CONTENTS

Call for Papers: POSIX Portability Workshop .......................................................... 3
Call for Papers: Winter 1988 USENIX Conference .................................................. 4
Call for Papers: Summer 1988 USENIX Conference ................................................ 5
Computer Graphics Workshop .................................................................................... 6
Multiple Programs in One UNIX Process .................................................................. 7
  Don Libes

tar vs. cpio .................................................................................................................. 13
How To Write a UNIX Daemon .................................................................................... 17
  Dave Lennert

Call for Papers: EUUG Spring 1988 Conference ....................................................... 24
Book Reviews ............................................................................................................. 25
  The Design of the UNIX Operating System ......................................................... 25
    Marc D. Donner
    Josiah C. Hoskins

UUNET Progress Report ............................................................................................. 28
The 1988 Election of the USENIX Board of Directors ............................................... 30
Call for Participation: USENIX C++ Workshop ....................................................... 30
Summary of the Board of Directors' Minutes New Orleans, 26-27 March 1987 ....... 31
Summary of the Board of Directors' Meeting Phoenix, June 7, 8, 10, 1987 .............. 32
Future Meetings .......................................................................................................... 34

Publications Available ................................................................................................. 35

4.3BSD UNIX Manuals ................................................................................................. 36
4.3BSD Manual Reproduction Authorization and Order Form .................................. 37
Local User Groups ....................................................................................................... 38

The closing date for submissions for the next issue of ;login: is August 28, 1987
NOTICE

;login: is the official newsletter of the USENIX Association, and is sent free of charge to all members of the Association.

The USENIX Association is an organization of AT&T licensees, sub-licensees, and other persons formed for the purpose of exchanging information and ideas about UNIX and similar operating systems and the C programming language. It is a non-profit corporation incorporated under the laws of the State of Delaware. The officers of the Association are:

President  Alan G. Nemeth  usenixlagn
Vice-President  Deborah K. Scherrer  usenixlscherrer
Secretary  Waldo M. Wedel  usenixlwedel
Treasurer  Stephen C. Johnson  usenixlscj
Directors  Rob Kolstad  usenixlkolstad
          Marshall Kirk McKusick  usenixlmckusick
          John S. Quarterman  usenixljsq
          David A. Yost  usenixlday

Executive Director  Peter H. Salus

The editorial staff of ;login: is:

Editor and Publisher  Peter H. Salus  usenixlpeter
Technical Editor  Kevin Baranski-Walker  usenixlkbc
Copy Editor  Michelle Dominijanni  (masscomp,usenix)lmmpl
Production Editor  Tom Strong  usenixlstrong

Other staff members are:
Betty J. Madden  Office Manager
Emma Reed  Membership Secretary
Jordan Hayes  Technical Consultant

USENIX Association Office
P.O. Box 2299
Berkeley, CA  94710
(415) 528-8649
(ucbvax,decvax)\usenix\office

Judith F. DesHarnais  Conference Coordinator
USENIX Conference Office
P.O. Box 385
Sunset Beach, CA  90742
(213) 592-3243 or 592-1381
(ucbvax,decvax)\usenix\judy

John L. Donnelly  Tutorial & Exhibit Manager
USENIX Exhibit Office
Oak Bay Building
4750 Table Mesa Drive
Boulder, CO  80303
(303) 499-2600
(ucbvax,decvax)\usenix\johnd

Contributions Solicited

Members of the UNIX community are encouraged to contribute articles to ;login:. Contributions may be sent to the editors electronically at the addresses above or through the U.S. mail to the Association office. The USENIX Association reserves the right to edit submitted material.

;login: is produced on UNIX systems using troff and a variation of the -me macros. We appreciate receiving your contributions in n/troff input format, using any macro package. If you contribute hardcopy articles please send originals and leave left and right margins of 1" and a top margin of 1\(\frac{1}{2}\)" and a bottom margin of 1\(\frac{3}{4}\)".

Acknowledgments

The Association uses a SUN 3/180S running SUN OS for support of office and membership functions, preparation of ;login:, and other Association activities.

Connected to the SUN is a QMS Lasergrafix 800 Printer System donated by Quality Micro Systems of Mobile, Alabama. It is used for general printing and draft production of ;login: with ditroff software provided by mt Xinu.

This newsletter is for the use of the membership of the USENIX Association. Any reproduction of this newsletter in its entirety or in part requires written permission of the Association and the author(s).

1UNIX is a registered trademark of AT&T.

2SUN is a trademark of Sun Microsystems, Inc.
*Lasergrafix is a trademark of Quality Micro Systems.
Call for Papers
POSIX Portability Workshop

Berkeley Marina Marriott
October 22-23, 1987

This USENIX workshop will bring together system and application implementors faced with the problems, “challenges,” and other considerations that arise from attempting to make their products compliant with IEEE Standard 1003.

The first day of the workshop will consist of presentations of brief position papers describing experiences, dilemmas, and solutions. On the second day it is planned to form smaller focus groups to brainstorm additional solutions, dig deeper into specific areas, and attempt to forge common approaches to some of the dilemmas.

Suggestions for topic areas and position papers include:

C Language Issues
Networked/Distributed Implementations
Timer resolution, ranges
Conformance verification
Job control, process groups
Implications for user interfaces
Internationalization
Pipes and FIFOs
Signals
Security concerns
Limits: documentation and inquiry
Implications for commands

Position papers must be submitted by August 15, 1987 to:

Jim McGinniss
Digital Equipment Corporation
Continental Boulevard MK02-1/HIO
Merrimack, NH 03054

(603) 884-5703
decvaxjmcg
jmcg@decvax.DEC.COM

For registration or hotel information, contact:

Judith F. DesHarnais
USENIX Conference Coordinator
PO Box 385
Sunset Beach, CA 90742

(213) 592-3243
usenix@judy

Volume 12, Number 4
July/August 1987
3
Call for Papers
Winter 1988 USENIX Conference
Dallas, Texas
February 9-12, 1988

Please consider submitting an abstract for your paper to be presented at the Winter 1988 USENIX conference. Abstracts should be around 250-750 words long and should emphasize what is new and interesting about the work. The final typeset paper should be 8-12 pages long.

The Winter conference will be four days long: two days of tutorials only and two days of papers only.

Suggested topic areas for this conference include (but are not limited to):

- Electronic Publishing
- Novel Kernels
- New Software Tools
- New Applications
- System Administration
  (including distributed systems and integrated environments)
- Security in UNIX
- Future Trends in UNIX

This conference may include a “miscellaneous” session which will include those papers which normally do not fit into normal tracks. Vendor presentations should contain technical information and be of interest to the general community.

Abstracts are due by October 23, 1987; papers absolutely must be submitted by January 4, 1988. Notifications of acceptance of abstracts will be sent out by November 6. Papers that do not meet the promise of their abstract will be rejected. Talks will be given on all papers published in the Proceedings; failure to submit a paper for an abstract will result in forfeiture of the talk.

Please contact the program chairman for additional information:
Rob Kolstad
CONVEX Computer Corporation
701 Plano Road
Richardson, TX  75081
214-952-0351 (W)
214-690-1297 (H)
214-952-0560 (FAX)
{usenix,ihnp4,uiucdcs,allegra,sun}@convex!kolstad

Please include your network address (if available) with all correspondence. It should be an ARPANET (EDUNET, COMNET), BITNET, or CSNET address or a UUCP address relative to a well-known host (e.g., mcvax, ucbvax, decvax, or, ihnp4).
Call for Papers
Summer 1988 USENIX Conference
San Francisco
June 20-24, 1988

Papers in all areas of UNIX-related research and development are solicited for formal
review for the technical program of the 1988 Summer USENIX Conference. Accepted
papers will be presented during the three days of technical sessions at the conference and
published in the conference proceedings. The technical program is considered the leading
forum for the presentation of new developments in work related to or based on the UNIX
operating system.

Appropriate topics for technical presentations include, but are not limited to:

- Kernel enhancements
- UNIX on new hardware
- User interfaces
- UNIX system management
- The internationalization of UNIX
- Performance analysis and tuning
- Standardization efforts
- UNIX in new application environments
- Security
- Software management

All submissions should contain new and interesting work. Unlike previous technical
programs for USENIX conferences, the San Francisco conference is requiring the submission
of full papers rather than extended abstracts. Further, a tight review and production cycle
will not allow time for rewrite and re-review. (Time is, however, scheduled for authors of
accepted papers to perform minor revisions.) Acceptance or rejection of a paper will be
based solely on the work as submitted.

To be considered for the conference, a paper should include an abstract of 100 to 300
words, a discussion of how the reported results relate to other work, illustrative figures, and
citations to relevant literature. The paper should present sufficient detail of the work plus
appropriate background or references to enable the reviewers to perform a fair comparison
with other work submitted for the conference. Full papers should be 8-12 single spaced
typeset pages, which corresponds to roughly 20 double spaced, unformatted, typed pages.
Format requirements will be described separately from this call. All final papers must be
submitted in a format suitable for camera-ready copy. For authors who do not have access
to a suitable output device, facilities will be provided.

Four copies of each submitted paper should be received by February 19, 1988; this is an
absolute deadline. Papers not received by this date will not be reviewed. Papers which
clearly do not meet USENIX's standards for applicability, originality, completeness, or page
length may be rejected without review. Acceptance notification will be by April 4, 1988, and
final camera-ready papers will be due by April 25, 1988.

Send technical program submissions to:

Sam Leffler
SF-USENIX Technical Program
PIXAR
P.O. Box 13719
San Rafael, CA 94913-3719
415-499-3600
ucbvax!sfusenix
The Fourth USENIX Computer Graphics Workshop will be held at the Boston Marriott Cambridge in Cambridge, MA, October 8 and 9, 1987, with a no-host reception on the evening of October 7.

Registration will be $200 per attendee and must be paid in advance. There will be no on-site registration.

There is a special hotel rate for workshop attendees of $115 per night, single or double. Call the Marriott direct for reservations: 617-494-6600. Be sure to mention that you are a USENIX Workshop attendee. The Marriott has a strict cut-off of September 16 for its special rate. Reservations made after that date will be on a space and rate available basis.

Partial Program

The BRL CAD Package – Michael John Muuss & Phillip Dykstra (BRL)
REMRT – A Network Distributed and Parallel Ray-Tracer – Michael John Muuss (BRL)
The Definition and Ray-tracing of B-Spline Objects in a Combinatorial Solid Geometry Modeling System – Paul R. Stay (BRL)
More Music Software for UNIX – Michael Hawley (MIT)
Dynamics for Everyone – Jane Wilhelms (UCSC)
Distributed Computation for Computer Animation – John W. Peterson (Utah)
It’s all done with Smoke and Mirrors – The Face Saver Project –
David Yost & Lou Katz (Consultants)
Paint Systems and Images of Arbitrary Size and Shape –
Ken Knowlton & Lou Katz (Consultants)
Hairy Brushes – Steve Strassman (MIT)

For further program information, contact:
Tom Duff at research!td or Lou Katz at ucbvax!lou.

For registration information, contact:
Judith F. DesHarnais
USENIX Conference Office
P.O. Box 385
Sunset Beach, CA 90742
(213) 592-3243
cb!judy

NOTE: Make your hotel reservations on or before September 16!
Abstract

A small operating system (XINU) was ported to UNIX 4.2BSD. The entire operating system runs as a single UNIX process. The code is approximately 1000 lines of C (including comments) and 6 lines of assembler.

All of the code is user-level, and thus presents a system easy to examine, understand, and experiment with further.

The code has been used as a base for an application of several cooperating processes communicating through global variables. Alternatively, the system provides semaphores and messages for interprocess communication.

Background – Why Did We Need This?

This project fell out of a recent porting effort at NBS. The original desire was to move an application from a non-UNIX computer to a UNIX computer. The non-UNIX computer ran a simple home-brewed operating system the details of which are unimportant except that it provided interprocess communication through global variables. While 4.2 promised shared memory, it failed to deliver on this. (This has since been remedied by [Libes 85].) To quote from the manual page for mmap(2) [Joy 83]:

DESCRIPTION

N.B.: This call is not completely implemented in 4.2.

