1-1-2010

First edition Unix: Its creation and restoration

Warren Toomey
Bond University, Warren_Toomey@bond.edu.au

First Edition Unix: Its Creation and Restoration

Warren Toomey
Bond University

Until recently, the earliest versions of the Unix operating system were believed to have been lost completely. In 2008, however, a restoration team from the Unix Heritage Society completed an effort to resurrect and restore the first edition Unix to a running and usable state from a newly discovered listing of the system's assembly source code.

The Unix operating system is arguably one of the most influential OSs in computing. Designed and implemented by a small group of researchers at AT&T's Bell Laboratories, Unix has since been modified and extended by thousands of individuals and companies. Commercialized in the 1980s by AT&T as the product System V, Unix spawned a multitude of derivative systems[1] including Solaris from Sun Microsystems, AIX from IBM, and most recently OS X from Apple. Although Unix itself was a proprietary system, its ease of modification combined with a user culture of self-help enabled programmers from the University of California, Berkeley, to produce the BSD derivative.[2] By replacing the proprietary Unix code in the 1980s and 1990s, BSD took Unix and its culture into the realm of open source.[3] Unix's design and implementation has also heavily influenced the creation of Linux, the most prolific open source system to date.[4]

Until recently, the earliest versions of the Unix operating system were believed to have been lost completely. In 2008, a restoration team from the Unix Heritage Society (TUHS) completed an effort to resurrect and restore the first edition Unix to a running and usable state from a newly discovered listing of the system's assembly source code. This article describes the evolution and features of that first edition as well as the technical and philosophical issues the project faced while restoring the software.

The creation of Unix

The catalyst for the creation of Unix was the withdrawal of AT&T's Bell Laboratories from the Multics project in 1969.[5] Multics was started in 1964 as a collaboration between MIT, General Electric, and AT&T's Bell Laboratories to build on the new concept of time sharing, pioneered by such systems as the Compatible Time-Sharing System (CTSS),[6] and to create a large centralized computing utility.[7] Multics also advanced many other architectural features such as virtual memory, memory-mapped file access, multiprocessor support, and an implementation in a high-level language (PL/I).[8] For various reasons, by the end of 1968, the project was under financial pressure and came close to cancellation. Although AT&T had originally joined the project to develop a system for their in-house computing needs,[9] the company eventually withdrew from the troubled venture in April 1969.

Bell Labs had invested heavily in Multics with the purchase of a GE 645 machine in 1965 and with several researchers dedicated to the project including Ken Thompson, Dennis Ritchie, Joseph Ossanna, Stu Feldman, Doug McIlroy, and Bob Morris.[7] The cancellation of Bell Labs' involvement had two effects: management was reluctant to fund further OS research, and the researchers were left without an interactive time-sharing system. As one of the Bell Labs researchers noted, ''The toy had gone . . . There was a clear lack of momentum.''[10]

The loss of Multics, however, did not stop all OS research at Bell Labs. Thompson in particular was still keen to create a useful development environment, and after the cancellation, he worked with Ritchie and Rudd Canaday on the file-system research that Thompson had started on Multics.[11] To continue their research, Thompson, Ritchie,

 Published by the IEEE Computer Society 1058-6180/10/$26.00

and Ossanna lobbied management for the purchase of a DEC PDP-10 and, later, an SDS Sigma 7, but these proposals were turned down.[5] In the summer of 1969, the researchers turned to a cast-off PDP-7 to implement their ideas. Thompson "allocated a week each to the operating system, the shell, the editor, and the assembler, to reproduce itself."[11]

The new system borrowed several ideas from the researchers' experience with Multics, including

- a tree-structured file system;
- a separate, identifiable program to do command interpretation—the name for the program, the *shell*, was borrowed from Multics;
- no predefined file structure, except that of an array of bytes;
- text files consisting of sequences of characters separated by new lines;
- the semantics of I/O operations (read and write) as referring to a file handle, a buffer, and a count, thus hiding the underlying disk blocks.[7]

But as Ritchie noted,

> We were a bit oppressed by the big system mentality. Ken wanted to do something simple . . . so Unix wasn't quite a reaction against Multics, it was more a combination of these things. Multics colored the Unix approach, but didn't dominate it one way or the other, towards an anti-Multics system, or a copy on the cheap.[7]

Multics was not the only system to influence the new OS research at Bell Labs. As an undergraduate at the University of California, Berkeley, Thompson had been exposed to the Berkeley Timesharing System,[12] from which the team borrowed the concept of per-process execution.

