Changes to the Kernel in 4.3BSD

April 16, 1986

Michael J. Karels

Computer Systems Research Group
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, California 94720

This document summarizes the changes to the kernel between the September 1983 4.2BSD distribution of UNIX† for the VAX‡ and the March 1986 4.3BSD release. It is intended to provide sufficient information that those who maintain the kernel, have local modifications to install, or who have versions of 4.2BSD modified to run on other hardware should be able to determine how to integrate this version of the system into their environment. As always, the source code is the final source of information, and this document is intended primarily to point out those areas that have changed.

Most of the changes between 4.2BSD and 4.3BSD fall into one of several categories. These are:

- bug fixes,
- performance improvements,
- completion of skeletal facilities,
- generalizations of the framework to accommodate new hardware and software systems, or to remove hardware- or protocol-specific code from common facilities, and
- new protocol and hardware support.

The major changes to the kernel are:

- the use of caching to decrease the overhead of filesystem name translation,
- a new interface to the *namei* name lookup function that encapsulates the arguments, return information and side effects of this call,
- removal of most of the Internet dependencies from common parts of the network, and greater allowance for the use of multiple address families on the same network hardware,
- support for the Xerox NS network protocols,
- support for the VAX 8600 and 8650 processors (with UNIBUS and MASSBUS peripherals, but not with CI bus or HSC50 disk controllers),
- new drivers for the DHU11 and DMZ32 terminal multiplexors, the TU81 and other TMSCP tape drives, the VS100 display, the DEUNA, Excelan 204, and Interlan NP100 Ethernet* interfaces, and the ACC HDH and DDN X.25 IMP interfaces, and
- full support for the MS780-E memory controller on the VAX 11/780 and 11/785, using 64K and 256K memory chips.

This document is not intended to be an introduction to the kernel, but assumes familiarity with prior versions of the kernel. Other documents may be consulted for more complete discussions of the kernel and its other subsystems. For more complete information on the internal structure and interfaces of the network

---

† UNIX is a trademark of Bell Laboratories.
‡ DEC, VAX, PDP, MASSBUS, UNIBUS, Q-bus and ULTRIX are trademarks of Digital Equipment Corporation.
* Ethernet is a trademark of Xerox Corporation.

subsystem, refer to ‘‘4.3BSD Networking Implementation Notes.’’

The author gratefully acknowledges the contributions of the other members of the Computer Systems Research Group at Berkeley and the other contributors to the work described here. Major contributors include Kirk McKusick, Sam Leffler, Jim Bloom, Keith Sklower, Robert Elz, and Jay Lepreau. Sam Leffler and Anne Hughes made numerous suggestions and corrections during the preparation of the manuscript.

1. General changes in the kernel

This section details some of the changes that affect multiple sections of the kernel.

1.1. Header files

The kernel is now compiled with an include path that specifies the standard location of the common header files, generally **/sys/h** or **../h**, and all kernel sources have had pathname prefixes removed from the **#include** directives for files in **../h** or the source directory. This makes it possible to substitute replacements for individual header files by placing them in the system compilation directory or in another directory in the include path.

1.2. Types

There have been relatively few changes in the types defined and used by the system. One significant exception is that new typedefs have been added for user ID’s and group ID’s in the kernel and common data structures. These typedefs, *uid_t* and *gid_t*, are both of type *u_short*. This change from the previous usage (explicit *short* ints) allows user and group ID’s greater than 32767 to work reasonably.

1.3. Inline

The inline expansion of calls to various trivial or hardware-dependent operations has been a useful technique in the kernel. In prior releases this substitution was done by editing the assembly language output of the compiler with the sed script *asm.sed*. This technique has been refined in 4.3BSD by using a new program, *inline*, to perform the in-line code expansion and also optimize the code used to push the subroutine’s operands; where possible, *inline* will merge stack pushes and pops into direct register loads. Also, this program performs the in-line code expansion significantly faster than the general-purpose stream editor it replaces.

1.4. Processor priorities

Functions to set the processor interrupt priority to block classes of interrupts have been used in UNIX on all processors, but the names of these routines have always been derived from the priority levels of the PDP11 and the UNIBUS. In order to clarify both the intent of elevated processor priority and the assumptions about their dependencies, all of the functions *splN*, where *N* is a small nonzero integer, have been renamed. In each case, the new name indicates the group of devices that are to be blocked from interrupts. The following table indicates the old and new names of these functions.

new name	devices blocked	old name	VAX IPL
spl0	none	spl0	0
splsoftclock	software clock interrupts	none	0x08
splnet	software network interrupts	splnet	0x0c
spltty	terminal multiplexors	spl5	0x15
splbio	disk and tape controllers	spl5	0x15
splimp	all network interfaces	splimp	0x16
splclock	interval timer	spl6	0x18
splhigh	all devices and state transitions	spl7	0x31

For use in device drivers only, UNIBUS priorities BR4 through BR7 may be set using the functions spl4, spl5, spl6 and spl7. Note that the latter two now correspond to VAX priorities 0x16 and 0x17 respectively, rather than the previous 0x18 and 0x1f priorities.