Taking the cryptic advice, we decided that it might be possible to port the entire operating system and application as a single UNIX process.

This proved to be possible with the help of several recent enhancements of 4.2 UNIX including sub-second interval timers and non-blocking I/O. Our first implementation did not require separate process stacks, and we realized that by adding them, we would have a tool of much more generality. Before proceeding much further, we quickly realized the similarity to XINU as presented by [Comer 84]. (Other approaches are discussed by [Kepecs 85] and [Tevanian 87].)

XINU

In Operating System Design, The XINU Approach Douglas Comer presents a layered and modular operating system. In contrast to other operating system texts which compare and contrast a variety of algorithms for typically only the most interesting tasks in an operating system, Comer chooses one technique for each problem, usually the most straightforward one or that leading to the simplest presentation. (Alternatives are often proposed in the exercises at the end of each chapter.)

The book is further unique in discussing every last aspect of a single implementation. This implementation is XINU. The entire source of XINU is in the text, including the machine dependent code for running XINU on a Digital Equipment Corporation LSI 11/2 (a microcomputer version of the PDP 11).
Based on Comer's unusually well-written text, we felt that it might be possible to bring up XINU on top of UNIX. Such a system would be able to provide the concurrency and shared variables that our original application needed, and at the same time be immediately useful to others, since it was already well-documented.

In fact, it was not hard. Our task was much easier than Comer's in that we had a complete set of device drivers already. This included a file system and terminal interface. We also did not have to worry (much) about operating system startup and C start up.

It took approximately 3 hours to type in the necessary parts of XINU. This included process management and utilities for fifo and priority queues. Our initial version of XINU did not use a real-time clock. Processes had to explicitly give up control (through calls to wait, sleep, etc). During that time, it was possible to use setjmp/longjmp to switch between processes. setjmp/longjmp saves/restore the registers including the pc (program counter) and sp (stack pointer), and also the signal mask (used as an interrupt mask).

**Process Rescheduling - resched()**

XINU processes call resched (via wait, sleep, etc) to give up control temporarily of the processor to a ready process. Here is the code fragment where an old process gives control to a new process.

```c
/* _resched.c - reschedule and context switch processes

_resched() {
  .
  /* old process is running */
  if (0 == setjmp(optr->regs)) longjmp(nptr->regs,1);
  /* new process resumes here */
  .
}
```

Each process has a structure describing its state whenever the process is not currently executing (analogous to the _u structure in the UNIX kernel). In the above code fragment, nptr points to the new process to begin executing. optr points to the old process that is going to be suspended. The field regs is the register save area. All that is necessary to switch processes is to save the current register values in optr, and restore the old register values in nptr.

setjmp saves most of the registers including the pc and sp. However, it does not save the condition codes. This is because setjmp and longjmp are never immediately followed by a test of the condition codes.

Using the 4.2BSD interval timer, we added a real-time clock. The real-time clock could interrupt computations anywhere, including the case where the conditions code had been set but not tested. At this point, it became necessary to do context switches ourselves. That required a small number of assembler statements.

The following amended version of resched (for an MC68000) can be called from interrupt handlers as well as user processes.

```c
/* _resched.c - reschedule and context switch processes */
/* Since we can't pass parameters to rte from resched, we use these */
/* variables that are global to both routines. */
static int *new_sp; /* new stack pointer register to be loaded */
static int (*new_pc)(); /* new program counter register to be loaded */
static int new_signal_mask; /* new signal mask to be loaded */
void _resched()
{
    register struct pentry *optr; /* pointer to old process entry */
    register struct pentry *nptr; /* pointer to new process entry */
```
/* no switch needed if current process priority higher than next */
if (((optr = &_proctab[_currpid])->pstate == PRCURR) &&
    (lastkey(_rdytail)<optr->pprio))
    return;
/* if the old process was still runnable, mark READY */
if (optr->pstate == PRCURR) {
    optr->pstate = PRREADY;
    _insert(_currpid,_rdyhead,optr->pprio);
}
/* remove highest priority process at end of ready list */
nptr = &_proctab[_getlast(_rdytail)];
if (optr->pstate == PRCURR) /* mark it currently running */
#endif
ifdef RTCLOCK
/* schedule an interrupt for the end of a quantum or the next event */
/* in the sleep queue, whichever is sooner */
_start_itimer(_slnempty && (*_sltop < QUANTUM))?*_sltop:QUANTUM);
#endif
/* ctxsw(optr->pregs,nptr->pregs);*/
/* at this point, optr->pregs == a5a, nptr->pregs = a4a */
/* save all registers in optr->pregs */
asm("moveml #0xffff,a5a"); /* save all the registers */
asm("movl #OLDPROC,a5a(64)"; /* change pc and save it */
optr->signal_mask = sigblock(O); /* save old interrupt reg */
/* we have completed putting the old process to bed */
/* now restart the new process */
/* prepare pc, sp and interrupt mask for rte() to use */
new_sp = nptr->sp; /* movi a4a(60),_new_sp */
new_pc = nptr->pc; /* movi a4a(64),_new_pc */
new_signal_mask = nptr->signal_mask;
/* load rest of registers directly except for a7 (sp) */
asm("moveml a4a,#0xffff"); /* restore d0-d7,a0-a3 */
asm("moveml a4a(52),#0x6000"); /* restore a5-a6 */
asm("movl a4a(48),a4a"); /* restore a4 */
kil(getpid(),RTE);
/* old process returns here */
asm("OLDPROC:";
}

Notice that the scheduler here is very simplistic. Highest priority processes are selected round-robib. (More complex schedulers might use per-process quantums as well as reassigning priorities.)

/* if the old process was still runnable, mark READY */
if (optr->pstate == PRCURR) {
    optr->pstate = PRREADY;
    _insert(_currpid,_rdyhead,optr->pprio);
}
/* remove highest priority process at end of ready list */
nptr = &_proctab[_getlast(_rdytail)];

An interval timer is then scheduled to occur at the end of the next quantum or for the first scheduled sleeping process, whichever is sooner.

/* schedule an interrupt for the end of a quantum or the next event */
/* in the sleep queue, whichever is sooner */
_start_itimer(_slnempty && (*_sltop < QUANTUM))?*_sltop:QUANTUM);

The real-time clock is simulated using the 4.2 interval timer. Rather than generating constant interval clock ticks, the interval timer is only set for known events (i.e. quanta and sleeping processes). This reduces the number of clock interrupts significantly.
Context Switching – rte()

Comer describes (p. 59) the LSI instruction rtt (return from trap) which reloads the pc and ps (processor status register) at the same time. We have a similar problem, although rather than reloading the ps, we want to reload the signal mask, sc_mask. The solution is to artificially provoke a signal (via kill) which at termination executes a rte (return from exception). This is the 68000's analog to the 11/2's rtt.

```c
/* rte() - indirectly execute 68K rte (return from exception) instruction */
/* Always called from resched(). This routine is necessary to load the */
/* signal mask at the same time as we load the new pc and sp. */
/* setjmp/longjmp is unusable as it doesn't save/restore all the registers. */
/* ARGUSED */
static void rte(int sig, int code, struct sigcontext *scp)
{
    scp->sc_sp = (int)new_sp;
    scp->sc_pc = (int)new_pc;
    scp->sc_mask = new_signal_mask;
    /* No need to reload ps, as no one looks at it anyway, upon return. */
}
```

A Simple Example

We want two XINU processes to execute simultaneously, one continuously printing "1", and the other continuously printing "2". To do it, we create two subroutines as follows:

```c
prog1()
{
    for (;;) printf("1");
}

prog2()
{
    for (;;) printf("2");
}
```

The following subroutine is all that is necessary to run them.

```c
user_main()
{
    xresume(xcreate(prog1,2000,20,"prog1",0));
    xresume(xcreate(prog2,2000,20,"prog2",0));
}
```

Compiling this together with the XINU support routines and running the executable produces the following output:

```
11111222221111122222111122222111122222...
```

xcreate takes a subroutine and creates a runnable (XINU) process, returning a process id. Passing the process id to xresume allows the process to run. The remaining parameters to xcreate are the stack size, a process priority, a tag for debugging, and a number and list of arguments passed to the process when started. Further information can be found in Comer's book.

Miscellaneous But Important Implementation Notes

The entire project took approximately two person-weeks. This included typing in the source, learning the necessary amount of both LSI 11/2 assembler (Comer's original) and a mongrel 68K/UNIX assembler provided by the vendor of our 4.2 system. Lastly, we had to figure out the undocumented C calling conventions for the 4.2 C compiler (very similar to what Comer discusses) as well as experiment with the undocumented asm statement in our C compiler.
Although we have no references on it, `asm` is a keyword in (apparently) many C compilers which allows the user to drop assembler statements into the C compiler’s assembler output. For example,

```c
foo();
asm("jsr bar"); /* bar(); */
```
calls bar after calling foo. The next logical step doesn’t work,

```c
sprintf(asm_buffer,"jsr %s","bar");
asm(asm_buffer);
```
evokes the error *syntax error at or near “asm_buffer”* from the 4.2 C compiler. You should try this on your particular C compiler.

### 4.2 XINU System Calls

The supported system calls are:

- `xsend()` send a message to another process
- `xreceive()` wait for a message and return it
- `xrecvclr()` clear messages, returning waiting message (if any)
- `xresume()` unsuspend a process, making it ready
- `xsuspend()` suspend a process, placing it in hibernation
- `xkill()` kill a process and remove it from the system
- `xcreate()` create a process to start running a new procedure
- `xgetpid()` get the process id of currently executing process
- `xgetprio()` return the scheduling priority of a given process
- `xchprio()` change the scheduling priority of a process
- `xwait()` make current process wait on a semaphore
- `xsignal()` signal a semaphore, releasing one waiting process
- `xscreate()` create and initialize a semaphore, returning its id
- `xsddelete()` delete a semaphore by releasing its table entry
- `xsleep()` put a process to sleep for this many seconds
- `xmsleep()` put a process to sleep for this many milliseconds

For complete documentation on the system calls, see Comer’s text. Most of the supported system calls function exactly as described in the book. The only changes were to provide a millisecond timer rather than a tenth of a second timer, and all XINU system calls are prefaced with ‘x’ (for XINU) to avoid clashes with UNIX calls.

All internal procedures and variables that are global have been prefaced with an underscore to avoid conflicting with application names. For example, `_resched_`.

The system is configurable and can be recompiled without any combination of the following optional services:

- realtime clock
- semaphores
- messages

These and other typical configuration changes are isolated in `_conf.h_`.

The smallest 4.2 XINU system comes with only 5 system calls:

- `xcreate()`
- `xresume()`
- `xsuspend()`
- `xkill()`
- `xgetpid()`

and is actually quite useful.
Other Minor Differences Between Comer's XINU and 4.2 XINU

Comer's XINU is based on the LSI 11/2. 4.2 XINU is based on 4.2BSD UNIX. The source is almost entirely in C, and makes few assumptions about the underlying machine. Much of the code ports without changes. Besides the differences mentioned elsewhere in this paper, the other primary differences are the size of the registers and interrupt handling.

The LSI is 16-bit while 4.2 is char *. Occasional assumptions are necessarily made, unfortunately.

Interrupts in the UNIX appear as software signals. Thus, disabling interrupts is done with the 4.2 signal support routines.

If you intend to write your own system calls, you must allocate an int to store the old interrupt mask rather than a char. For example, disable(ps) is used to disable interrupts while storing the old processor status in ps. Comer's definition of disable is:

```c
/* disable interrupts - LSI 11/2 */
#define disable(ps) asm(”mfps "ps”); asm(”mtps $0340”)
```

while for 4.2 XINU, it is

```c
/* disable interrupts - 4.2BSD software interval timer */
#define disable(oldmask) oldmask = sigblock(1<<((SIGALRM-1))
```

Here, only the quantum timer interrupt is blocked. You may find that other signals should be blocked, however not all should (e.g. SIGTSTP should probably not be trapped).