The PDP-7 was useful in the short term, but it was on loan, "was already obsolete, and its successors in the same line offered little of interest."[5] In 1970, the team submitted a new proposal for the purchase a PDP-11, ostensibly for in-house text-processing use and not for OS research; the proposal was also "an order of magnitude less than what we had previously asked."[5] Management accepted the proposal, and a PDP-11/20 (the first model in the PDP-11 family) arrived in mid-1970, although the machine's disks did not arrive until December 1970.

PDP-7 Unix and the arrival of the PDP-11/20

By the time the PDP-11/20 arrived, Unix on the PDP-7 was self-hosting and provided many of the constructs and features that still exist on today's Unix systems. (The term *Unix* was coined in 1970, most likely by Brian Kernighan.) The file system provided directories and files, and file metadata was kept separate from the file's data in an array of information nodes (i-nodes), with the exception of file names. The latter were stored in the directory structure, along with the index numbers of the corresponding i-nodes. In this early stage of development, path names did not yet exist; file names were taken relative to the current directory. To mitigate the lack of path names, the system provided hard links, whereby a file could have multiple names in various directories, all of equal importance.[5]

Although the file system from 1970 bears similarity to current Unix file systems, the process control mechanism underwent significant evolution. In the earliest PDP-7 system, only one process was in memory at any time; all other processes were swapped to disk. On each command received by the shell interpreter, the shell exited and replaced itself (as a process) with the requested command. When the command exited, the OS reloaded the shell to obtain the next user's command. Despite the primitive mechanism, the system was able to support I/O redirection where the I/O of the executed command comes from or is sent to files instead of the terminal; the shell simply disconnected the terminal and opened the appropriate files before replacing itself with the requested command.[5]

The PDP-7 Unix had borrowed the open/close/read/write I/O semantics from Multics, but the Unix I/O redirection was a vast simplification over the Multics model:[5]

> Where under Unix we might say `ls > xx` to get a listing of the names of files in `xx`, on Multics the notation was
>
> ```
> iocall attach user_output file xx
> list
> iocall attach user_output syn user_i/o
> ```

Another I/O advance that Unix brought was device abstraction, which made devices appear to be ordinary files and hence amenable to the same open/close/read/write I/O operations. The kernel had a hard-coded map of i-node numbers to specific devices. That is, an I/O operation (such as a read,

write, or seek) to a file with an i-node number in the map caused a corresponding I/O operation on the raw device. Seeks on block-structured devices, however, were performed on block number instead of byte offset, as would have been the case with ordinary files.

The early Unix process-control mechanism then evolved into the framework normally associated with Unix—that of `fork()` and `exec()`—which was influenced by the Berkeley Timesharing System on the SDS 940 computer, not Multics. Thompson noted,

> We stole per-process execution [from the Berkeley Timesharing System]. You know: create a process, execute the command.... Multics wanted to do it, but it was so expensive creating a process that it ended up creating a few processes and then using them and putting them back on the shelf, then picking them up and reinitializing them. So, they never really created a process for command because it was just too expensive.[11]

During a `fork()` operation, a clone of the current running process is made, distinguished from the original process by its process ID. To accommodate the `fork()` system call, the PDP-7 system's process table size was increased, and a `fork()` system call was added that copied an image of the running process to swap, with appropriate modifications to the process table.[5] This let the PDP-7 Unix run processes detached from the keyboard and made shell scripting possible because the shell no longer had to exit to start a new process.

