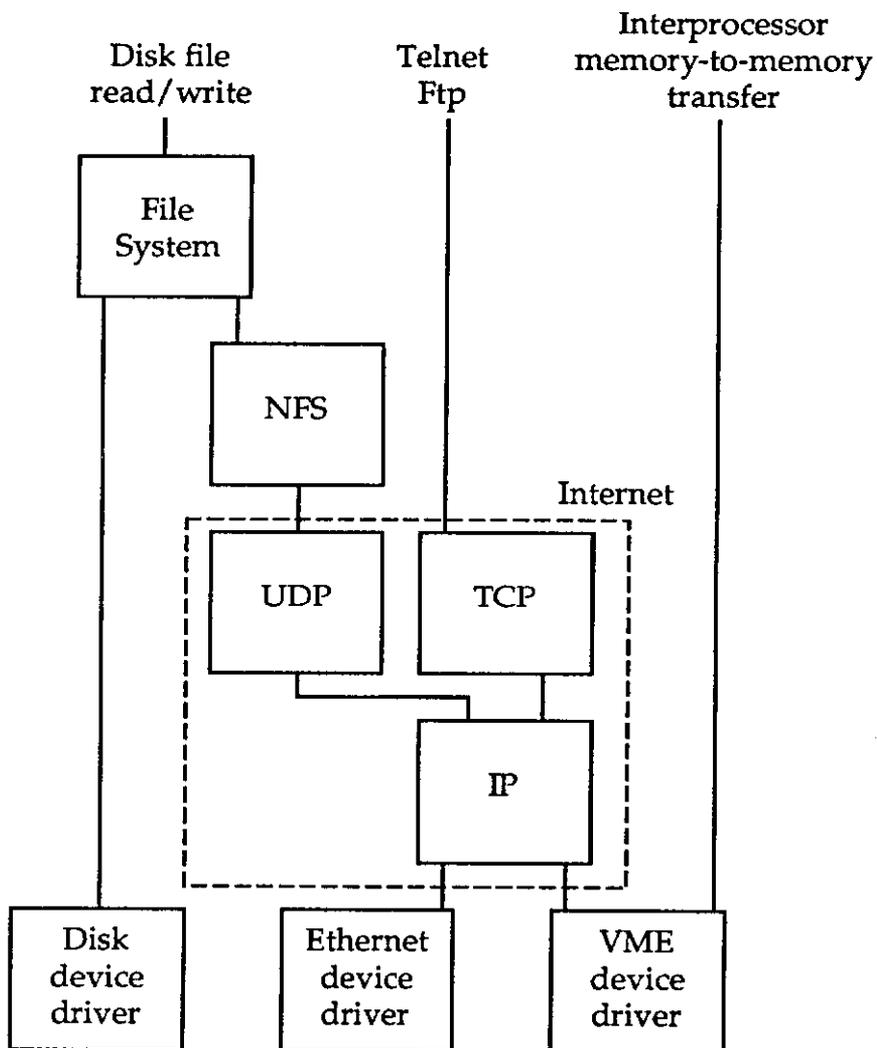


Figure 1. Interprocessor communication paths

its own disk. This would require as many disks as there are processor modules. This is not only expensive but also mostly unnecessary. In a typical use of an online farm, each processor will, in the steady state, run a memory resident program with no need to use a disk file for virtual

memory paging or reading disk files. Access to a disk is needed only for booting UNIX and starting up the online application program. This means that the disk I/O bandwidth needs of each processor are small. It is, therefore, acceptable to have a single disk shared between many processors. At Fermilab, we have achieved this by producing a "diskless" version of UNIX.

This diskless version of UNIX works in the following way. One processor in each VME crate boots UNIX by using a VME disk controller module to read UNIX from a disk. This processor, called the "boot server" (Figure 2) also connects with a VME Ethernet controller module when it boots so it has access to all other computers on Ethernet. The boot server then writes a copy of the UNIX kernel into the memory of each of the other processors in the VME crate (which are called "diskless processors"). The boot server then starts each of the diskless nodes running. Each diskless node then uses NFS file reads over VME (which are serviced by the boot server) to complete its UNIX boot. Once a diskless node has complete booting, it is in full network communication to the boot server (over VME) and to any processors on Ethernet (over VME to the boot server which acts as a network gateway). Thus, for example, anyone logged onto a computer attached to Ethernet may Telnet to any of the nodes in the crate and logon. The crate is essentially a network of UNIX processors with VME taking the place of the more common Ethernet network connection. The diskless processors also use the boot server for any virtual memory paging that they require. Of course, there is a potential bottleneck if too many network requests are made by the diskless nodes to the boot server. As long as the steady state online process running in each processor is memory resident and does not read disk files, this is not an issue. If an abnormal condition is detected by an online process, the full power of UNIX (virtual memory paging, disk file references, symbolic debuggers, crash dumps, etc.) are available. If one processor has a problem, it can be probed by logging on to it over the network and this will have minimal impact on the other running processors.