Using UNIX System Calls From XINU

All UNIX system calls and many library calls should be made with some thought as to their consequences. exec, for example, will completely overlay the entire XINU application and system. Where functionality is duplicated by UNIX, it is generally better to use XINU's calls. For example, if a process wants to go to sleep, calling the UNIX sleep will stall the entire XINU system until the next interval timer occurs. If the process calls xsleep (the XINU equivalent) the current process is put on a queue waiting for the clock, and another XINU process is given control of the cpu.

We have not reimplemented I/O, since we were able to use UNIX I/O without change, however the default behavior of UNIX I/O is to block, leading to a similar problem as sleep (i.e. block until operation complete or until quantum expires).

Note that 4.2 system calls restart automatically upon interrupts. This allows programs to run without having to explicitly handle the quantum interrupt.

If this "blocking until quantum" behavior is undesirable, it is possible to use non-blocking I/O, either directly, or through a generalized interface leading to a second set of I/O system calls. Future work in this direction would be very useful.

Using UNIX Library Calls From XINU

Many UNIX library calls are nonreentrant, and do not protect themselves against this. This means that they use static variables which are common from one call to the next. If two processes make the same nonreentrant library call at the same time, it is likely that the routines will misbehave.

Using reentrant versions of libraries is the best solution. Alternatively, one can embed (or surround, if you don't have the source) semaphores in the library calls (provided by XINU), one per common data structure (such as _iob which is shared with all the routines that are part of the standard I/O library).

XINU system calls are protected against reentrancy problems by disabling timer interrupts. Since malloc and free are used for XINU memory management, you should disable timer interrupts or set up a semaphore for access to the malloc data structures when doing memory management.
Conclusion

We have ported an operating system to the UNIX environment by emulating the environment of a microprocessor in a single UNIX process. We now have a tool that is capable of simulating any set of cooperating real-time processes.

The applications have the ability to access all the power of UNIX, simply because the emulator runs as normal user code on a 4.2BSD system. Because all the XINU processes run in one UNIX process, it is especially easy to debug multiple programs with one debugger.

Perhaps the nicest benefit of this work, has been the ability to write processes that efficiently share data structures, at the expense of using distinct global names. This has long been a missing feature of UNIX.

References


tar vs. cpio

The following memorandum was delivered at the June meeting of the P1003 committee in Seattle by John S. Quartersman, USENIX Institutional Representative to the Committee. I feel its content is of great importance to the membership and have reproduced it here with John's consent. The final note is a consequence of the June meeting. - PHS

In both the Trial Use Standard and the current Draft 10, POSIX §10.1 describes a data interchange format based on the tar program. That section has appeared in every draft of IEEE 1003.1 in some form and has always been based on tar format. The P1003.1 Working Group has recently received two related proposals regarding that section: one to add cpio format (including old-style, non-ASCII (non c option) format); [N.048 Lorraine C. Kevra] [V11N14] [V11N25 Eric S. Raymond] the other to replace the existing tar-based format with cpio format. [N.043 X/OPEN] [V11N13] Some clarifications were received to the former. [N.064 Dominic Dunlop] [V11N15] It was also proposed verbally in the latest Working Group meeting to drop §10.1 altogether and let P1003.2 handle the issue. [V11N08] [V11N11] [V11N09 Guy Harris] [V11N12 Doug Gwyn]

The present note is a response to those proposals. Much of the detail in it is derived from articles posted in the USENET newsgroup...
There are a number of problems with both 
cpio formats. First, those related to the non-
ASCII format:

1. Numerous parameters, including inode
   numbers, mode bits, and user and group IDs,
   are kept in two-byte binary integers. This has
   historically produced serious byte-order
   problems when data is moved among systems
   with different byte orders. [V11N09 Guy
   Harris]

2. The byte-swapping and word-swapping
   options to the cpio program are inadequate
   patches; with an ASCII format the problem
   would not be present. The options are not
   consistent across versions of the program: in
   System III, data blocks and file names are byte
   swapped; in System V, only data blocks are
   byte swapped. [V11N09 Guy Harris]
   [V11N47 Andrew Tannenbaum]

3. The two-byte integer format limits the
   range of inode numbers to 0..65535. Many
   current file systems are bigger than that.
   [V11N37 Paul Eggert] [V11N39 Henry
   Spencer]

Non-ASCII cpio format is clearly not port-
able and should not even be considered for
standardization. [V11N12 Doug Gwyn]

There are several problems that occur even
with the ASCII cpio format:

1. Many implementations of cpio only look
   at the lower 16 (or even 15) bits of the inode
   number, even in ASCII format. [V11N39
   Henry Spencer] This is because the variable
   that is used to contain the value is declared to
   be unsigned short, just as in binary format.
   Thus, even though ASCII cpio format only
   constrains this number to the range 0..262143,
   the format is still less than portable. [V11N37
   Paul Eggert]

2. The proposed cpio ASCII format as specified,
   [N.048 Lorraine C. Kevra] [V11N14]
   is not portable because the proposal assumes
   that sizeof(int) == sizeof(long). [N.064
   Dominic Dunlop] [V11N15]

3. The file type is written in a numerical
   format, making it UNIX specific rather than
   POSIX specific, since POSIX (and tar) specifies
   symbolic, rather than numerical, values for file
   types. [V11N09 Guy Harris]

4. Hard links are not handled well, since
   cpio format does not directly record that two
   files are linked. If two files that are linked are
   written in cpio format, two copies will be
   written. The cpio program detects duplicate
   files by matching pairs of (h.dev, h.ino) and
   producing links, but that is done after the fact.
   [V11N09 Guy Harris] [V11N45 Guy Harris]
   [V11N54 Ian Donaldson] (There is a program,
   afio, that handles cpio format more efficiently
   in this and other cases than the licensed ver-
   sions of the program.) [V11N21 Chuck
   Forsberg]

5. Symbolic links are not handled at all, and
   no type value is reserved for them. This
   makes cpio useless on a large class of historical
   implementations (those based on 4.2BSD or its
   file system) for one of the main purposes of
   POSIX §10.1: archiving files for later retrieval
   and use on the same system. Although it is
   possible to extend cpio to handle symbolic
   links, and at least one vendor has done this,
   [V11N45 Guy Harris] the format proposed to
   P1003.1 is the format in the SVID, and does
   not handle symbolic links.

6. The cpio format is less common than tar
   format: there are few historical implementa-
   tions from Version 7 on that do not have tar;
   there are many that do not have cpio.
   [V11N09 Guy Harris] [V11N10 Charles
   Hedrick] [V11N24 Jim Cottrell] It is true that
   cpio (non-ASCII format) was invented before
   tar, [V11N22 Joseph S. D. Yao] apparently in
   PWB System 1.0. [V11N26 Joseph S. D.
   Yao] The cpio program was first available out-
   side AT&T with PWB/UNIX 1.0, [V11N45
   Guy Harris] [V11N63 Joseph S. D. Yao] and
   later with System III. However, in the
   interim, Version 7, which did not include cpio
   [V11N53 Bill Jones] [V11N62 Guy Harris] but
did include tar, became the most influential system. There was a V7 addendum tape, but it also did not include cpio (according to its README file); [V11N65 Rick Adams] the addendum tape was in tar format. Also, it appears that the cpio format of PWB was not the same as that of System III. [V11N39 Henry Spencer] And System III and all releases of System V include tar. [V11N26 Joseph S. D. Yao] [V11N63 Joseph S. D. Yao] [V11N45 Guy Harris] [V11N47 Andrew Tannenbaum]

7. It is very late in the process to propose that P1003.1 adopt cpio format now, especially considering that it was originally proposed to and rejected by the /usr/group committee before P1003.1 was even formed. [V11N39 Henry Spencer]

**Advantages of cpio format include:**

1. Both X/OPEN [N.043 X/OPEN] [V11N13] and the SVID [N.048 Lorraine C. Kevra] [V11N14] use it, although evidently defined somewhat differently. [N.064 Dominic Dunlop] [V11N15]

2. Archives made in cpio format are often smaller than ones in tar format. [V11N44 Mark Horton] But this is only because of the headers, and thus the effect diminishes with larger files.

3. On a local (non-networked) system, cpio is more efficient at copying directory trees than tar. [V11N46 Steve Blasingame] However, this is really an implementation issue.

4. The format specified in §10.1 is upward-compatible with tar format. Old tar archives can be extracted by a program that implements §10.1. Archives using some of the extensions of §10.1 can be extracted with old (Version 7) tar programs, although symbolic links will not be extracted and contiguous files will not be handled properly (cpio does not handle these capabilities at all). Files with very long names will not be handled properly (cpio does no better at this). All tar implementations are compatible to this extent. [V11N17 John Gilmore]

5. The /usr/group working group and P1003.1 have already done the work [P.061] [M.019 5.1.121 Pg.13] [RFC.003 #121] [P.038] [P.006] required to add optional extensions (such as symbolic links, long file names, [V11N49 Jerry Schwarz] [V11N50 Michael MacDonald] and contiguous files) that are needed on many historical implementations and that cpio format lacks.

6. The format is extensible for future facilities. [V11N39 Henry Spencer]

7. There is a public domain implementation of the format of §10.1. That implementation provided feedback which led to improvements in the current specification, and has been in use for years in transferring data with licensed tar implementations. [V11N17 John Gilmore]

8. Many people prefer the user interface of the cpio program to that of the tar program, because the former can accept a list of pathnames to archive on standard input while the latter takes them as arguments, limiting the length of the list. [V11N34 Andrew Tannenbaum] However, the above-mentioned public domain implementation of tar accepts pathnames on standard input, [V11N17 John Gilmore] [V11N19 Jim Cottrell] and at least one vendor sells a version of tar that can do this. [V11N48 Michael Gersten] Diffs to standard tar to add an option to accept pathnames on standard input when creating an archive have also been posted to USENET. [V11N36 John Gilmore] The user interface is, in any case, irrelevant to P1003.1. [V11N39 Henry Spencer] [V11N40 Rahul Dhesi]

**Disadvantages of tar format:**

1. If an attempt is made to extract only the second of a pair of hard linked files the tar program will attempt to link the second file to the nonexistent first file, and nothing will be...
extracted. Although a sufficiently clever implementation could avoid this, the problem can be considered to be in the archive format. [V11N66 Kenneth Almquist]

There are some problems that neither tar nor cpio handles well.

1. File names still longer than the length of PATH_MAX (at least 255) [V11N50 Michael MacDonald] that the POSIX format allows (and than the 128 that cpio permits or than the 100 that historical tar allows) would be preferable, although the POSIX limit is useful for most cases. [V11N54 Ian Donaldson]

2. An option to prevent crossing mount points would be useful for backups. [V11N19 Jim Cottrell] [V11N22 Joseph S. D. Yao] However, this appears to be more of an implementation issue than a format issue, [V11N28 Dave Brower] [V11N32 Joseph S. D. Yao] especially considering that there are options to find in 4.2BSD, [V11N24 Jim Cottrell] SunOS 3.2, [V11N36 John Gilmore] and System V Release 3.0 [V11N35 Mike Akre] that take care of this.

3. The default block size in many tar implementations is too large for some tape controllers to read [V11N27 Rob Lake] (the 3B20 has this problem). This is not a problem with the interchange format, however.

There is nothing that the proposed cpio can handle that the tar-based format already in POSIX §10.1 cannot handle; in fact, the former is less capable. If cpio format were augmented to handle missing capabilities, it would be subject to the same objections now aimed at the format given in §10.1: that it was not identical with an existing format.

There is no advantage in replacing the current tar-based format of §10.1 with cpio format. There is also no advantage in adding cpio format, because two standards are not as good as a single standard.

Some have recommended removing §10.1 from POSIX altogether, [V11N12 Doug Gwyn] perhaps with a recommendation for P1003.2 to pick up the idea. [V11N09 Guy Harris] While I believe that that would be preferable to adding cpio format, whether or not tar format remains, I recommend leaving §10.1 as it is, because:

- The inclusion of an archive/interchange file format is in agreement with the purpose of POSIX to promote portability of application programs across interface implementations. Some format will be used. It is to the advantage of the users of the standard for there to be a standard format.

- The de facto standard is tar format. The current §10.1 standardizes that, and provides upward-compatible extensions in areas that were previously lacking.

The Archive/Interchange File Format should be left as it is.