The arrival of the PDP-11/20 with 24 Kbytes of memory helped to overcome the constraints of the PDP-7 (only 8,192 words of 18-bit core memory). The nascent Unix system was ported from the PDP-7 assembly language to the PDP-11 assembly language. And to satisfy the original funding proposal, the *roff* text processing system was transliterated from the PDP-7 version, which itself had been derived from Jerome Saltzer's *runoff* program on CTSS.[13] The system gained useful path names for files, and the process-control mechanism was completed with the `exec()` and `wait()` system calls. By mid-1971, Unix on the PDP-11/20 was supporting three typists from AT&T's internal patent department, which used the *roff* processing system to edit and format patent applications.[5]

Unix programmer's manual, first edition

In its earliest days, Unix was a continually evolving system; if you were not one of the researchers developing the system, you often had to ask how to use it.[14] Doug McIlroy insisted that the team document the system's operation with a programmer's manual:

> [McIlroy] had a lot to do specifically with the manual.... That he insisted on a high standard in the manual meant that he insisted there was a high standard in every one of the programs that was documented.... The work that had gone into producing the manual had involved re-writing all sorts of programs in order that they should meet the same high standard. And then added to all of that, it was probably the first manual that ever had a section with bugs in it. That's a level of honesty you don't find. It wasn't that they documented the bugs but were too lazy to fix them. They fixed a lot of bugs—but some of them weren't so easy to fix, or there were uncertainties as to what they would do—so they documented that. I think a level of intellectual honesty was present in that activity that was rare.[10]

The early Unix releases were dated by the release of a new user manual. McIlroy noted that

> Cleaning up something up so you can talk about it is really quite typical of Unix. Every time another edition of the manual would be made, there would be a flurry of activity. When you wrote down the uglies, we'd say, ''We can't put this in print.'' [We had to t]ake features out, or put features in, in order to make them easier to talk about.[15]

The first edition of the Unix programmer's manual documents the system as it stood in November 1971. From the user's perspective, the system provided a hierarchical file system with directories, subdirectories, and files with up to six-character names and up to 64 Kbytes in size. It supported 16 processes, although only one running process was in memory at a time; all other processes were on the swap device. Many of the available applications would be recognizable to current Unix users:

> ar, as, cal, cat, chmod, chown, cmp, cp, date, dc, df, du, echo, ed, find, ld, ln, login, ls, mail, mesg, mkdir, mv, nm, od, pr, rm, rmdir, roff, sh, size, sort, strip, su, sum, tty, wc, who, write

Other applications included games such as blackjack, chess, and tic-tac-toe. For the system administrator, there were tools to dump and restore disk images to DECtape,

to read and write paper tapes, and to create, check, mount, and unmount file systems on removable RK03 disk packs.

The only editor available in 1971 was the ''ed'' editor, which is still the only editor guaranteed to be present on all Unix systems. In 1976, Kernighan and Plauger wrote,

> The earliest traceable version of the editor presented here is TECO, written for the first PDP-1 timesharing system at MIT [by Dan Murphy]. It was subsequently implemented on the SDS-940 as the ''quick editor'' QED by L. P. Deutsch and B. W. Lampson; see ''An online editor,'' CACM December, 1967. K. L. Thompson adapted QED for CTSS on the IBM 7090 at MIT, and later D. M. Ritchie wrote a version for the GE-635 (now HIS-6070) at Bell Labs. The latest version is ed, a simplified form of QED for the PDP-11, written by Ritchie and Thompson.[16]

From a programmer's perspective, the system offered an interactive, multiuser, preemptive multitasking environment with a process address space of 8 Kbytes (16 Kbytes were reserved for the kernel). Several languages were available, including Basic, Fortran, shell scripting, assembly language, and B, a descendant of the Basic Combined Programming Language (BCPL) that ultimately evolved into the C language.[17]

The first edition Unix kernel provided the applications programmer with 34 system calls. It is a testament to the system's design that nearly all these systems calls are still available (and used heavily) in modern Unix systems nearly four decades later, including `open()`, `close()`, `read()`, `write()`, `fork()`, `exec()`, `exit()`, and `wait()`. In particular, 24 of the system calls from the first edition Unix appear with the same system call number in current versions of the Linux kernel. The initial API design for the Unix kernel has well and truly stood the test of time.