2. Header files

This section details changes in the header files in **/sys/h**.

acct.h Process accounting is now done in units of 1/**AHZ** (64) seconds rather than seconds.

buf.h The size of the buffer hash table has been increased substantially.

cmap.h The core map has had a number of fields enlarged to support larger memories and filesystems. The limits imposed by this structure are now commented. The current limits are 64 Mb of physical memory, 255 filesystems, 1 Gb process segments, 8 Gb per filesystem, and 65535 processes and text entries. The machine-language support now derives its definitions of these limits and the cmap structure from this file.

dmap.h The swap map per process segment was enlarged to allow images up to 64Mb.

domain.h New entry points to each domain have been added, for initialization, externalization of access rights, and disposal of access rights.

errno.h A definition of EDEADLK was added for System V compatibility.

fs.h One spare field in the superblock was allocated to store an option for the fragment allocation policy.

inode.h New fields were added to the in-core inode to hold a cache key and a pointer to any text image mapping the file. A new macro, ITIMES, is provided for updating the timestamps in an inode without writing the inode back to the disk. The inode is marked as modified with the IMOD flag. A flag has been added to allow serialization of directory renames.

ioctl.h New *ioctl* operations have been added to get and set a terminal or window's size. The size is stored in a *winsize* structure defined here. Other new *ioctl*s have been defined to pass a small set of special commands from pseudo-terminals to their controllers. A new terminal option, LPASS8, allows a full 8-bit data path on input. The two tablet line disciplines have been merged. A new line discipline is provided for use with IP over serial data lines.

mbuf.h The handling of mbuf page clusters has been broken into macros separate from those that handle mbufs. MCLALLOC(*m*, *i*) is used to allocate *i* mbuf clusters (where *i* is currently restricted to 1) and MCLFREE(*m*) frees them. MCLGET(*m*) adds a page cluster to the already-allocated mbuf *m*, setting the mbuf length to CLBYTES if successful. The new macro M_HASCL(*m*) returns true if the mbuf *m* has an associated cluster, and MTOCL(*m*) returns a pointer to such a cluster.

mtio.h Definitions have been added for the TMSCP tape controllers and to enable or disable the use of an on-board tape buffer.

namei.h This header file was renamed, completed and put into use.

param.h Several limits have been increased. Old values are listed in parentheses after each item. The new limits are: 255 mounted filesystems (15), 40 processes per user (25), 64 open files (20), 20480 characters per argument list (10240), and 16 groups per user (8). The maximum length of a host name supported by the kernel is defined here as MAXHOSTNAMELEN. The default creation mask is now set to 022 by the kernel; previously that value was set by login, with the effect that remote shell processes used a different default. Clist blocks were doubled in size to 64 bytes.

proc.h Pointers were added to the *proc* structure to allow process entries to be linked onto lists of active, zombie or free processes.

protosw.h The address family field in the *protosw* structure was replaced with a pointer to the *domain* structure for the address family. Definitions were added for the arguments to the protocol *ctloutput* routines.

signal.h New signals have been defined for window size changes (SIGWINCH) and for user-defined functions (SIGUSR1 and SIGUSR2). The *sv_onstack* field in the *sigvec* structure has been redefined as a flags field, with flags defined for use of the signal stack and for signals to interrupt pending systems calls rather than restarting them. The *sigcontext* structure now includes the frame and argument pointers for the VAX so that the complete return sequence can be done by the kernel. A new macro, *sigmask*, is provided to simplify the use of *sigsetmask*, *sigblock*, and *sigpause*.

socket.h Definitions were added for new options set with *setsockopt*. SO_BROADCAST requests permission to send to the broadcast address, formerly a privileged operation, while SO_SNDBUF and SO_RCVBUF may be used to examine or change the amount of buffer space allocated for a socket. Two new options are used only with *getsockopt*: SO_ERROR obtains any current error status and clears it, and SO_TYPE returns the type of the socket. A new structure was added for use with SO_LINGER. Several new address families were defined.

socketvar.h The character and mbuf counts and limits in the *sockbuf* structure were changed from *short* to *u_short*. SB_MAX defines the limit to the amount that can be placed in a *sockbuf*. The *sosendallatonce* macro was corrected; it previously returned true for sockets using non-blocking I/O. *Soreadable* and *sowriteable* now return true if there is error status to report.

syslog.h The system logging facility has been extended to allow kernel use, and the header file has thus been moved from **/usr/include**.

tablet.h A new file that contains the definitions for use of the tablet line discipline.