Thank you,
John S. Quarterman
Institutional Representative from USENIX

At its June meeting in Seattle, the P1003 committee decided to put off a decision on formatting until its September meeting. In the interim,

- the present cpio format is included in the next draft of the standard, preceded by a note saying that the Working Group must decide which (none, either, or both) will be in the standard, and that a revised proposal is forthcoming.

The options appear to be the obvious:

1. Leave the issue to P1003.2 and remove section 10 from POSIX.
2. Include only ustar format in POSIX.
3. Include only extended cpio format in POSIX.
4. Include both ustar and extended cpio formats as options in POSIX.
5. Require both ustar and extended cpio formats in POSIX.

The next issue of ;login: will continue this discussion. There will be a POSIX implementors workshop in October, see page 3 for the Call for Papers. – PHS
How To Write a UNIX Daemon

Dave Lennert
Hewlett-Packard Company
hplabs!hpda!davel

ABSTRACT

On UNIX systems users can easily write daemon programs that perform repetitive tasks in an unnoticed way. However, because daemon programs typically run outside a login session context and because most programmers are unfamiliar with designing a program to run outside this context, there are many subtle pitfalls that can prevent a daemon from being coded correctly. Further, the incompatibilities between various major UNIX variants compound these pitfalls. This paper discusses these pitfalls and how to avoid them.

Daemon: runs around in the shadows (background) doing devilish deeds.

found in some daemon source code

Introduction

A daemon is a program which performs periodic tasks in such a manner that it is normally unnoticed by users.

Some daemons run constantly, waiting for a significant event. Examples include init which respawns login sessions (gettys) as they end, cron which launches programs at specified times, and sendmail which listens on a socket for incoming mail messages.

Other daemons are launched periodically and terminate after completing one execution of their task. Such daemons include the uucp file transfer utility, uucico, which can be launched as a login shell when a remote machine logs in, calendar which is launched nightly by cron to examine users' calendars and mail them notification of upcoming events, and various mail handling utilities which allow the user's shell to continue while the collected mail message is delivered asynchronously.

Daemon programs are very easy to write in the UNIX environment. They can be written by casual users and launched periodically via the at command or, on System V, by a user's personal crontab file, or perhaps at each login via csh's .login command file. System administrators write daemons whenever they recognize a particular administrative task is becoming routine enough to be handled automatically.

However, daemon programs appear easier to write correctly than they really are. This is because there are many quirks and side effects of UNIX which are automatically taken care of in a login session context but not in a detached, daemon program. The init, getty, login, and shell programs oversee such functions as setting up user ID's, establishing process groups, allocating controlling terminals, and managing job control.

If a daemon process is launched outside a login session (e.g., via /etc/rc or a similar function during system startup) then it needs to manage these functions itself explicitly. If a daemon process is launched from within a login session (e.g., as a background command from a login shell) then it needs to undo much of what the login process sequence has done. In order to code a daemon robustly, both concerns must be addressed.

This paper discusses these concerns and the methods for addressing them. Note that all the example coding fragments lack necessary error condition checking or handling; such handling should, of course, be added to any real daemon.
Programming Rules

The following is a set of programming rules which avoid several subtle pitfalls. A discussion of each pitfall is also given along with the rule.

Make immune to background job control write checks.

On systems which support 4.2BSD style job control, daemons which attempt I/O to their controlling terminal will stop if they were launched from csh in the background (with &). The real way around this is to disassociate yourself from your controlling terminal (see below). In some cases though, the daemon will want to perform some setup checks and output error messages before it loses its controlling terminal.

There is no way to allow a background process to read from its controlling tty. However, output can be reliably performed if the calling process ignores the SIGTTOU signal, as in:

```c
#ifdef SIGTTOU
    signal(SIGTTOU, SIG_IGN);
#endif
```

For safety, it is probably a good idea to ignore the other stop signals as well, as in:

```c
#ifdef SIGTTIN
    signal(SIGTTIN, SIG_IGN);
#endif
#ifdef SIGTSTP
    signal(SIGTSTP, SIG_IGN);
#endif
```

Ignoring SIGTTIN also has the side effect of causing all background attempts to read from the controlling terminal to fail immediately and return the EIO error.

Close all open file descriptors, especially stdin, stdout, stderr.

Do not leave stray file descriptors open. More importantly, if any of the file descriptors are terminal devices then they must be closed to allow proper reset of the terminal state during logout (see below). The typical code sequence is:

```c
for (fd = 0; fd < _NFILE; fd++)
    close(fd); /* close all file descriptors */
```

Disassociate from your process group and controlling terminal.

Daemons launched during a login session inherit both the controlling terminal and the process group of that session (or, in the case of job control, of that job within the session).

As long as the daemon is still in the process group associated with a controlling terminal it is subject to terminal-generated signals (such as SIGINT or SIGHUP). As long as the daemon still has a controlling terminal it is subject to job control terminal I/O restrictions on systems which support job control.

Further, while the daemon remains in the original process group in which it started, it is subject to any signals sent to that process group by another program via kill(2).

One way to prevent the daemon from receiving these “unintended” signals is simply to ignore all signals. However, this means that the signals cannot be used by the daemon for other purposes (such as rudimentary interprocess communication). Also, this approach is insufficient because there are some signals which a process cannot ignore (for example, SIGKILL or SIGSTOP).

A better approach is for the daemon to disassociate itself from both the controlling terminal and from the process group which it inherited. On 4.2BSD systems, the former can be performed via the TIOCNOTTY ioctl(2) and the latter via setpgid(2). Under AT&T UNIX, setpgid(2) performs both functions.
However, (under AT&T UNIX) in order for `setpgid(2)` to have its desired effect, this must be the first time the process has called `setpgid(2)`; that is, the process must not already be a process group leader. (A process group leader is a process whose process group ID is equal to its process ID.) Since a program has no control over the process which `exec(2)'d` it, it must `fork(2)` to ensure that it is not already a process group leader before calling `setpgid(2)`. (This is especially important if the daemon is launched from a `csh` which supports job control since `csh` automatically makes its children process group leaders. But this also happens, for example, when an imprudent user launches a daemon from a login shell via the `exec` command.)

In order to prevent locking up a user's terminal when a daemon is started (i.e., without `&`), the daemon usually `fork(2)'s` anyway and runs in the child while the parent immediately `exit(2)'s` without waiting for the child. This causes the shell to believe that the daemon has terminated.

A typical code sequence would be:

```c
if (fork() != 0)
    exit(0); /* parent */
/* child */
#ifdef BSD
    setpgid(0, getpid()); /* change process group */
    if ((fd = open("/dev/tty", O_RDWR)) >= 0) {
        ioctl(fd, TIOCNOTTY, 0); /* lose controlling terminal */
        close(fd);
    }
#else /* AT&T */
    setpgid(); /* lose controlling terminal & change process group */
#endif
```

*Do not reacquire a controlling terminal.*

Once the daemon is a process group leader without a controlling terminal (having called `setpgid(2)` as described above) it is now potentially capable of reacquiring a controlling terminal. If it does, other processes (for example, logins) will not be able to acquire the terminal correctly as their controlling terminal.

(Interestingly, this problem does not exist under 4.2BSD. Unlike AT&T UNIX, where a terminal can only be acquired as a controlling terminal if it is not already a controlling terminal, 4.2BSD allows a process to join an already allocated controlling terminal and its process group. Basically, the process merges with the already established process group.)

The symptoms of this problem are somewhat subtle. Since `getty` and `login` are not able to acquire a controlling terminal, the special file, `/dev/tty`, cannot be successfully opened. Because of this, the `getpass(3)` routine, used by `login` to obtain the user's password, fails without ever printing the `Password:` prompt. All login attempts for accounts with passwords silently fail without ever prompting for a password. Login attempts for accounts without passwords succeed (because `getpass(3)` is never called), however the login shell does not have a controlling terminal. Terminal input and output still succeeds (via `stdin`, `stdout`, and `stderr`), but any keyboard signals are not sent to the processes spawned during this login session. Instead the signals are sent to the process which acquired this terminal as its controlling terminal (the daemon) and its descendants.

For this reason the daemon program must ensure that it does not re-acquire a controlling terminal.

On 4.2BSD systems, a new controlling terminal can only be acquired by a process with a process group ID of zero. After calling `setpgid(2)` to set its process group ID equal to its process ID, the daemon cannot re-acquire a controlling terminal.
Under AT&T UNIX, a new controlling terminal is acquired whenever a process group leader without a controlling terminal opens a terminal which is not already the controlling terminal for another process group. On such systems the daemon can reacquire a controlling terminal when opening, say, /dev/console, to perform logging or error reporting. Even if the daemon subsequently closes the terminal it still possesses it as a controlling terminal. There is no way to relinquish it since subsequent setpgrp(2) calls are ineffective. (setpgrp(2) has no effect if the caller is already a process group leader.) Therefore the acquisition must be prevented.

One simple way to prevent the acquisition of a new controlling terminal is to fork(2) yet another time after calling setpgrp(2). The daemon actually runs in this second child and the parent (the first child) immediately exit(2)'s. However, on AT&T UNIX when the parent (first child) terminates, the SIGHUP signal is sent to the child since the parent is a process group leader. Thus, the parent must ignore SIGHUP before fork(2)'ing the second child otherwise the child will be killed. (The ignored setting is inherited by the child.) The final side effect of the terminating (process group leader) parent is to set the process group of the child to zero. The daemon (second child) now has no controlling terminal, it is in a new (zero) process group which is immune to signals from the tty driver, and it cannot acquire a new controlling terminal since it is not a process group leader.

Thus the typical code sequence becomes:

```c
if (fork() != 0) exit(0); /* first parent */
/* first child */
setpgrp(); /* lose controlling terminal & change process group */
signal(SIGHUP, SIG_IGN); /* immune from pgrp leader death */
if (fork() != 0) /* become non-pgrp-leader */
exit(0); /* first child */
/* second child */
```

Do not “hold” open tty files.

Even after ensuring that the daemon will not re-acquire a controlling terminal when a terminal device is opened, there is a further concern:

Terminal state settings, such as BAUD rate and signal character definitions, are only reset to the default state when the last process having the terminal open finally closes it. Thus, if the daemon has a terminal open continuously, then the last close never happens and the terminal settings are not reset at logout.

Typical examples of terminal files held open by a daemon are stdin, stdout, stderr, and /dev/console.

It's probably best to log errors and status messages to a disk file rather than a terminal. However, when terminal logging is desired, the “correct” method is to hold the terminal open only long enough to perform a single logging transaction. Note that this logging transaction still represents a window of time during which a logout would not reset the terminal state.

4.2BSD systems have a further problem which makes this suggestion mandatory. Whenever a new login session is initiated via getty or similar routine, the vhangup(2) system call is invoked to prevent any existing process from continuing to access the login terminal. This results in read and write permissions being removed from any currently open file descriptor which references the login terminal; this affects all processes regardless of user ID. Therefore, daemons which access a terminal that is also used for regular login sessions, must reopen it whenever access is desired. If a file descriptor for such a terminal is continuously held open, it is very likely that vhangup(2) will quickly destroy its usefulness.

To determine if an unknown file descriptor is a terminal device use isatty(3).
Change current directory to ‘/’.

Each process has a current working directory. The kernel holds this directory file open during the life of the process. If a process has a current directory on a mounted file system, the file system is “in use” and cannot be dismounted by the administrator without finding and killing this process. (The hard part is finding the process!) Unless a process explicitly alters this via chdir(2), it inherits the current directory of its parent. When launched from an interactive shell, the current directory will be whatever the user has most recently selected via the cd command.

Because of this, daemons should adopt a current directory which is not located on a mounted file system (assuming that the daemon’s purpose allows this). The root file system, ‘/’, is the most reliable choice. The simple call is:

```c
chdir("/");
```

Reset the file mode creation mask.

A file mode creation mask, or umask, is associated with each process. It specifies how file permissions are to be restricted for each file created by the process. Like the current directory, it is inherited from the parent process and remains in effect until altered via umask(2). When launched from an interactive shell, the umask will be whatever the user has most recently selected via the umask(1) command.

A daemon should reset its umask to an appropriate value. The typical call would be:

```c
umask(0);
```

Other attributes to worry about.