Internally, the first edition Unix kernel consisted of approximately 4,200 lines of PDP-11 assembly code. In his 1972 study of the kernel, T.R. Bashkow noted that ''it has been mentioned parenthetically that Unix is not very modular.''[1] Kernel modularity would come in stages, first with the rewrite of the kernel into the more portable C language in 1974 and then with some significant redesign that produced the seventh edition Unix system in 1979.

Although modern Unix users would find the first edition of Unix familiar, there were some differences they would find either confusing or irritating. For example, while the system distinguishes between users, and files have separate read/write permissions for the file's owner and for the other system users, the concept of user groups did not yet exist. Although Unix supported the I/O redirection concept, several tools such as sort still required explicitly named input and output files. The shell supported the pattern-matching metacharacters `?` and `*`, but it used an external program called *glob* to perform the pattern matching.

For its time, the first edition Unix OS provided a remarkably powerful development milieu.

The Unix kernel was itself divided into two sections: cold and warm Unix kernels. Depending on the setting of the console switches at boot time, cold Unix would boot and initialize the RF-11 drum device with a minimal root file system, including the *init* program, a shell, and the required device files. Alternatively, warm Unix would boot and bring the system up into multiuser mode. And because the PDP-11/20 provided no memory protection against kernel corruption by the running process, it was considered a courtesy for a programmer to yell ''a.out?''—the name of the Unix assembler's default output file—before running a new executable for the first time. This gave the other users a chance to save any files they were editing.[18]

For its time, the first edition Unix OS provided a remarkably powerful development milieu. With a kernel weighing in at 4,200 lines of assembly code and occupying only 16 Kbytes of main memory, Unix could support half a dozen simultaneous users, giving them an interactive editing, typesetting, and programming environment. The programmer had access to several programming languages and a set of well-designed system calls; the latter provided elegant file I/O semantics (including I/O redirection and access to devices) and a flexible process management model.

Restoring the first edition Unix

Unix's history stretches back nearly 40 years to its initial design in 1969. This time span has lead to the loss of many of the artifacts, documentation, and memories from the early eras of Unix development and use. Fortunately, there have been several efforts to capture and document Unix's history.[5,7,19–21] In 1995, the TUHS (originally named the PDP-11 Unix Preservation Society) was founded with a charter to preserve, maintain, and restore historical and non-mainstream Unix systems. TUHS has been successful in unearthing artifacts from many important historical Unix systems, including system and application source code, system and application executables, user manuals and documentation, and images of populated file systems.

It had been assumed that the earliest documented version of Unix from 1971, named after the first edition of the programmer's manual, had vanished forever. However, a printed listing of the assembly language source code to the first edition Unix kernel was unearthed in 2006. After much effort, the earliest documented Unix system has been made to run again.

Issues in restoring the software

Most programming practitioners are aware of the concept of *bit rot*, whimsically defined by the *New Hacker's Dictionary* as a ''hypothetical disease, the existence of which has been deduced from the observation that unused programs or features will often stop working after sufficient time has passed, even if 'nothing has changed.'''[20] And yet, if the software has not changed and exists in pristine form on some medium, then what causes the decay?

Bit rot is not a function of the software, rather it is a function of the change in the software's environment. Software does not exist *in vacuo*. For it to execute, software requires

- hardware on which to execute it, or a suitable hardware simulation;
- documentation describing how to install, configure, and run the software;
- ancillary files required for the software to run (such as configuration, database, input files); and
- if the software is in source form, tools that can reassemble or recompile the source into executable form.

The last requirement necessitates some amount of recursion—that is, the tools to reassemble or recompile software are themselves pieces of software, and require their own hardware, documentation, ancillary files, and recompilation tools.

Restoration of legacy software to working order therefore depends on the restoration of the environment in which the software executed. In many cases, a complete restoration of either or both is impossible, which raises several issues:

- Should bugs or faults found in the legacy software be fixed, or should the environment be modified to work around the problem?
- If the software is incomplete, is it acceptable to write new components in order to complete it?
- Similarly, if the environment is incomplete, is it acceptable to rebuild the missing pieces?
- Even if the software is intact, is it acceptable to use non-contemporaneous tools such as a modern assembler or cross-compiler to reconstruct the executable from the historical source code?

The effort to restore the source code of the first edition Unix kernel faced by all these issues and many others specific to the software and its environment.