text.h Linkage fields have been added to the text structure for use in constructing a text table free list. The structure used in recording text table usage statistics is defined here.

time.h The *time.h* header file has been split. Those definitions relating to the *gettimeofday* system call remain in this file, included as <*sys/time.h*>. The original <*time.h*> file has returned and contains the definitions for the C library time routines.

tty.h The per-terminal data structure now contains the terminal size so that it can be changed dynamically. Files that include <*sys/tty.h*> now require <*sys/ioctl.h*> as well for the *winsize* structure definition.

types.h The new typedefs for user and group ID's are located here. For compatibility and sensibility, the *size_t*, *time_t* and *off_t* types have all been changed from *int* to *long*. New definitions have been added for integer masks and bit operators for use with the *select* system call.

uio.h The offset field in the *uio* structure was changed from *int* to *off_t*. Manifest constants for the *uio* segment values are now provided.

un.h The path in the Unix-domain version of a *sockaddr* was reduced so that use of the entire pathname array would still allow space for a null after the structure when stored in an mbuf.

unpcb.h A Unix-domain socket's own address is now stored in the protocol control block rather than that of the socket to which it is connected. Fields have been added for flow control on stream connections. If a *stat* has caused the assignment of a dummy inode number to the socket, that number is stored here.

user.h The user ID's, group ID's and groups array are declared using the new types for these ID's. A new field was added to handle the new signal flag avoiding system call restarts. The index of the last used file descriptor for the process is maintained in *u.u_lastfile*. The global fields *u_base*, *u_count*, and *u_offset* have been eliminated, with the new *nameidata* structure replacing their remaining function. The *a.out* header is no longer kept in the user structure.

vmmac.h Several macros have been rewritten to improve the code generated by the compiler. New macros were added to lock and unlock *cmap* entries, substituting for *mlock* and *munlock*.

vmmeter.h All counters are now uniformly declared as *long*. Software interrupts are now counted.

3. Changes in the kernel proper

The next several sections describe changes in the parts of the kernel that reside in **/sys/sys**. This section summarizes several of the changes that impact several different areas.

3.1. Process table management

Although the process table has grown considerably since its original design, its use was largely the same as in its first incarnation. Several parts of the system used a linear search of the entire table to locate a process, a group of processes, or group of processes in a certain state. 4.2BSD maintained linkages between the children of each parent process, but made no use of these pointers. In order to reduce the time spent examining the process table, several changes have been made. The first is to place all process table entries onto one of three doubly-linked lists, one each for entries in use by existing processes (*allproc*), entries for zombie processes (*zombproc*), and free entries (*freeproc*). This allows the scheduler and other facilities that must examine all existing processes to limit their search to those entries actually in use. Other searches are avoided by using the linkage among the children of each process and by noting a range of usable process ID's when searching for a new unique ID.

3.2. Signals

One of the major incompatibilities introduced in 4.2BSD was that system calls interrupted by a caught signal were restarted. This facility, while necessary for many programs that use signals to drive background activities without disrupting the foreground processing, caused problems for other, more naive, programs. In order to resolve this difficulty, the 4.2BSD signal model has been extended to allow signal handlers to specify whether or not the signal is to abort or to resume interrupted system calls. This option is specified with the *sigvec* call used to specify the handler. The *sv_onstack* field has been usurped for a flag field, with flags available to indicate whether the handler should be invoked on the signal stack and whether it should interrupt pending system calls on its return. As a result of this change, those system calls that may be restarted and that therefore take control over system call interruptions must be modified to support this new behavior. The calls affected in 4.3BSD are *open*, *read*/*write*, *ioctl*, *flock* and *wait*.

Another change in signal usage in 4.3BSD affects fewer programs and less kernel code. In 4.2BSD, invocation of a signal handler on the signal stack caused some of the saved status to be pushed onto the normal stack before switching to the signal stack to build the call frame. The status information on the normal stack included the saved PC and PSL; this allowed a user-mode *rei* instruction to be used in implementing the return to the interrupted context. In order to avoid changes to the normal runtime stack when switching to the signal stack, the return procedure has been changed. As the return mechanism requires a special system call for restoring the signal state, that system call was replaced with a new call, *sigreturn*, that implements the complete return to the previous context. The old call, number 139, remains in 4.3BSD for binary compatibility with the 4.2BSD version of *longjmp*.

3.3. Open file handling

Previous versions of UNIX have traditionally limited each process to at most 20 files open simultaneously. In 4.2BSD, that limit could not be increased past 30, as a 5-bit field in the page table entry was used to specify either a file number or the reserved values PGTEXT or PGZERO (fill from text file or zero fill). However, the file mapping facility that previously used this field no longer existed, and its replacement is unlikely to require this low limit. Accordingly, the internal virtual memory system support for mapped files has been removed and the number of open files increased. The standard limit is 64, but this may easily be increased if sufficient memory for the user structure is provided. In order to avoid searching through this longer list of open files when the actual number in use is small, the index of the last used open file slot is maintained in the field *u.u_lastfile*. The routines that implement open and close or implicit close (*exit* and *exec*) maintain this field, and it is used whenever the open file array *u.u_ofile* is scanned.