The environment attributes discussed above are the primary ones to worry about, but the list is not exhaustive. Any attribute inherited across an exec(2) system call is of concern. Some other calls to be cautious of are the nice priority value (see nice(2)), the time left until an alarm clock signal (see alarm(2)), and, on 4.2BSD systems, the signal mask and set of pending signals (see sigvec(2)). However, these are less likely to be accidentally set “wrong.”

Interactions With init

The system initialization process, init, is responsible for directly or indirectly starting all processes on the system (with the exception of kernel processes such as the swapper or pageout process). On many versions of UNIX, init keeps track of all processes which it directly spawned and it can optionally respawn them if they die or it can kill them when changing to a new system run state (or level). Under AT&T UNIX, the /etc/inittab file specifies the programs init should spawn in which run levels and whether or not they should be automatically respawned when they die. (Note that this file differs in both format and capabilities between System III and System V.)

Historically, system daemon programs are launched by the /etc/rc shell script which init launches when moving the system from the single user run state to multi-user mode.

Some system administrators now prefer to launch daemons directly from init by placing the appropriate commands in /etc/inittab. They rely on init respawning the daemon should it inadvertently die and on init killing the daemon during system state changes.

Note that the respawning and terminating capabilities of init depend on the spawned program not terminating prematurely. The above programming rules, however, suggest that daemons should immediately fork(2) and have the original process exit(2). If launched from /etc/inittab, this procedure would cause init to believe that the daemon was no longer running and hence it would not terminate the daemon during state changes and would instead immediately relaunch the daemon (if automatic respawn were requested). This procedure thus defeats both the respawning and terminating capabilities provided by /etc/inittab.
What can be done to correct this? The only solution is to prevent the daemon from following the above procedure if it is launched from /etc/inittab.

One tempting approach is for the daemon to retrieve the process ID of its parent immediately using getppid(2) and, if it is init's process ID (1), skip the problematic code. However this is not perfectly reliable since any process whose original parent has terminated assumes init as its new parent. If a daemon is launched interactively from a user's shell, the shell might subsequently terminate before the daemon has executed the getppid(2) call. In short, there is a race condition. However, for practical purposes, this is a quick and easy way to solve the problem.

Another approach is to pass a command line flag to the daemon indicating whether the daemon is being launched from /etc/inittab or not. But this requires the user to set the flag correctly during both automatic and interactive invocations. A common error would be for a user to examine the launching command in /etc/inittab and then use it verbatim interactively.

Regardless of what approach is used, all the above mentioned pitfalls must still be recognized and avoided.

In the final analysis it seems that launching daemons from /etc/inittab, as opposed to /etc/rc, is unnecessary for the following reasons: (1) Relying on init to respawn a daemon is really masking a bug in the daemon; the daemon should never terminate by itself.† (2) Changing run states is an unusual occurrence on most systems; usually a system will move to multi-user mode and stay there.

Conclusions

Without following the above rules, strange symptoms which are hard to track down often result. Many times the errant daemon program is the last thing suspected (e.g., when terminal settings are not reset after logout). Other times it is the daemon that silently and mysteriously dies (e.g., when it attempts background I/O on a job control system). Frequently these symptoms only begin occurring well after the “debug period” for the daemon.

Example

The example below collects the above coding fragments into a single routine which a daemon calls to detach itself from the context of a login session.

/* Detach a daemon process from login session context. *
 * (This is a skeleton; add error condition checking and handling.) *
 */
#include <signal.h>
#include <stdio.h>
#ifdef BSD
#include <sys/file.h>
#include <sys/ioctl.h>
#endif

sessdetach()
{
    int fd; /* file descriptor */

    /* If launched by init (process 1), there's no need to detach. *
    * Note: this test is unreliable due to an unavoidable race

† One interesting counterexample is that some systems (e.g., ACSnet) allow system administrators to reset things by killing the appropriate daemons. It's nice to have the daemon start correctly (i.e., right arguments) by itself through the auspices of /etc/inittab. However, it's arguably better to have the daemon catch the termination signal and perform the reset without actually terminating; this may even be essential in the case of orderly shutdown of operations such as line printer spooling.
* condition if the process is orphaned.
*/
if (getppid() == 1)
goto out;
/* Ignore terminal stop signals */
#ifdef SIGTTOU
    signal(SIGTTOU, SIG_IGN);
#endif
#ifdef SIGTTIN
    signal(SIGTTIN, SIG_IGN);
#endif
#ifdef SIGTSTP
    signal(SIGTSTP, SIG_IGN);
#endif
    /* Allow parent shell to continue.
       * Ensure the process is not a process group leader.
    */
    if (fork() != 0)
        exit(0); /* parent */
/* child */
/* Disassociate from controlling terminal and process group.
   * Ensure the process can't reacquire a new controlling terminal.
   * This is done differently on BSD vs. AT&T:
   *   BSD won't assign a new controlling terminal
   *   because process group is non-zero.
   *   AT&T won't assign a new controlling terminal
   *   because process is not a process group leader.
   *   (Must not do a subsequent setpgrp())
   */
#ifdef BSD
    setpgrp(0, getpid()); /* change process group */
    if ((fd = open("/dev/tty", O_RDWR)) >= 0) {
        ioctl(fd, TIOCNOTTY, 0); /* lose controlling terminal */
        close(fd);
    }
#else /* AT&T */
    setpgrp(); /* lose controlling terminal & change process group */
    signal(SIGHUP, SIG_IGN); /* immune from pgrp leader death */
    if (fork() != 0) /* become non-pgrp-leader */
        exit(0); /* first child */
/* second child */
#endif
out:
    for (fd = 0; fd < _NFILE; fd++)
        close(fd); /* close all file descriptors */
    chdir("/"); /* move current directory off mounted filesystem */
    umask(0); /* clear any inherited file mode creation mask */
    return;
Call for Papers
EUUG Spring 1988 Conference

London
April 11-15, 1988

The UKUUG will host the Spring '88 European UNIX systems User Group Technical Conference at the Queen Elizabeth II Conference Center in London. Technical tutorials will be held on April 11 & 12, followed by the three day conference. A pre-conference registration packet will be issued in early December, 1987.

The EUUG invites abstracts from those wishing to present their work. All submitted papers will be refereed. They will be judged with respect to their quality, originality, and relevance. Suggested topics include, but are not limited to:

- Programming Environments and Tools
- Recent work in Standards and Portability
- Real-time UNIX
- Communications

Submissions from students are particularly encouraged under the EUUG Student Encouragement Scheme, details of which are available from the EUUG Secretariat.

Abstracts must be submitted by post to the EUUG Secretariat at the address below. All submissions will be acknowledged by return of post.

Those interested in offering tutorials should contact the EUUG Secretariat as soon as possible.

Important Dates

Abstract Deadline: October 30, 1987
Acceptance Notification Posted: November 20, 1987
Final Paper Received: January 15, 1987

The EUUG Secretariat may be contacted at:

EUUG
Owles Hall
Buntingford, Herts. SG9 9PL
United Kingdom

Phone: (+44) 763 73039
Fax: (+44) 763 73255 (G2)
Book Reviews

The Design of the UNIX Operating System
by Maurice J. Bach

Reviewed by Marc D. Donner
IBM Thomas J. Watson Research Center
ucbvax!ibm!donner

There are four potential audiences for this book. The first potential audience is a class of students in an operating systems course. The second potential audience is UNIX system hackers interested in deepening their understanding of the internals of UNIX. The third potential audience is application programmers in the UNIX world who think a better understanding of how the system works will help them do a better job (it will). The fourth potential audience is everyone with a prurient interest in finding out what goes on under the covers, with no other goal than improving his understanding.

The version of UNIX that Bach focuses on in this book is System V Release 2, but he also discusses some of the major Berkeley improvements, though not the file system.

As an operating systems text, I give this book a B. It earns a solid A for the second and third audiences and it earns a stellar A+ for the fourth audience. It suffers from a scattering of minor errors, both technical and typographical, but with one exception they are insignificant. The rest of this review consists of an evaluation of the book’s merits for each of the potential audiences, a summary of the contents, and a brief precis of the significant errors.

As an Operating Systems Textbook

In many ways this book is attractive as a text for an operating systems course. It is remarkably well written, the implementation of UNIX internals is clearly explained, and the exercises are generally well designed. Its weakness as a text is that it is very UNIX-centric. In many places the exposition focuses on how it is done in UNIX-as-it-is rather than on what the alternatives are in the general operating system design context. The book is not terribly strong on historical perspective, nor on citation to the relevant non-UNIX literature. This is a particularly dangerous blind spot for a text, since it can give impressionable students a “there is one way and Ken and Dennis are the prophets” view of operating system technology. This book might be an excellent adjunct to an advanced operating system course, with general principles taken from some other text or, better yet, from a collection of the important papers in the field.

As a Text for UNIX System Programmers

This book is the shortest path to a solid understanding of the important issues in the UNIX kernel that I can imagine. Its completeness is amazing, something that is difficult to appreciate without spending several hours reading and referring to the book. The explanations of the various kernel functions, which Bach reduces to pseudo-code and calls “algorithms,” are clear and complete. There is good attention to detail, including careful analysis of race conditions that motivate various details of the implementation.

The book is an excellent item to have by your side when digging into some ugly piece of kernel code. It has helped in the deciphering of more than one piece of typically impenetrable and inscrutable source.

As a Text for UNIX Application Programmers

The understanding of the underlying design and implementation of the UNIX kernel will help any application programmer make better design decisions. This book doesn’t give any instruction in the writing of application programs, but it does demonstrate the working of many kernel calls, with excellent exposition of their
properties. Eccentric behavior is exposed and explained, something that the manual pages are notorious for not doing.

Without descending unbearably into horrible detail, the book manages to illustrate the functioning of most of the interesting facilities. The discussion of signals in section 7.2 is excellent. It cleared up a number of problems in my own understanding of what was going on with signals.

*As an Expose of the Internals for UNIX Lovers*

This is where the book shines best. It is so comprehensive and generally so clear that it is a delight to read. The reader really should examine all the algorithms carefully, and it helps to work the examples to get the most from the book. There is a wealth of sample programs that highlight various odd or interesting behaviors. Most are worth typing in and executing, though some of the most interesting require superuser privileges.

The section that explains how demand paging works gives a good explanation of how to do it on a machine without a reference bit, tactfully refraining from any comment on machines with such a lack.

**Chapter Summary**

Chapter 1 is an introduction, with some history and philosophy. Chapter 2 gives an overview of the structure of the kernel, explaining the major sections and the division of responsibilities among the file system, the process control system, the I/O system, and the other components. Chapter 3 examines the buffer cache in detail. This chapter contains the only serious technical error in the book. The algorithm getblk has an error in it as presented in figure 3.4. The problem is that the original C code has a slimy interlocked loop, but the translation presented in the book has lost the function by simplifying it incorrectly. The text on page 48 describes what should happen correctly and a minor change to the algorithm will make it work, but it took two of us quite a while, plus a visit to our source code, to verify the error.

Chapter 4 contains more than you ever thought possible about the representation of files. The only topic that I can think of that it neglects is recent improvements that result in better clustering of blocks in the file system. Chapter 5 explores the file system calls, explaining the functioning of each one and relating it to the underlying data structures described in the previous chapter.

Chapter 6 begins the second half of the book with a detailed description of the structure and composition of processes. Process execution state, context, and memory are covered in detail. This is probably the trickiest chapter of the book. Chapter 7 covers process control, with fork, exec, pipe, dup, and other system calls being covered in detail. Signals are beautifully covered in this chapter as well. The exercises for this chapter are particularly numerous and challenging. Chapter 8 talks about process scheduling, with the time-related system calls thrown in for good measure.

Chapter 9 discusses memory management, both swapping and paging, in reasonable detail. There is a misleading statement in the introduction of the chapter that might cause a naive reader to think that 4.0 BSD was the first implementation ever of a demand paging policy, when it was really just the first UNIX system with demand paging. Chapter 10 talks about the I/O subsystem, with an explanation of the architecture of device drivers and some discussion of various device types. Chapter 11 is entitled “Interprocess Communication” and discusses both System V IPC and BSD Sockets. Chapter 12 discusses some of the issues involved in building a multiprocessor UNIX, while Chapter 13 talks about distributed versions.