Restoring the kernel

In 2006, Al Kossow from the Computer History Museum unearthed and scanned a document by T.R. Bashkow titled ''Study of Unix,'' dated September 1972.[23] The document covers ''the structure, functional components and internal operation of the system.'' The study also includes what appeared to be a complete listing of an assembly version of the Unix kernel. A second document was also found that appeared to be the handwritten notes made by J. DeFelice in preparation of Bashkow's study.[24] Dates within the latter document indicate that the analysis of the Unix kernel began in January 1972, implying that the kernel being studied was the first edition.

While the paper listing of the first edition Unix kernel (and its analysis) was interesting in its own right, its discovery lead to the inevitable question: Can the kernel be restored to working order? There were several factors that suggested that the kernel's environment could be restored, including a full copy of the programmer's manual,[25] the SIMH simulator for the PDP-11/20,[26] a number of userland

executables that date from mid-1972, and fragments of assembly source for the userland executables. (*Userland* refers to the system software that does not execute in kernel mode.)

On initial reflection, three main hurdles appeared to stand in the way of the restoration:

- there was no suitable Unix assembler to recreate the kernel executable,
- there was no extant file-system image or tool to create a file system, and
- no boot chain existed to load a kernel executable into the PDP-11/20's memory and prepare the hardware for the kernel's operation.

For these reasons, the restoration effort was abandoned. In April 2008, however, new courage was found and the kernel's restoration began in earnest.

Initially, a team of approximately 10 people undertook the task of scanning the paper document with OCR software to convert it into machine-readable format, including the arduous task of visually inspecting the output to correct mistakes. This revealed a few handwritten comments on the paper listing that contradicted the source code comments and did not inspire confidence in the venture's success. As one of the team commented, ''I have already noticed quite a few errors in the listing, so it's not clear that the [listing] was something that actually ran, or whether it had been retyped by somebody.''

With the kernel's source code in machine-readable form and verified, the search for an assembler began. The seventh edition Unix assembler was known to work, but it was a PDP-11 executable; fortunately an existing PDP-11 userland emulator, *Apout*, allowed the assembler to run on top of current platforms. The assembler did not, however, produce executables with the a.out structure that the early kernel required. This was solved by a slight restructuring of the initial kernel code and with a tool to post-process the assembler's output.

Analysis of the kernel listing and the programmer's manual indicated that first edition Unix required a PDP-11/20 with 24 Kbytes of core, RF-11 and RK03 disks, up to eight teletypes on a DC-11 interface, and a TC-11 DECtape device. The SIMH simulator was configured to provide this hardware environment. Following the successful assembly of the kernel into an executable binary, the next problem was to recreate the boot sequence. In the long term, a full boot chain needed to be rewritten from scratch, but first the SIMH simulator was commanded to load the kernel into the appropriate memory locations, initialize a few registers, and begin execution at the octal address 0400.

Executing the kernel's first instructions revealed further problems. The kernel required the PDP-11/20 to contain a numerical coprocessor known as a KE11A, which SIMH did not simulate. Restoration halted while KE11A support was added to SIMH, using the original PDP-11/20 processor manual.[27] With a working KE11A, the kernel ran into an infinite loop; a decrement instruction, missing from the paper listing, was intuited and added. This permitted the kernel, in cold Unix mode, to write what appeared to be a minimal file system onto the RF-11 device, containing a number of device files, the *init* program, and the command shell. Two more errors in the kernel source were fixed, allowing the kernel, in warm Unix mode, to run the *init* program, output a login prompt, and invoke a shell on successful login of the root user.

Technically, the kernel restoration was a success, but with no applications the executing system was unusable. It was possible, but not certain, that the Unix userland executables from mid-1972 would run on top of the first edition kernel. Using the existing minimal file system and the programmer's manual, a standalone program was written to create and populate a file-system image with the executables. Inspection of the executables also revealed that some of them adhered to the a.out structure used in the second edition Unix, dated June 1972. It was decided to modify the first edition kernel to support the newer a.out structure. With the populated file system and later a.out support, the restored Unix system was again usable, with nearly all the documented system tools, text editor and document processing tools, and programming languages.