Online Tools: Finally, some tools were constructed to support online use of UNIX on the ACP/R3000. Two important examples of these tools are a set of UNIX system calls for physical memory mapping and the implementation of a fast interrupt service mechanism.

Memory Mapping: First, a set of UNIX system calls was written that provides a way for an ordinary UNIX process to map a range of its virtual memory space to a fixed range of physical addresses. This mapping will not be changed by UNIX if virtual memory paging or process swapping occurs. A process can establish such a mapping and then communicate its physical address to an event builder. The event builder can then pass a stream of events to the process by placing them in the buffer. Because the ACP/R3000 has a window to the VME bus that appears as a range of

physical addresses, a process can also use the system calls to establish a memory mapped window to the VME bus.

Fast Interrupt Service: An issue of great concern to online applications of a processor is the interrupt service time. The version of UNIX for the ACP/R3000 allows implementation of a fast, simple interrupt service routine. This was done by adding a new system call to the the UNIX kernel. By using this call, an interrupt service routine can be

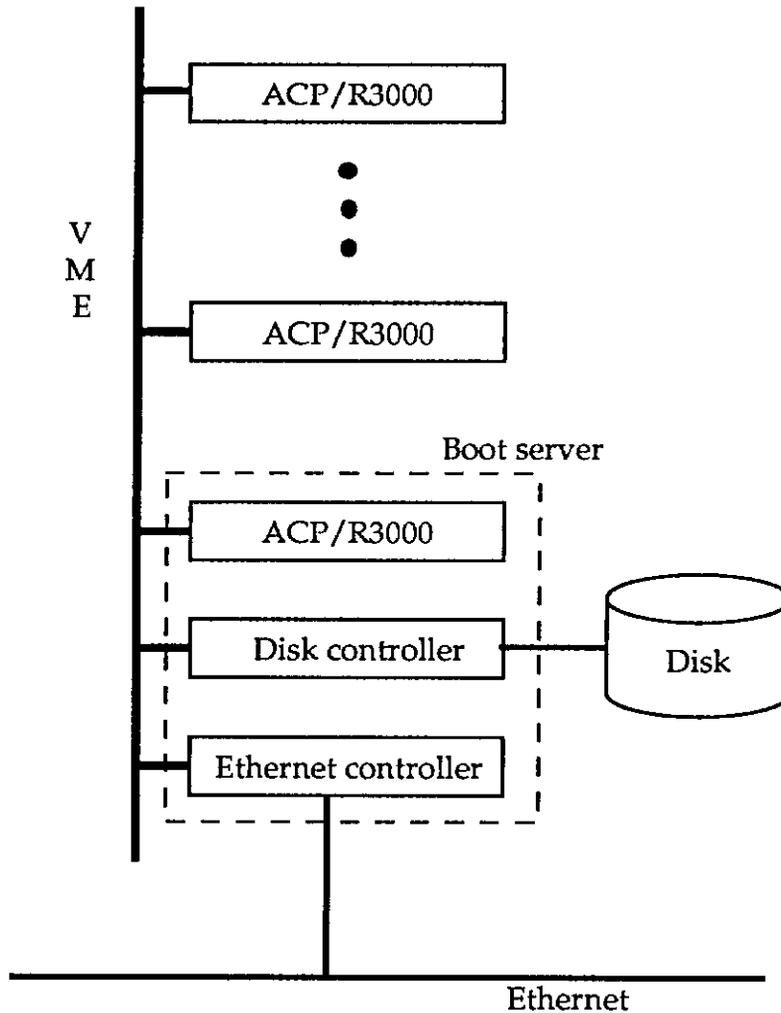


Figure 2. Mostly "diskless" UNIX farm

attached to an interrupt. Using this mechanism, the minimum time to service an interrupt is approximately 5 microseconds.

CONCLUSIONS

A fully supported general purpose computer has a great many potential advantages for use in an online processor farm. The combination of UNIX with a high performance RISC processor makes an attractive candidate for building online farms. The work that the Fermilab Computer Research and Development Department has done in porting UNIX to its ACP/R3000 VME processor module has explored many of issues involved in constructing a successful online farm.