There is an appendix containing a summary of all the system calls, with brief descriptions of what each does. The bibliography is fairly extensive, though it is focussed primarily on UNIX. The index is good but not great.

**Important Errors in Brief**

Exercise 2 on page 37 has the names of bp and bp1 swapped from the names used in figure 2.7. Figure 3.4 on page 44 has a serious error, already discussed above in the section on Chapter 3. Figure 5.20 and the text describing it do not agree. The hyphen breaking the word “Superusers” on page 121 extends into the margin of the page. The description on page 205 of figure 7.9 refers to an address as hexadecimal ee, but it should say f9. In figure 7.34 the first two occurrences of the variable srcfile should be replaced by argv[1] and argv[2] respectively. The swapper algorithm described
on page 281 is not quite right for the revised algorithm.

There are several stylistic and presentation problems as well. The table of contents is presented double spaced and in ALL CAPITALS, thus making it difficult to browse. In addition, it seems to me that there is really a structure over the chapter structure that should be exhibited by breaking the book into parts, with chapters 1 and 2 in the first part, chapters 3 through 5 in the second part, chapters 6 through 8 in the third part, chapters 9 and 10 in the fourth part, and chapters 11 through 13 in the last part. The exhibition of this structure in the table of contents would help illustrate the system structure that it mirrors. Another minor complaint: the publisher yielded to the modern temptation to slip junk in at the end of the book, thus inserting four blank pages and a page of advertising in between the index and the end paper. This is a recent disease among publishers that should be strongly discouraged, since it reduces the utility of the index of a book. There should be absolutely nothing between the index and the end paper. Nothing.

Summary

This is a truly outstanding book and it belongs in the library of every serious UNIX programmer. System hackers will find it invaluable, while application programmers will find it at least very interesting and informative. I think that it does not make an outstanding operating system textbook, but it would be an excellent adjunct to an operating system course.


by Samuel P. Harbison and Guy L. Steele, Jr.

Reviewed by Josiah C. Hoskins
joho@mcc.com

I think most everyone would agree that the definitive C reference book for the past decade has been The C Programming Language by Kernighan and Ritchie (1978). However, in 1984 the first edition of A C Reference Manual by Harbison and Steele appeared which provides an even more detailed and current description of the C language. I am not proposing that Harbison and Steele's reference manual is a replacement for K&R's The C Programming Language. I am stating that Harbison and Steele's book provides an excellent companion to K&R which reflects the fact that C is a dynamic and changing language and reference manuals must change as the C language matures. I attribute the popularity of this reference manual to four assets: 1) it has provided a current reference for the C programming language (filling the gap since 1978), 2) it reflects common interpretations of C in different machine environments, 3) it reflects the current ANSI standardization effort, 4) it provides one of the most complete, accessible, and well written references to the C programming language.

The second edition has varied little from its original intent to provide a detailed description of the current C language. Several changes have taken place that may encourage a C programmer to fork over the approximately $31.00 (hardcover, $24.95 paper) for the 2nd edition. However, the first change one notices is the smaller point size of the type face. This is not so nice as I program late and the smaller the print the less I see. Cheers, in section 1.3 Syntax Notation we find that that Backus-Naur Form (BNF) notation has been dropped! The first obvious substantive change is that there are many more references to suggestions of the Draft Proposed ANSI C Standard. The References now contain pointers to Draft Proposed ANSI C - included in chapter eleven.
Unlike some second editions it's obvious that the whole book was read for content and modified as needed. Of course, the homey character graphics figures remain and there still exists a wealth of example code. A welcome improvement is the expanded treatment of the character type (important in portability). As in the first edition chapter five on data types is worth the price of the book. Chapter six contains a new section entitled Representational Issues. It covers addressing structure and byte ordering, I mean where can you find that kind of information. The same precedence chart (note p.141 in both editions) is there again; give me p.49 in K&R. Some chapters like eight and nine hardly changed at all. Small changes were found in the stack example of chapter ten to better demonstrate program structure.

The rest of the second edition is quite different in both organization and somewhat different in content. Chapter eleven is a totally new chapter which summarizes Draft Proposed ANSI C by specifying how it differs from the C language as presented in H&S. The run-time library chapter has been completely reorganized and contains additional coverage of topics such as signals, time of day functions, and operations on arbitrary blocks of memory. Appendix B contains a revised syntax of the C language which includes both the traditional and Draft Proposed ANSI C. It is a nice presentation with both the Lexical rules and the Draft Proposed ANSI C clearly marked in the text.

As you should have guessed, I like the H&S C Reference Manual and have made good use of the first edition. If you already have the first edition, the second edition will not change the way you program significantly, however, it will provide you an up to date status of C. If you don't have a copy of A C Reference Manual I recommend getting one and you will find yourself writing more clear, correct, and more importantly portable C programs.

---

UUNET Progress Report

At the Winter '87 USENIX Conference in Washington, DC, the USENIX Association announced the funding of the UUNET project on an experimental basis. UUNET became operational in mid-May. As of July 1 UUNET has over 50 subscribers. This article introduces UUNET to the USENIX membership and provides a progress report.

UUNET is a non-profit communications service that provides access to USENET news, UUCP mail, and ARPANET mail. UUNET is the new experimental project of the USENIX Association with the unprecedented cooperation of DARPA.

For this experiment, DARPA has authorized the use of the Center for Seismic Studies personnel, resources, and communications facilities. This allows UUNET to house its host computer at a well-staffed and maintained computer center and to provide the high quality services necessary for this project. In addition, DARPA has authorized use of the ARPANET gateway at the Center on an experimental basis to test the feasibility of mail forwarding between ARPANET and non-ARPANET sites.

This is the first time a joint project like this has been initiated and the experiment will be carefully conducted to assure that all ARPANET and Center policies are followed. The technical results of the experiment will be presented to DARPA for their consideration of the long-term possibilities of continued interaction and to USENIX for their funding consideration.

The USENIX Association will provide financial and administrative support during the course of the experiment. In addition to news and mail, UUNET will provide access to various source archives, many standards, (including the Internet RFC's and comp.std.unix archives), and NIC service for the USENIX community.
There are no restrictions on what you may send nor on redistributing what you obtain from or through UUNET. UUNET is effectively a common carrier. Charges are calculated to recover costs.

UUNET exists as a communications relay. It is not used for any other project and is maintained 24 hours per day. The number of intermediate hops to members for news and mail is greatly reduced, thereby increasing the reliability. In the first month that UUNET was operational, there was no unscheduled downtime.

Subscribers can be directly connected to a backbone site and not have to depend on others to redistribute newsgroups. Subscribers may have a full newsfeed, a partial newsfeed, or none at all. (A full newsfeed costs about $175 per month in connect time.) UUNET always carries all newsgroups. This includes any new news categories that may appear other than those in the standard set.

UUNET makes available for UUCP access an extensive archive of publicly available UNIX software. This includes the latest GNU software, the latest Kermit distributions (for many CPU types, not just UNIX), all the Internet RFCs, the latest UUCP map information (updated daily from the master copy), access to the Simtel-20 archives, and the netlibd archives at Argonne (EISPACK, LINPACK, etc.).

Operationally, UUNET consists of a 10 processor Sequent Balance 21000 located at the Center for Seismic Studies at Arlington, VA. The system is connected to Tymnet via a high-speed leased line. It can handle 25 simultaneous uucico transfers and will be upgraded to match demand. It is administered by the same people who are currently administering seismo. Operations personnel are on site 24 hours/day Monday-Friday and someone is always on call on weekends. Currently, the UUNET machine is tightly coupled to seismo. This means that having a connection to UUNET is effectively having a connection to seismo, i.e., a well-connected news and mail relay.

To access the UUNET system from within the United States, subscribers dial a local phone number (from thousands of US cities) and connect to Tymnet. Subscribers are then connected to UUNET via the Tymnet X.25 public data network. International sites may access UUNET via direct host-to-host X.25 connection. No special hardware or software is required (other than the standard UUCP protocols). The connection to Tymnet is made with an ordinary modem (V.22bis / Bell 212 / Bell 103).

The cost of using UUNET is $3 per hour of connect time during off-peak times ($5 per hour from Hawaii). Off-peak times are 6:00 p.m. to 7:00 a.m. Monday-Friday and all day Saturday and Sunday. (The subscriber's time zone is used to determine peak or off-peak time, not necessarily the time zone in which the UUNET system is located.) UUNET accepts calls 24 hours a day. Access is available during peak rate time at substantially higher rates ($20-$32 per hour depending on location). There is a membership charge of $30 per month to cover administrative costs.

As previously mentioned, USENIX has funded UUNET for an experimental period. UUNET expects to offer these services at these prices by generating a large volume of traffic. If a large enough volume of traffic is seen by the end of the experiment, USENIX will spin off the UUNET experiment into an independent, non-profit organization that will continue the service with the same basis. If a large enough interest is not shown to allow UUNET to recover its operating costs, USENIX will regrettably have to discontinue funding.

For further information on UUNET, please contact:

Peter Salus
UUNET / USENIX
P.O. Box 2299
Berkeley, CA 94710
+1 415 528-8649
(seismo, uunet, ucbvax, cbosgd, ames, amdahl)!usenix!peter
The 1988 Election of the USENIX Board of Directors

The May/June of ;login: carried an article concerning the formation of a Nominating Committee for the 1988 Board elections. As required in the bylaws, this is the official notice of the constituency of the committee. The nominees will be announced at the 1988 Dallas Technical Conference, the first week of February.

The following have agreed to serve on the Committee:

Lew Law (chair) (usenix.harvard)!!law 617-495-2627
Bruce Borden hplabs!dana!bsb 415-960-1980
Tom Ferrin tef@cgl.ucsf.edu 415-476-1100
Mike O'Dell mo@seismo 703-893-3660
Peter Langston psl@bellcore 206-641-6106
Mike Tilson mike@hcrvax 416-922-1937

The Board is comprised of eight members: President, Vice-President, Treasurer, Secretary, and four Board members.

Some attributes which contribute to promoting the smooth functioning and continuing development of the Association are:

- a real interest in what the USENIX Association is doing,
- a commitment to improve and expand the Association activities,
- willingness and available time to commit to Association activities.

If you are interested, know of anyone who is or might be interested, or know of anyone who would be an asset to the USENIX Association as a Board member, and willing to serve, please contact me or any member of the Nominating Committee.

Further, the bylaws state that "Nominations ... may also be made by any five members. All nominations must bear the signature of at least five Voting Members."

Peter H. Salus
Executive Director

Call for Participation: USENIX C++ Workshop

Santa Fe, New Mexico
November, 1987

This two-day USENIX workshop will bring together users of the C++ language to share their experiences. The exact dates of the workshop will be announced on the net soon. We are currently leaning heavily toward Santa Fe, but that may change.

Bjarne Stroustrup, designer of the C++ language, has agreed to speak on topics guaranteed not to be in the C++ book, including multiple inheritance and other futures.

If you are interested in presenting either a full paper or a 10-minute discussion of your current work, please contact the Conference Chair as soon as possible. Suggested topics include, but are not limited to: Case Studies, Programming Support Environments, Teaching & Learning C++, Class Libraries, Debugging C++, and C++ Compiler implementations. Abstracts for full papers must be submitted by September 15, 1987, preferably electronically.

Conference Chair:
Keith Gorlen
Building 12A, Room 2017
National Institutes of Health
Bethesda, MD 20892
301-496-5363
usenix!nih-cs!keith
Summary of the Board of Directors’ Minutes
New Orleans, 26-27 March 1987

Attendance

Present at the meeting were: Stephen C. Johnson, Marshall Kirk McKusick, Alan G. Nemeth, John S. Quarterman, Deborah K. Scherrer, Waldo M. Wedel, David A. Yost—Directors; Rick Adams, UUNET; Eric Allman, Phoenix program chair; Ed Borkovsky, /usr/group director; Judith F. DesHamais, Conference Coordinator; John L. Donnelly, Tutorial and Exhibit Manager; Mike O’Dell, UUNET; Jeannie Patton, Wells Communications; Lizabeth Reilly, /usr/group Executive Director; Peter H. Salus, Executive Director.

