Writings from the Past

Machine-readable versions of early Unix material are hard to come by, even for us. "Backup" in those days (1969 through the early 70s) consisted of punched cards, paper tapes, or uploading to a Honeywell machine. We no longer have those cards, tapes, or the Honeywell.

When we got a PDP-11 around 1971, we did get DECtape, and did save some material, though not enough. A few years ago Keith Bostic and Paul Vixie resurrected a PDP-11 DECtape drive and offered to read any old tapes we might have around, and I sent several to them. These notes are among the small treasures discovered there.

Two files named "notes1" and "notes2" were on the tape labelled "DMR", and their date, if I correctly interpret the time representation on the tape, is 15 March, 1972.

I reproduce them below. I have no memory of why I wrote them, but they look very much like something to have in front of me for a talk somewhere, because of the references to slides. From the wording at the end ("the public, i.e. other Labs users"), I gather that it intended to be internal to Bell Labs. HTML markup and the corrections and annotations in [] were added in September 1997, but otherwise it's original.

Dennis

---

notes1

UNIX is a general-purpose timesharing operating system for the Digital Equipment Corp PDP-11.

It is the only such system known to me; in any case it is probably the first. DEC, in particular, shows no signs of producing a multi-user system.

Some history and credits.

UNIX was written by K. Thompson. I wrote much of the system software; Ken most of the rest; Other contributors have been Joe Ossanna, Doug McIlroy, and Bob Morris.

The first UNIX was on a PDP-7. Most features were present, some in rudimentary form.

UNIX-11 system was largely written in Jan.-Mar., 1971; since the[n] changes are generally refinements.

The charter for the project, and the reason the machine was obtained, was to develop a document editing and formatting system. The original notion was to use UNIX as a development tool only, and have the editing system run stand alone. [This is true as it applies to justifying the purchase of the PDP-11. The earlier work on the PDP-7 just happened in the course of doing research.]

It turned out, however, that it was quite practical to have the editing system run under UNIX, and this is [how it] operates.

The editing system is now being used by the Patent Division at Murray Hill to prepare patent applications. I undertand that the bulk of the applications are being done by UNIX. UNIX is also being tried out by two typists in the MH typing pool.

At the start I stated that UNIX was a general-purpose time-sharing system. I imagine the concept is familiar, but I want to bring out a few points.

First, it is general purpose. This means it may, or may not, be suitable as a basis for various special-purpose applications. The two things that come to mind are management of large data bases and applications requiring very rapid real time response.

For several reasons, not all defensible, very large files of information are not very well handled by the file system proper. I will return to this point.

In the real-time area, the system has no direct hooks to allow a user-written program to respond very rapidly to an external event. "very rapidly", here, means in less than about half a second, which is the approximate time to swap programs off the disk. There are thoughts about how to arrange this, but it has not been done.

Notice, however, that the system does in fact make rapid responses to events, in the sense that it is able to pick up characters from typewriter terminals even when they come only a few milliseconds apart. Thus if the "real-time" requirement is really that if collecting characters for a terminal or other machine, that ability is already there; what may be lacking is the possibility of scheduling a non-system program which wants to act on those characters within a very short time.

So far I have been talking about the disadvantages of the general-purpose system. The advantage lies in the ease with which program development can be done. With any reasonable number of users, responses to user's commands is quite fast, and the cycle of edit-translate-execute-debug is speed-limited by thinking time, not compute time. For example, it takes about 50 seconds to assemble and install a new UNIX system.

The time-sharing aspect of the system is also vital to a program development effort, since it means that several people can be using the machine at once.

Likewise when the system is adapted to deal with a particular application, the fact that multi-programming is built in from the start means that the most subtle problems which come up when several things must go on at approximately the same time have already been solved.

Given that UNIX is excellent for program development, but may have limitations as a base upon which to build a more specialized system, the question arises as to the advisibility of writing one's own system using UNIX. I can only say that of the several projects using UNIX, all have in fact decided to use it not only for development but to support their application directly. Often some modifications to the system have proved necessary, however. Typically these relate, though, to support of some device the standard UNIX does not provide for, and to rearranging the management of core memory.

