Introduction

Over the past few years, the Unix operating system has come into wide use, and suffered a lot of rewriting and tinkering since the first publication describing it in 1974. Actually, Unix was born in 1969 not 1974 with instructive behind story.

Origins

During the period 1968-1969, computer science at Bell Laboratories was unsettled because of inevitable withdrawal of the Labs from the Multics project. Multics failed to deliver sort of usable system promptly. Moreover, the organizational was separated by computing services and computing research.

The group that related to the beginnings of Unix felt need to succeed. Moreover, the convenient interactive computing service was only available to under Multics. They wanted a good environment to do programming and a system around which a fellowship could form provided by Multics. Thus, during 1969, they tried to find an alternative to Multics. They lobbied intensively to purchase a medium-scale machine. However, the proposal was rejected because they were asking the Labs to spend too much money on them with vague plan.

At that time, Thompson, R. H. Canaday, and Ritchie developed the basic design of a file system that was become the hub of Unix. Thompson also wrote a detailed simulation of the file system performance and paging behavior of programs. In addition, he studied a new operating system for the GE-645, writing an assembler for the machine and a fundamental operating system kernel.

Moreover, during 1969, Thompson developed the game transliterated into Fortran for GECOS. Thompson had to find a little-used PDP-7 computer with an excellent display processor. They had to ignore all existing software, to write in assembly language and to produce paper tapes to be carried to the PDP-7. As a result, this game introduced the early technology of preparing programs for the PDP-7. Thompson also developed implementing the paper file system. He solved it with the other requirements for a working operating system especially notion of processes. After assembler was completed, the system did not need to depend on the programs using GECOS and files transferred to the PDP-7 on paper tape. Finally, in 1970, that Brian Kernighan suggested the name ‘Unix,’ the operating system we know today was born.

The PDP-7 Unix file system

The file system of PDP-7 Unix was nearly identical to today’s one. It had an i-list, directories, special files describing devices. The important file system calls (Read, write, open, create) were present from the start. A difference was that the unit of I/O was the word, not the byte. This meant all programs dealing with character streams ignored null characters. Another difference was the lack of erase and kill processing for terminals. In spite of its similarity to the current file system, the PDP-7 file system were no path names, and each file-name relative to the current directory. Links were important because the lack of path names. Also, there was a directory called dd that contained entries for the directory, so each user did not need to maintain a link to all directories. To use subdirectories sufficiently was hard and there was no way to create a directory while the system was running.

File system was difficult to change its configuration because the code for devices was spread widely throughout the system. Also, the machine had only a single fixed-head disk so there was no notion of a removable disk pack. The operating system code was a simplified version of the present scheme. The system was not multi-programmed, only one program was in memory at a time, and control was passed between processes only when an explicit swap took place.

The current buffering mechanism was no overlap of disk I/O with computation. The DMA controller was unable to access memory during an instruction, because the upshot could incur overrun errors. So, the interrupt routines for the clock and terminals had to be coded in strange way to avoid indirection.

Process control

Process control means the mechanisms by which processes are created and used. (For example, fork, exec, wait, and exit implement). Today's commands which is different with PDP-7 Unix are executed by the shell. However, Processes in PDP-7 Unix with each of the two terminals attached to the machine. There was no fork, wait, or exec. When shell closed all its open files, it opened the terminal special file to read a command line from the terminal and linked to the file in command. It could not support background processes, but IO redirection was supported when replaced its standard input or output with the appropriate file. Implementation of the shell as a user-level program was subsequent development. The structure of this process control scheme is similar to that of many interactive systems that require special mechanisms to implement useful facilities. Unix at that stage didn’t supply the special mechanisms and showed some problems that a shell had to close all its open files to get rid of any open files. Moreover, the shell could not retain memory across commands, because it was executed after each command. Thus, a further file system convention was required.

Process control in modern form was represented small, easily-coded changes to what existed such as separation of the fork and exec functions. The most common model of new processes involves a program for the process to execute. In Unix, a forked process continues to run the same program as its parent until it performs an explicit exec. The separation of the functions exists in Unix mainly because it was easy to see which fork implemented without changing.

The initial implementation of fork required only expansion of the process table and addition of a fork call. Operating system and user programs needed change, but a combined fork-exec was more complicated because its function was already performed, using explicit IO by the shell. A different protocol was used between the initialization program and the shells for each terminal. Anyway, the new process control scheme provided some very important features such as detached processes and recursive use of the shell. Systems had to supply batch facility and a command interpreter.