Washington Conference Wrap-Up

Despite the snowstorm, it was generally agreed that the Washington, DC, conference had gone well. There had been 303 UniForum registrants who also attended the USENIX meeting and 63 USENIX registrants who had checked off the appropriate box on the UniForum registration. There were 1570 registrants for the USENIX conference and “about 1500” registrants for the /usr/group technical sessions and tutorials.

There was a good deal of discussion concerning the formatting of technical sessions and topical sessions. The question of four-day vs. three-day conferences was considered and it was decided that if DesHamais could make appropriate arrangements, the Dallas conference should be enlarged to four days [this has been effected].

Large Systems Workshop

It was reported that there were three dozen registrants for the forthcoming workshop in Philadelphia. The Workshop will have pre-prints, so no Proceedings volume is scheduled; Salus stated that a summary would appear in ;login:.

Phoenix meeting

Allman reported that there would be 41 papers offered in Phoenix in 14 sessions. Donnelly reported that there will be 8 tutorials each on Monday and Tuesday. There will be four “new” tutorials: Advanced System V, Graphics offered by Borden, X-windows, and NeWs. Yost stated that the FaceSaver on which he and Katz had been working would be operative in Phoenix.

Future meetings

The table at the end of this summary shows the status of planning for future semiannual meetings of the USENIX Association. Where the word None appears in a field, in means that no commitments have been made for this category for this meeting.

UUNET

Adams and O’Dell presented the UUNET business plan (following their January proposal) which was greeted with enthusiasm. After considerable discussion, Salus was asked to consult with the Association’s lawyer and accountant and to set aside $35,000 for the brief experimental period to cover excess expenses, even though this was larger than the predicted possible loss. Salus was to sign on behalf of the Association as guarantor wherever possible, rather than actually expending funds.

Standards

Nemeth and Quarterman detailed the history of the P1003 committee and the Association’s representation on it. Quarterman spoke of his roles and of the steps by which the trial use standard could become a full use standard. Reilly told /usr/group’s side of POSIX and said that /usr/group’s pamphlets had renewed interest in standards in the user community. The question of a Standards workshop in the autumn was brought up. The Board expressed warmth and Quarterman agreed to find a chair and fix upon a location.
Meeting with /usr/group

Following Washington, Nemeth had invited the /usr/group Board and Executive Director to the New Orleans meeting; Borkovsky and Reilly now invited the USENIX Board and Executive Director to a portion of /usr/group's next Board meeting on May 1 in Santa Clara, CA.

Women and Minorities

Quarterman and Scherrer brought up a number of proposals made by Liz Sommers concerning women and minorities in USENIX. There was considerable discussion, in the course of which it was noted that there were three distinct questions: the ethnic and sex representation in the industry, the representation in USENIX, and the phobia where “wizards” and “gurus” were concerned, which made the Association seem like an “in” group.

Salus was requested to get in touch with the United Negro College Fund.

It was decided to look into the following items:
- science fairs
- advertising what USENIX does
- using student help at meetings
- instructing program chairs to use more “new” folks
- problem BoFs.
- work-in-progress sessions

Future Meetings Status

<table>
<thead>
<tr>
<th>Date</th>
<th>Location</th>
<th>Hotel</th>
<th>Host</th>
<th>Program Chair</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/8-12/87</td>
<td>Phoenix</td>
<td>Hyatt</td>
<td>None</td>
<td>Eric Allman</td>
</tr>
<tr>
<td>2/8-12/88</td>
<td>Dallas</td>
<td>Registry</td>
<td>None</td>
<td>Rob Kolstad</td>
</tr>
<tr>
<td>6/20-24/88</td>
<td>San Francisco</td>
<td>Hilton</td>
<td>None</td>
<td>Sam Leffler</td>
</tr>
<tr>
<td>1/31-2/3/89</td>
<td>San Diego</td>
<td>Town &amp; Country</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>6/12-16/89</td>
<td>Baltimore</td>
<td>Hyatt</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>1/23-26/90</td>
<td>Wash. DC</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>6/11-15/90</td>
<td>Anaheim</td>
<td>Marriott</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>1/22-25/91</td>
<td>Dallas</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>6/10-14/91</td>
<td>Nashville</td>
<td>Opryland</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

Summary of the Board of Directors’ Meeting
Phoenix, June 7, 8, 10, 1987

Attendance

Present at the meeting were: Stephen C. Johnson, Rob Kolstad, Marshall Kirk McKusick, Alan G. Nemeth (President), John S. Quarterman, Deborah K. Scherrer, Waldo M. Wedel – Directors [David A. Yost was working on the FaceSaver project during most of the meeting]; Rick Adams, UUNET; Eric Allman, Phoenix program chair; Judith F. DesHarnais, Conference Coordinator; John L. Donnelly, Tutorial and Exhibit Manager; Betty Madden, Office Manager; Mike O'Dell, UUNET; Jeannie Patton, Wells Communications; Peter H. Salus, Executive Director.

Meetings

Rob Kolstad will be the Program Chair for the Dallas 1988 technical conference, which will be held concurrently with UniForum; Sam Leffler will be the Program Chair for the San Francisco, summer 1988 conference. Jim McGinniss has agreed to chair a POSIX Implementors Workshop.

There was a discussion of the reception of the past graphics workshops. Tom Duff will chair the October 1987 Graphics Workshop. Yost is working on a C++ Workshop. Salus and Marc Donner are meeting with IEEE on a possible joint Real Time Workshop in May 1988. The Philadelphia Systems Administrators Workshop was enthusiastically received by
the attendees. There was discussion of a second System Administrators Workshop as well as future Graphics workshops.

**UUNET**

After one month in service, UUNET has 50 actual subscribers. Furthermore, as a result of lengthy negotiations, Adams has been authorized by DARPA to make UUNET a DARPA gateway for an experimental period of three years.

The Board decided to extend the UUNET experimental period to October 31, 1987. Johnson, Quarterman, Scherrer, and Wedel will act as a Board subcommittee to oversee the experiment. Salus is to confer with the Association's lawyer concerning UUNET's independent, non-profit incorporation and other legal matters.

It was announced that a usenix.org domain had been set up and would be run by Adams and O'Dell. Further the comp.std.unix archives will be placed on the UUNET machine.

**net.sources**

Kolstad reported on the progress of collecting, unpacking, sorting, and canonicalizing net.sources and mod.sources and an assortment of other publically-available software. It appears the project will be complete within two months, and will include some 1200 software packages. The results will be made available via distribution tapes. There was a good deal of discussion concerning permissions. Wherever possible, authors will be asked for permission to redistribute materials which had been posted.

**Public Relations**

There was a lengthy discussion of the performance of Wells Communications on behalf of the Association. Wells has been retained as Public Relations consultants to the Association over the past year. It was agreed that /etc/motd (the daily newspaper at the Phoenix conference) was a success as was the Editorial Roundtable, set up by Wells to enable a limited number of media editors to confer with some of the USENIX Board. Other activities had been less successful. It was decided to reduce Wells Communications' activities in some areas, but to retain them in others which had shown benefits.

It was noted that Jobs' keynote and the FaceSaver also served as public relations vehicles for the Association.

**Membership Categories**

Scherrer presented her recommendations on membership categories, services, and fees for 1988. The addition of several new services, including distribution of the new technical journal, was proposed. Consideration of fees was postponed until the October Board meeting, when a more definite statement on the costs of the new services would be available.

There was a discussion of manual sales. It was decided that if the lawyers agree, individual members should be permitted to purchase manuals.

**Women's BOF**

There was a discussion of the Women in USENIX BOF and of the exchanges that had appeared in comp.org.usenix. There was support for the BOF and for the ideas and information to be gained from it. [The Executive Director and six of the eight Board members attended the BOF.]

**Journal**

There was a discussion of the proposed new technical quarterly including the nature of the editorial panel and the fact that submissions would be fully refereed. Salus presented proposals from several publishers who were interested in the USENIX quarterly. The Board authorized him to pursue several of these publishers and to thank the others for their interest.

**2.10BSD**

It was reported that the Computer Systems Research Group of UC Berkeley would like the USENIX Association to distribute 2.10BSD (formerly 2.9). This is the PDP 11 version of UNIX. It was decided that this might be possible through the normal USENIX tape distribution mechanism. Salus was asked to pursue this with the Berkeley Software Office.
Current Meeting

There was a discussion of the current meeting. It was felt that `/etc/motd` and the WIP sessions had gone well, as had some of the Public Relations activities. The great success had been the FaceSaver project, which had been widely acclaimed. Yost and Katz were asked to make the FaceSaver available to the Association in Dallas and in San Francisco next year.

Future Meetings

AUUG Winter Conference
August 27 & 28, 1987 New South Wales, Australia

The 1987 Australian UNIX-Systems Users Group Winter conference will be held on August 27 & 28 at the New South Wales Institute of Technology. For further information, contact: Greg Webb at (02) 218 9580 or gregw@nswitgould.oz, Tony McGrath (02) 218 9178 or tony@nswitgould.oz, or

ACSnet: auug@nswitgould.oz
ARPA: auug%nswitgould.oz@seismo.css.gov
uucp: seismo!munnari!nswitgould.oz!auug

or write to:
AUUG Winter Conference 1987
Computer Center
NSWIT
P.O. Box 123
Broadway, NSW 2007 Australia

4th Computer Graphics Workshop
Oct. 8 & 9, 1987, Cambridge, MA

For information, contact Tom Duff, the program chair, at research!tdu or (201) 582-6485.

POSIX Portability Workshop
Oct. 15 & 16, 1987, Berkeley, CA

For information, contact Jim McGinniss, at decvax!jmcg or (603) 884-5703.

USENIX 1988 Winter Conference and UniForum – Dallas

The USENIX 1988 Winter Conference will be held on February 8-12, 1988, at the Registry Hotel in Dallas, Texas. It will be concurrent with UniForum 1988, which will also be in Dallas. The Conference will feature tutorials and technical sessions.

USENIX 1988 Summer Conference and Exhibition – San Francisco

The USENIX 1988 Summer Conference and Exhibition will be held on June 20-24, 1988, at the Hilton Hotel in San Francisco, California. There will be a conference, tutorials, and vendor exhibits.

Long-term USENIX Conference Schedule

Feb 8-12 '88 Registry Hotel, Dallas
Jun 20-24 '88 Hilton Hotel, San Francisco
Jan 31-Feb 3 '89 Town & Country Inn, San Diego
Jun 12-16 '89 Hyatt Regency, Baltimore
Jan 23-26 '90 Washington, DC
Jun 11-15 '90 Marriott Hotel, Anaheim
Jan 22-25 '91 Dallas
Jun 10-14 '91 Opryland, Nashville
Publications Available

The following publications are available from the Association Office or the source indicated. Prices and overseas postage charges are per copy. California residents please add applicable sales tax. Payments must be enclosed with the order and must be in US dollars payable on a US bank.

Conference and Workshop Proceedings

Note the reduction in price of all USENIX conference proceedings more than a year old.