HARDWARE

UNIX is running on at least five PDP-11, no two with the same complement of hardware. The slide shows the minimal complement possible.

The basic requirements above a PDP11 processor are

- 12 K core
- some kind of disk
- a clock
- an EAE (extended arithmetic element, for multiply/divide
- Some sort of tape, to provide for loading the system software and saving the disk

Above the minimum, it is of course desirable to have more core, lots of disk, communications interfaces, paper tape reader, and a ROM containing a bootload program.

A summary of the devices which have been attached to actual UNIX systems includes:

- DC-11 communications interfaces attached by DATAPHONE to 10, 15, or 30 cps ASCII terminals (not DM-11 as yet)
- RF fixed head disk (256K words)
- RK moving head disk (1.2M words)
- RP moving head disk (2314-style, 10 M words)
- paper tape reader/punch
- 201 DATAPHONE interface
- ACU on DATAPHONE
- DECtape
- Magtape (9-track)
- card reader
- line printer

SOFTWARE

There is a good deal of software that goes along with UNIX. It should be pointed out that it was all written using UNIX; none of it comes from DEC or elsewhere.

It should also be pointed out that almost all of it is being worked on in one way or another. That is while all I will list is usable, there are a number of things which we regard as desirable that are not complete.

The major pieces of UNIX software are:

- assembler
- link editor
- text editor
- FORTRAN compiler
- B compiler
- symbolic debugger
- text formatting program
- M6 macro processor
- TMGL compiler-compiler, the last two contributed by Doug McIlroy
- command line interpreter
- many utilities, mostly to deal in one way or another with the file system

SYSTEM

The system proper can be regarded as falling into three parts: the file system, the process control system, and the rest.

I will only talk about the file system in any detail at all.

Files in UNIX are arranged in a hierarchical, tree shaped structure. There are two types of object: files, and directories. A directory is actually no more than a file, but its contents are controlled by the system, and the contents are names of other files. (A directory is sometimes called a catalog in other systems.)

Thus we have the type of arrangement shown in the slide: there is a root directory, which is the "base" of the tree (as usual, the tree is upside-down) which contains files and directories.

Each of the directories under the root also can contain files and directories, and so on.

In UNIX, files are named by giving a sequence of directories, separated by slashes, and ending in a file or directory (for example: ...)

The name of the root is "/", and so it begins the sequence.

(example...)

Every user always has a current directory, which belongs to him. Files can also be named with respect to the current directory, when the name does not begin with a "/". (examples: ...)

It is possible for the same file to appear in several different directories, under possibly different names. This feature is called "linking" (example).

It is also possible for the directory hierarchy to be split across several devices. Thus the system can store a directory, and all [files] and directories lower than it in the hierarchy, on a device other than the one on which the root is stored.

In particular, in our own version of the system, there is a directory "/usr" which contains all user's directories, and which is stored on a relatively large, but slow moving head disk, while the othe files are on the fast but small fixed-head disk.

---

notes2

One of the most interesting notions in the file system is the special file.

Certain files do not refer to disk files at all, but to I/O devices. By conve[n]tion, such special files reside in a particular directory, although this is not necessary. When a special file is read or written, the device it refers to is activated. For example, all the communications interfaces attached to typewriters have special files associated with them. Thus, provided you have permission, anyone can send a message to another user simply by writing information onto his typewriter's special file. There are special files, for example, to refer to the paper tape reader and punch, to the 201 dataphone, the console typewriter, and whatever other devices may be on the system. An effort is made to make these special files behave exactly the same way that ordinary disk files behave. This means that programs generally do not need to know whether they are reading or writing on some device or on a disk file.

The system calls to to I/O are designed to be very simple to use as well as efficient. There is no notion corresponding to GEFRC on the Honeywell machines and "access methods" in OS [/360 from IBM] because the direct use of system entries is so straightforward.