Although the multiple-process idea was effective, it was discovered that the chdir command had stopped working. Finally, they found that in the old system chdir was an ordinary command and it adjusted the current directory of the process attached to the terminal. They had to make chdir as a special command, executed internally within the shell. Also, there was another mismatch between the system and the new process control scheme. The problem with this organization was evident when we tried to use command files. To solve this problem, new system table was required to contain the IO pointers of open files independently of the already opened process.

IO Redirection

The IO redirection, using the ‘>’ and ‘<’ characters, was inspired by an idea from Multics. Multics has a general IO redirection mechanism embodying that can be redirected to various devices, files, and stream-processing modules. Even in the version of Multics, there was a command that switched subsequent output and another command to reattach output to the terminal.

However, its sequence that was used often during the Multics time did not occur to them at the time because the keepers of the shell may not even have known of the usefulness. And finally, Thompson found he could use both the Unix IO system and its shell.

The advent of the PDP-11

By the beginning of 1970, PDP-7 Unix was still capable of providing a more proper programming environment than its alternatives. However, the PDP-7 was old and the next one failed to get interest. So they proposed acquisition of a PDP-11. The amount of money was less expensive and it used a system designed for editing and formatting text. They ordered for a PDP-11 in May and the processor arrived at the end of the summer. Moreover, core-only version of Unix was written using a cross-assembler on the PDP-7.

The first PDP-11 system

Once the disk arrived, the system was quickly completed but there was a transliteration problem. However, there were important changes in the interface to the user. It had the present directory structure like full path names, the modern form of exec and wait, and conveniences. Also, it had a small size. Usefulness of PDP-7 Unix was proved as a development tool to create PDP-11 Unix. So, they transliterated the roff text formatter into PDP-11 assembler language. Finally, they prepared to submit patent applications. The main advantages they offered were Teletype’s model 37 terminals and roff with the ability to produce line-numbered pages. Unix had a reputation for supplying interesting services on modest hardware and it got a high score in the benefit/equipment. The experiment was trying but successful. Patent department adopt Unix and they achieved sufficient credibility to acquire one of the first PDP 11/45 systems made.

Pipes

Pipe which was used in a pipeline of commands is one of the most widely admired contributions of Unix to the culture of operating systems and command languages. The ‘communication files’ of the Dartmouth Time-Sharing System did similarly what Unix pipes do. M. D. McIlroy suggested that commands should be thought of as binary operators and left/right operand specified the input/output files. Moreover, to make a pipeline, command operators could be stacked up. There were some oppositions to the idea. However, with McIlroy’s persistence, pipes were finally installed in the operating system and a new notation was introduced. It used the same characters as for I/O

redirection. The new facility was enthusiastically received, and the term ‘filter’ was made. Many commands were changed to make them usable in pipelines. However, there were some problems with the notation like lexical problem. For example, the string after ‘>’ was delimited by blanks, so they need a parameter. Also, the notation was not unique. As a result, the pipe notation using ‘<’ and ‘>’ was replaced by the present one that uses a unique operator to separate components of a pipeline. Of course, it too has limitations when there are situations in which multiple redirected inputs and outputs are called for.

Because Multics provided a mechanism which IO streams could be directed to the device or file serving, the stream-splicing in Multics might be the direct precursor of Unix pipes. However, The Unix pipeline is constructed from the same commands used constantly in simplex fashion.

High-level languages

Every program for the original PDP-7 Unix system was written in assembly language. Moreover, there was no loader or link-editor, so every program had to be complete in itself. The first language to appear was a version of McClure’s TMG that was implemented by McIlroy. Soon after, Thompson produced a compiler for the new language B. The compiler produced simple interpretive code. There were the benefits of using a reasonable language to write what are usually called ‘systems programs’ like compilers, assemblers. When the PDP-11 arrived, B was moved to it almost immediately. However, B did not take over instantly because of the slowness of the interpretive code. Moreover, there were mismatch of the word-oriented B language with the byte-addressed PDP-11. Thus, in 1971, work to create C language began. The most important point occurred when the operating system kernel was rewritten in C. The most extensive change was the introduction of multi-programming because the internal structure of the system became much more rational. The success of this effort convinced that C was useful as a nearly universal tool for systems programming. The success of Unix was responsible for the readability, modifiability, and portability of its software in high-level languages.

Conclusion

The programming environment provided by the early versions of Unix seems extremely harsh and primitive. Nevertheless, the memory fixes on what was good and what lasted. They hope we can reminisce with the same mixed impression of progress.