<table>
<thead>
<tr>
<th>Meeting</th>
<th>Location</th>
<th>Date</th>
<th>Price</th>
<th>Overseas Mail</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>USENIX</td>
<td>Phoenix</td>
<td>Summer '87</td>
<td>$20</td>
<td>$25</td>
<td>$5</td>
</tr>
<tr>
<td>USENIX</td>
<td>Wash. DC</td>
<td>Winter '87</td>
<td>$20</td>
<td>$25</td>
<td>$5</td>
</tr>
<tr>
<td>UniForum</td>
<td>Wash. DC</td>
<td>'87</td>
<td>$20</td>
<td>$20</td>
<td>/usr/group</td>
</tr>
<tr>
<td>Graphics Workshop III</td>
<td>Monterey</td>
<td>December '86</td>
<td>$10</td>
<td>$15</td>
<td>$5</td>
</tr>
<tr>
<td>USENIX</td>
<td>Atlanta</td>
<td>Summer '86</td>
<td>$10</td>
<td>$25</td>
<td>$5</td>
</tr>
<tr>
<td>USENIX</td>
<td>Denver</td>
<td>Winter '86</td>
<td>$10</td>
<td>$25</td>
<td>$5</td>
</tr>
<tr>
<td>UniForum</td>
<td>Anaheim</td>
<td>'86</td>
<td>$15</td>
<td>$20</td>
<td>/usr/group</td>
</tr>
<tr>
<td>Graphics Workshop II</td>
<td>Monterey</td>
<td>December '85</td>
<td>$3</td>
<td>$7</td>
<td>$5</td>
</tr>
<tr>
<td>USENIX</td>
<td>Dallas</td>
<td>Winter '85</td>
<td>$10</td>
<td>$25</td>
<td>$5</td>
</tr>
<tr>
<td>Graphics Workshop I</td>
<td>Monterey</td>
<td>December '84</td>
<td>$3</td>
<td>$7</td>
<td>$5</td>
</tr>
<tr>
<td>USENIX</td>
<td>Salt Lake</td>
<td>Summer '84</td>
<td>$10</td>
<td>$25</td>
<td>$5</td>
</tr>
<tr>
<td>UniForum</td>
<td>Wash. DC</td>
<td>'84</td>
<td>$ 5</td>
<td>$20</td>
<td>/usr/group</td>
</tr>
</tbody>
</table>

EUUG Publications

The following EUUG publications may be ordered from the USENIX Association office.

The EUUG Newsletter, which is published four times a year, is available for $4 per copy or $16 for a full-year subscription. The earliest issue available is Volume 3, Number 4 (Winter 1983).

The July 1983 edition of the EUUG Micros Catalog is available for $8 per copy.

EUUG

Publications

The following EUUG publications may be ordered from the USENIX Association office.

The EUUG Newsletter, which is published four times a year, is available for $4 per copy or $16 for a full-year subscription. The earliest issue available is Volume 3, Number 4 (Winter 1983).

The July 1983 edition of the EUUG Micros Catalog is available for $8 per copy.
4.3BSD UNIX Manuals

The USENIX Association is sponsoring production of the 4.3BSD UNIX Manuals for its Institutional and Supporting members. This article provides information on the contents, cost, and ordering of the manuals.

The 4.3BSD manual sets are significantly different from the 4.2BSD edition. Changes include many additional documents, better quality of reproductions, as well as a new and extensive index. All manuals are printed in a photo-reduced 6"x9" format with individually colored and labeled plastic “GBC” bindings. All documents and manual pages have been freshly typeset and all manuals have “bleed tabs” and page headers and numbers to aid in the location of individual documents and manual sections.

A new Master Index has been created. It contains cross-references to all documents and manual pages contained within the other six volumes. The index was prepared with the aid of an “intelligent” automated indexing program from Thinking Machines Corp. along with considerable human intervention from Mark Seiden. Key words, phrases and concepts are referenced by abbreviated document name and page number.

While two of the manual sets contain three separate volumes, you may only order complete sets.

The costs shown below do not include applicable taxes or handling and shipping from the publisher in New Jersey, which will depend on the quantity ordered and the distance shipped. Those charges will be billed by the publisher (Howard Press).

Manuals are available now. To order, return a completed “4.3BSD Manual Reproduction Authorization and Order Form” to the USENIX office along with a check or purchase order for the cost of the manuals. You must be a USENIX Association Institutional or Supporting member. Checks and purchase orders should be made out to Howard Press. Orders will be forwarded to the publisher after license verification has been completed, and the manuals will be shipped to you directly from the publisher.

<table>
<thead>
<tr>
<th>Manual</th>
<th>Cost*</th>
</tr>
</thead>
<tbody>
<tr>
<td>User’s Manual Set (3 volumes)</td>
<td>$25.00/set</td>
</tr>
<tr>
<td>User’s Reference Manual</td>
<td></td>
</tr>
<tr>
<td>User’s Supplementary Documents</td>
<td></td>
</tr>
<tr>
<td>Master Index</td>
<td></td>
</tr>
<tr>
<td>Programmer’s Manual Set (3 volumes)</td>
<td>$25.00/set</td>
</tr>
<tr>
<td>Programmer’s Supplementary Documents, Volume 1</td>
<td></td>
</tr>
<tr>
<td>Programmer’s Supplementary Documents, Volume 2</td>
<td></td>
</tr>
<tr>
<td>System Manager’s Manual (1 volume)</td>
<td>$10.00</td>
</tr>
</tbody>
</table>

* Not including postage and handling or applicable taxes.

4.2BSD Manuals are No Longer Available

† Tom Ferrin of the University of California at San Francisco, a former member of the Board of Directors of the USENIX Association, has overseen the production of the 4.2 and 4.3BSD manuals.
4.3BSD Manual Reproduction Authorization and Order Form

This page may be duplicated for use as an order form

USENIX Member No.: ______________
Purchase Order No.: ______________
Date: ______________

As the representative of a USENIX Association Institutional or Supporting Member in good standing, and as a bona fide license holder of both a 4.3BSD software license from the Regents of the University of California and a UNIX®/32V or System III or System V license or sublicense from AT&T and pursuant to the copyright notice as found on the rear of the cover page of the UNIX®/32V Programmer's Manual stating that

"Holders of a UNIX®/32V software license are permitted to copy this document, or any portion of it, as necessary for licensed use of the software, provided this copyright notice and statement of permission are included,"

I hereby appoint the USENIX Association as my agent, to act on my behalf to duplicate and provide me with such copies of the Berkeley 4.3BSD Manuals as I may request.

Signed (authorized Institutional representative): ____________________________
Institution: ____________________________

Ship to: ____________________________ Billing address, if different: ____________________________
Name: ____________________________ Name: ____________________________
______________________________ ________________________________
______________________________ ________________________________
______________________________ ________________________________

Phone: ____________________________ Phone: ____________________________

The prices below do not include shipping and handling charges or state or local taxes. All payments must be in US dollars drawn on a US bank.

4.3BSD User's Manual Set (3 vols.) _______ at $25.00 each = $___________
4.3BSD Programmer's Manual Set (3 vols.) _______ at $25.00 each = $___________
4.3BSD System Manager's Manual (1 vol.) _______ at $10.00 each = $___________
Total _______ $___________

[ ] Purchase order enclosed; invoice required.
(Purchase orders must be enclosed with this order form.)

[ ] Check enclosed for the manuals: $___________
(Howard Press will send an invoice for the shipping and handling charges and applicable taxes.)

Make your check or purchase order out to Howard Press and mail it with this order form to:

Howard Press
c/o USENIX Association
P.O. Box 2299
Berkeley, CA  94710

for office use: l.v.: _________ check no.: _________ amt. rec'd: _________ 37
Local User Groups

The USENIX Association will support local user groups in the following ways:

- Assisting the formation of a local user group by doing an initial mailing for the group. This mailing may consist of a list supplied by the group, or may be derived from the USENIX membership list for the geographical area involved. At least one member of the organizing group must be a current member of the USENIX Association. Membership in the group must be open to the public.

- Publishing information on local user groups in ;login; giving the name, address, phone number, net address, time and location of meetings, etc. Announcements of special events are welcome; send them to the editor at the USENIX office.

Please contact the USENIX office if you need assistance in either of the above matters. Our current list of local groups follows.

In the Atlanta area there is a group for people with interest in UNIX or UNIX-like systems, which meets on the first Monday of each month in White Hall, Emory University.

Atlanta UNIX Users Group
P.O. Box 12241
Atlanta, GA 30355-2241
Marc Merlin (404) 442-4772
Mark Landry (404) 365-8108

In the Boulder, Colorado area a group meets about every two months at different sites for informal discussions.

Front Range Users Group
USENIX Association Exhibit Office
Oak Bay Building
4750 Table Mesa Drive
Boulder, CO 80303
John L. Donnelly (303) 499-2600
(usenix,boulder)@johnnd

A UNIX user's group has formed in the Coral Springs, Florida, area. For information, contact:

S. Shaw McQuinn (305) 344-8686
8557 W. Sample Road
Coral Springs, FL 33065

Dallas/Fort Worth UNIX User's Group
Seny Systems, Inc.
5327 N. Central, #320
Dallas, TX 75205
Jim Hummel (214) 522-2324

In East Lansing, Michigan, Michigan USENIX meets approximately monthly. It is open to all interested persons, not only University affiliates. For meeting times and more information, contact:

John H. Lawitzke (517) 355-3769
Division of Engineering Research
364 Engineering Building
Michigan State University
East Lansing, MI 48824
ihnp4!msudoc|leeae!lawitzke

In Fresno, California, a Central California UNIX User's Group is now being formed to bring together UNIX users, programmers, and administrators. Initially the group will consist of an electronic mailing list to which questions, comments, answers, rumors, and tips will be posted. Communication will be by uucp.

Educational and governmental institutions contact:
Brent Auernheimer (209) 294-4373
Department of Computer Science
California State University
Fresno, CA 93740-0109
CSNET: brent@CSUFresno.edu
uucp: csufres!brent

Commercial institutions, contact:
Gordon Crumal (209) 435-4242
Harry J. Wilson & Company
Insurance Correspondents, Inc.
2350 W. Shaw Ave, Suite 110
Fresno, CA 93711
uucp: csufres!towerl!gordon
The Los Angeles UNIX Group (LAUG) meets on the third Thursday of each month in Redondo Beach, California. For additional information, please contact:

Drew Bullard (213) 535-1980
{ucbvax,ihnp4!trwrb!bullard
Marc Ries (213) 535-1980
{decvax,sdcrdf!trwrb!ries

In Minnesota a group meets on the first Wednesday of each month. For information, contact:

UNIX Users of Minnesota
Shane McCarron (612) 786-1496
ihnp4!meccts!ahby
ahby@mecc.com

In the northern New England area is a group that meets monthly at different sites. Contact one of the following for information:

Emily Bryant (603) 646-2999
Kiewit Computation Center
Dartmouth College
Hanover, NH 03755
dcvax!dartvax!emilyb

David Marston (603) 883-3556
Daniel Webster College
University Drive
Nashua, NH 03063

dcvax!dartvax!emilyb

In the New York City area there is a non-profit organization for users and vendors of products and services for UNIX systems.

Unigroup of New York
G.P.O. Box 1931
New York, NY 10116
Ed Taylor (212) 513-7777
{attunix,philabs}!pencom!taylor

The New Zealand group provides an annual Workshop and Exhibition and a regular newsletter to its members.

New Zealand UNIX Systems User Group
P.O. Box 13056
University of Waikato
Hamilton, New Zealand

An informal group has started in the St. Louis area:

St. Louis UNIX Users Group
Plus Five Computer Services
765 Westwood, 10A
Clayton, MO 63105
Eric Kiebler (314) 725-9492
ihnp4!plus5!slug

In the San Antonio area the San Antonio UNIX Users (SATUU) meet twice each month with the second Wednesday being a dinner meeting and the third Wednesday being a "roving" meeting at a user site.

San Antonio UNIX Users
7950 Floyd Curl Dr. #102
San Antonio, TX 78229-3955
William T. Blessum, M.D. (512) 692-0977
ihnp4!petro!bles!wtb

In the Seattle area there is a group with over 150 members, a monthly newsletter, and a software exchange system. Meetings are held monthly.

Bill Campbell (206) 232-4164
Seattle UNIX Group Membership Information
6641 East Mercer Way
Mercer Island, WA 98040
uw-beaver!tikal!camco!bill

A UNIX/C language users group has been formed in Tulsa. For current information on meetings, etc. contact:

Pete Rourke
$USR
7340 East 25th Place
Tulsa, OK 74129
Calls for Papers

How to Write a UNIX Daemon

Multiple Programs in One UNIX Process

Book Reviews

tar vs. cpio

UUNET Progress Report

---

Change of Address Form

Please fill out and send the following form through the U.S. mail to the Association Office at the address above.

Name: ___________________________________________________________ Member #: ______

OLD: ____________________________________________________________

______________________________________________________________

______________________________________________________________

______________________________________________________________

NEW: ____________________________________________________________

______________________________________________________________

______________________________________________________________

______________________________________________________________

Phone: __________________________________________________________

uucp: {decvax,ucbvax}! _________________________________________