Two important system tools were still missing, however. One was the *mkfs* tool to build new file systems from within the running first edition kernel; the external file-system creation tool was able to fill this deficiency. The second missing tool was the `mount` command needed to attach the file system on the RK03 disk, which contained the users' home directories. The solution to the latter was to modify the *init*

program to add a call to the `mount` system call to attach the RK03 file system at boot time. A proper `mount` command would be trivial to recreate, but the `mkfs` command less so.

The first edition Unix system was now running in single-user mode, but attempts to configure the system to enable multiple user logins resulted in the system hanging at boot time. Again, a deficiency was found in the SIMH simulator; it did not have support for the DC-11 serial interface device. Support for the DC-11 device was added to SIMH, using the 1972 PDP-11 peripherals handbook,[28] and the first edition Unix was able to run again in multiuser mode. The kernel restoration was essentially complete.

The first edition Unix did not contain a C compiler, but by the second edition in June 1972, the system contained a compiler that recognized an adolescent form of the C language.[7] The collection of mid-1972 executables contained C compiler binaries, and elsewhere Dennis Ritchie had unearthed their source code. The first edition kernel had already been modified to recognize the binaries' a.out structure, but the kernel only provided executables with 8 Kbytes of address space and the C compiler required 16 Kbytes.

At this stage, the restoration team debated whether to modify the first edition kernel to allow the second edition C compiler to run. One argument held that, in doing so, the kernel would no longer be a first edition Unix. An opposing argument held that the system had already been modified to run executables later than the first edition and so it was already a hybrid system. Pragmatism eventually won the day, and the kernel and file system were modified to provide a 16-Kbyte process address space and 16-Kbyte swap areas on the disk. With these modifications, the Unix system was able to run the C compiler, and the C compiler was able to recompile itself.

Two final tasks in the kernel restoration were left. One was the reconstruction of the boot chain. Although not technically necessary, the boot chain was an integral part of the first edition system; it has been recreated and is now a standard part of the restored system. The second task, now completed, was to get the changes made to SIMH put back into the main SIMH source tree to make them available to other SIMH users.

While restoring the first edition Unix, the team chose a pragmatic approach. Specifically, it addressed the issues regarding software restoration that I raised in the previous section as follows:

- Bugs and faults in the original software should be fixed, as long as the changes are well documented. The restoration effort faced errors and omissions in the kernel listing, which suggests that these occurred during the preparation of the kernel's study. In the restoration, text documents were kept that accurately reflect each of the pages of the kernel listing in Bashkow's study. They were then transformed into the 11 documented source files that contain the kernel source code. Finally, these kernel files had patches applied to them to fix any errors and modify the kernel to support later a.out executables and the C compiler. Therefore, there is a clear separation between the code in the paper listing and the code that produces a running system.
- It was deemed acceptable to write new components to complete the Unix restoration; in particular, the code for the system's boot sequence was rewritten, and the *init* program was extended to mount the RK03 file system.
- The Unix restoration would not have been successful without the addition of KE11A and DC-11 support to the SIMH simulator and the construction of a tool to create and populate first edition file systems. The team deemed it acceptable to (re)build components to complete the software environment.
- The team was stymied with the lack of a first edition assembler to recreate the kernel executable from the kernel source code. If the goal was to restore the kernel to a running state, then there was no choice but to use a later assembler. Originally, the team used the assembler from the seventh edition Unix to build the kernel. With the second edition assembler now working, it can rebuild the first edition kernel, making the use of this assembler more palatable than the seventh edition assembler.

The Unix restoration effort has produced a number of tools to assemble the kernel source, create suitable file-system images, build and configure a suitable PDP-11/20

simulator, and provide an automated workflow such that other people can recreate a working first edition Unix system. The kernel listing from Bashkow's study has been converted into machine-readable form verbatim, and the changes required to transform the listing into working kernel source are well-documented and kept separate from the original listing. Interested reader can find the results of the restoration online at http://unix-jun72.googlecode.com.