Files are uniformly regarded as consisting of a stream of bytes; the system makes no assumptions as to their contents. Thus the structure of files is controlled solely by the programs which read and write them. A file of ASCII text, for example, consists simply of a stream of characters delimited by the new-line characters. The notion of physical record is fairly well submerged.

For example, the system entry to read a file has only three arguments: the file which is being read; the location where the information is to be placed; and the number of bytes desired. Likewise the write call need only specify the file under consideration, th[e] location of the information, and the number of characters to write. The system takes care of splitting the read or written information into physical blocks as required.

The I/O calls are also apparently synchronous; that is, for example, when something is written, so far as the user is concerned, the writing has already been done. Actually the system itself contains buffers which contain the information, so that the physical writing may actually be delayed.

There is not distinction between "random" and sequential I/O. The read and write calls are sequential in that, for example, if you read 100 bytes from a file, the next read call will return bytes starting just after the last one read. It is however possible to move the read pointer around (by means of a "seek" call) so as to read the file in any order.

I should say that that it is not always desirable to ignore the fact of physical record sizes. Program which reads one character at a time from a file is clearly at a disadvantage compared to one which reads many, if only because of system overhead. Thus I/O bound programs are well-advised to read and write in multiples of the physical record size (which happens to be uniformly 512 bytes). But it is efficiency, not a logical requirement, which dictates this.

PROBLEMS

I mentioned earlier that UNIX was not especially suited to applications involving vast quantities of data. The reason is this: files are limited in size to 64K bytes. The reason for this is not particularly defensible, but it has to do with the fact that the PDP-11 word size is 16 bits.

There are a couple of ways around this problem. One of them is simply to split one large logical file into several smaller actual files. This approach works for a while. The limitation here comes from the fact that directories are searched in a linear fashion. Thus if the are a vast number of files, it can become quite time-consuming tosearch directories to find the files they contain. We have not noticed this to be a problem, so far, it is only a worry.

Another way around the small file size is to use a disk as a special file. For various reasons, when an entire disk drive is accessed as a special file, the size limitation does not occur. Thus one can set up a program which manages its own data-- in effect is its own, special-purpose file system-- and expect reasonable results.

This again bears on the general versus special purpose system: it probably is more efficient anyway to do your own data management, provided the extra labor is worth the cost.

PROCESS CONTROL

As I said, the second part of UNIX is that part concerned with process control. A process in UNIX is simply the execution of a program. Each user has at least one process working on his behalf: its task is to read his typewriter and interpret what he types as commands to the system to do something. The program associated with this process is called the Shell, and it has many valuable features, including the redirection of I/O, so that you can execute programs which ordinarily write, for example, on the typewriter, and arrange that their output go on a file.

I will not go into any details, except to say that either by use of the Shell, or from within a program, it is possible to create an asynchronously running process executing any program designated.

SUMMARY

If you are interested in using UNIX, there are a number of points about which you should be aware.

First, having to do with the PDP-11 hardware:

the PDP-11, although probably more powerful that most people realize, is not a large machine: a PDP-11 can only accommodate 28K 16-bit words of core.

Moreover, the 11-20 has no hardware protection features: any user can at any time crash the system by executing a program with any of an infinite variety of bugs. This fact is probably most important during program development.

The PDP11/45 essentially solves both of these problems, in a very cost-effective way-- it is hardly more expensive than an 11/20 when the total system cost is considered. It has hardware segmentation and 256K of core can be attached. Since we will be one of the first to get and 11/45, there will definitely be a UNIX on it very soon after it arrives. (however the date is still uncertain.)

Perhaps more important is the fact that UNIX is essentially a two-man operation at present. Anyone who contemplates a UNIX installation should have available some fairly sophisticated programming talent if any modifications planned, as they almost certainly will be. The amount of time that we can spend working on behalf of, or even advising, new UNIX users is limited. Documentation exists, but never seems to be complete.

There have been rumblings from certain departments about taking over the maintenance of UNIX for the public (i.e., other Labs users) but I cannot promise anything.