Although the Unix restoration effort faced a significant amount of work and many hurdles, it should be noted that good fortune and luck smiled upon the project. The task could never have been achieved without the preservation of the earliest Unix documentation, the discovery of userland executables from mid-1972, or the efforts to preserve and restore other editions of the Unix operating system.[29]

Observations

Restoring the first edition Unix to working order revealed aspects of the system that were not readily apparent from either the historical literature or a reading of the assembly source code.

The first edition Unix was known to support multiple users concurrently on the one machine—for example, ''we supported three typists from the Patent department''[5]—but the exact number of supported users was unknown. The system restoration revealed that Unix supported up to nine concurrent users on the PDP-11/20, although the limit of 16 processes in the system at any time would have severely impeded those nine users' ability to perform useful work.

Internally, the first edition Unix supported uppercase and lowercase ASCII characters, and all command names were lowercase. The terminals used to interface to the system, however, were uppercase-only Teletype ASR-33s. The terminal driver in the kernel had code to convert uppercase letters entered by the user into lowercase letters. Upon output, the terminal driver code inserted delays to ensure that no more than 10 characters per second were delivered to the ASR-33s, ensuring that output was not lost due to mechanical delays. Both the input translation and the output delays were removed in the restoration to make the system more acceptable to current users, but the patch is easily reversed to return the system to its authentic state.

The first edition's hardware platform, the PDP-11/20, did not provide hardware for memory management nor memory protection; the kernel was loaded into memory at a fixed physical address, and the running process was kept at a fixed physical address adjacent to the kernel. Although the system's architectural features (the file system and process management) were previously documented, the system restoration revealed that the designers had chosen to use a true TRAP-based system call mechanism as opposed to subroutine jumps via a fixed-location dispatch table. In a private email, Ritchie notes that ''a clear system versus user distinction was another of the things that was absorbed, not only from Multics but also from CTSS and the Berkeley SDS 940 system . . . It seemed only natural to do things this way.''

Finally, the restoration demonstrates conclusively that, as early as 1971, Unix provided a comfortable and powerful time-sharing environment for text processing with a line-based editor and the sophisticated *roff* text processor, and with development tools for several programming languages. The system is responsive and immediate, in contrast to contemporary batch systems. The command-line shell and the toolbox approach that this system introduced remains a core user interface metaphor in Unix and Linux systems after four decades of development.

Conclusion

From the beginning, the Unix operating system combined a set of simple and elegant components into a sophisticated environment for both users and developers. Nearly four decades later, these components are still the core of all Unix and Unix-derived systems: multitasking via processes, a single hierarchical file system, simple and flexible I/O redirection, device abstraction, byte-oriented files, and a command shell.

The first edition Unix, documented by the 1971 programmer's manual and arriving just a year after Unix was conceived, shows both a remarkable maturity of design and an economy of code embodying this design. Its rediscovery as a code listing in Bashkow's review of the early Unix kernel lets us examine the system's intricacies, but only in the form of a taxidermy exhibit. With a great deal of effort and fortune, this code listing has been restored to working order, returning the

first edition Unix to a working artifact of computing history.

Acknowledgments

I thank all the members of the Unix restoration project including Johan Beisser, Tim Bradshaw, Ralph Logan, Doug Merritt, James Markevitch, Brad Parker, and most particularly Tim Newsham.

References

1. Eric Lévénez, ''Unix History Timeline,'' 2008; http://www.levenez.com/unix/.
2. K. McKusick et al., *The Design and Implementation of the 4.4 BSD Operating System,* Addison-Wesley, 1996.
3. M. K. McKusick, *Open Sources: Voices from the Open Source Revolution,* O'Reilly, 1999, chap. 3.
4. L. Torvalds, *Open Sources: Voices from the Open Source Revolution,* O'Reilly, 1999, chap. 7.
5. D.M. Ritchie, ''The Evolution of the Unix Time-Sharing System,'' *BSTJ,* vol. 63, no. 8, 1984, pp. 1577–1594.
6. F.J. Corbatò, M.M. Daggett, and R.C. Daley, ''An Experimental Time-Sharing System,'' *Proc. Spring Joint Computer Conf.,* ACM Press, 1962, pp. 335–344.
7. P.H. Salus, *A Quarter Century of Unix,* Addison-Wesley, 1994.
8. F.J. Corbatò and V.A. Vyssotsky, ''Introduction and Overview of the Multics System,'' *IEEE Annals of the History of Computing,* vol. 14, no. 2, 1992, pp. 12–13.
9. N. Peirce, ''Putting Unix in Perspective: An Interview with Victor Vyssotsky,'' *Unix Rev.,* vol. 3, no. 1, 1985, pp. 58–70, 102–106.
10. M. Mahoney, ''Interview with A.G. (Sandy) Fraser,'' Unix Oral History Project, 1989; http://www.princeton.edu/~hos/mike/transcripts/fraser.htm.
11. M. Mahoney, ''Interview with Ken Thompson,'' Unix Oral History Project, 1989; http://www.princeton.edu/~hos/mike/transcripts/thompson.htm.
12. B.W. Lampson, W.W. Lichtenberger, and M.W. Pirtle, ''A User Machine in a Time-Sharing System,'' *Proc. IEEE,* vol. 54, Dec. 1966.
13. J.H. Saltzer, *TYPSET and RUNOFF: Memorandum-editor and type-out Commands,* tech. report MAC-M-193, MIT Computation Center, 1964.
14. M. Mahoney, ''Interview with Joseph H. Condon,'' Unix Oral History Project, 1989; http://www.princeton.edu/~hos/mike/transcripts/condon.htm.
15. M. Mahoney, ''Interview with D. McIlroy,'' Unix Oral History Project, 1989; http://www.princeton.edu/~hos/mike/transcripts/mcilroy.htm.
16. B. Kernighan and P.J. Plauger, *Software Tools,* Addison-Wesley, 1976.
17. D.M. Ritchie, ''The Development of the C Language,'' *Proc. 2nd History of Programming Languages Conf.,* ACM Press, 1993, pp. 671–698.
18. D.M. Ritchie, ''Odd Comments and Strange Doings in Unix,'' 2002; http://cm.belllabs.com/cm/cs/who/dmr/odd.html.
19. M. Hauben and R. Hauben, *Netizens: On the History and Impact of Usenet and the Internet,* IEEE CS Press, 1997.
20. M. Mahoney, Unix Oral History Project, 1989; http://www.princeton.edu/~mike/expotape.htm.
21. D. Ritchie and K. Thompson, ''The Unix Time-Sharing System,'' *Comm. ACM,* vol. 17, no. 7, 1974, pp. 365–375.
22. E.S. Raymond, *The New Hacker's Dictionary,* MIT Press, 1996.
23. T.R. Bashkow, ''A Study of the Unix Operating System,'' Sep 1972; http://www.bitsavers.org/pdf/bellLabs/unix/PreliminaryUnixImplementationDocument_Jun72.pdf.
24. J. De Felice, ''Unix Kernel Routine Descriptions,'' Apr 1972; http://bitsavers.org/pdf/bellLabs/unix/Kernel_Subroutine_Descriptions_Mar72.pdf.
25. K. Thompson and D.M. Ritchie, *Unix Programmer's Manual,* Nov. 1971; http://bitsavers.org/pdf/bellLabs/unix/Unix_ProgrammersManual_Nov71.pdf.
26. M. Burnett and R. Supnik, ''Preserving Computing's Past: Restoration and Simulation,'' *Digital Technical J.,* 1996, pp. 23–38.
27. Digital Equipment, ''PDP-11/20 Processor Handbook,'' 1971; http://bitsavers.org/pdf/dec/pdp11/handbooks/PDP1120_Handbook_1972.pdf.
28. Digital Equipment, ''PDP-11 Peripherals Handbook,'' 1972; http://www.bitsavers.org/pdf/dec/pdp11/handbooks/PDP11_PeripheralsHbk_1972.pdf.
29. W. Toomey, ''Saving Unix from /dev/null,'' *Proc. AUUG Open Source Conf.,* AUUG, 1999.

Warren Toomey is an assistant professor in the School of IT at Bond University, where he researches and lectures in programming, networks, operating systems, and computer security. Toomey has a PhD in network performance from the University of New South Wales, Australia. Contact him at wtoomey@staff.bond.edu.au.