3.4. Niceness

The values for *nice* used in 4.2BSD and previous systems ranged from 0 though 39. Each use of this scheduling parameter offset the actual value by the default, NZERO (20). This has been changed in 4.3BSD to use a range of -20 to 20, with NZERO redefined as zero.

3.5. Software interrupts and terminal multiplexors

The DH11 and DZ11 terminal multiplexor handlers had been modified to use the hardware's received-character silo when those devices were used by the Berknet network. In order to avoid stagnation of input characters and slow response to input during periods of reduced input, the low-level software clock interrupt handler had been made to call the terminal drivers to drain input. When the clock rate was increased in 4.2BSD, the overhead of checking the input silos with each clock tick was increased, and the use of specialized network hardware reduced the need for this optimization. Therefore, the terminal multiplexors in 4.3BSD use per-character interrupts during periods of low input rate, and enable the silos only during periods of high-speed input. While the silo is enabled, the routine to drain it runs less frequently than every clock tick; it is scheduled using the standard timeout mechanism. As a result, the software clock service routine need not to be invoked on every clock tick, but only when timeouts or profiling require service.

3.6. Changes in initialization and kernel-level support

This section describes changes in the kernel files in **/sys/sys** with prefixes *init_* or *kern_*.

init_main.c Several subsystems have new or renamed initialization routines that are called by *main*. These include *pqinit* for process queues, *xinit* for the text table handling routines, and *nchinit* for the name translation cache. The virtual memory startup *setupclock* has been replaced by *vminit*, that also sets the initial virtual memory limits for process 0 and its descendants. Process 1, *init*, is now created before process 2, *pagedaemon*.

init_sysent.c In addition to entries for the two system calls new in 4.3BSD, the system call table specifies a range of system call numbers that are reserved for redistributors of 4.3BSD. Other unused slots in earlier parts of the table should be reserved for future Berkeley use. Syscall 63 is no longer special.

kern_acct.c The process time accounting file in 4.2BSD stored times in seconds rather than clock ticks. This made accounting independent of the clock rate, but was too large a granularity to be useful. Therefore, 4.3BSD uses a smaller but unvarying unit for accounting times, 1/64 second, specified in *acct.h* as its reciprocal AHZ. The *compress* function converts seconds and microseconds to these new units, expressed as before in 16-bit pseudo-floating point numbers.

kern_clock.c The hardware clock handler implements the new time-correction primitive *adjtime* by skewing the rate at which time increases until a specified correction has been achieved. The *bumptime* routine used to increment the time has been changed into a macro. The overhead of software interrupts used to schedule the *softclock* handler has been reduced by noting whether any profiling or timeout activity requires it to run, and by calling *softclock* directly from *hardclock* (with reduced processor priority) if the previous priority was sufficiently low.

kern_descrip.c Most uses of the *getf()* function have been replaced by the GETF macro form. The *dup* calls (including that from *fcntl*) no longer copy the close-on-exec flag from the original file descriptor. Most of the changes to support the open file descriptor high-water mark, *u.u_lastfile*, are in this file. The *flock* system call has had several bugs fixed. Unix-domain file descriptor garbage collection is no longer triggered from *closef*, but when a socket is torn down.

kern_exec.c The *a.out* header used in the course of *exec* is no longer in the user structure, but is local to *exec*. Argument and environment strings are copied to and from the user address space a string at a time using the new *copyinstr* and *copyoutstr* primitives. When invoking an executable script, the first argument is now the name of the interpreter rather than the file name; the file name appears only after the interpreter name and optional argument. An *iput* was moved to avoid a deadlock when the executable image had been opened and marked close-on-exec. The *setregs* routine has been split; machine-independent parts such as signal action modification are done in *execve* directly, and the remaining machine-dependent routine was moved to *machdep.c*. Image size verification using *chksize* checks data and bss sizes separately to avoid overflow on their addition.

kern_exit.c Instead of looping at location 0x13 in user mode if */etc/init* cannot be executed, the system now prints a message and pauses. This is done by *exit* if process 1 could not run. The search for child processes in *exit* uses the child and sibling linkage in the *proc* entry instead of a linear search of the *proc* table. Failures when copying out resource usage information from *wait* are now reflected to the caller.

kern_fork.c One of the two linear searches of the proc table during process creation has been eliminated, the other looks only at active processes. As the first scan is needed only to count the number of processes for this user, it is bypassed for root. A comment dating to version 7 (‘‘Partially simulate the environment so that when it is actually created (by copying) it will look right.’’) has finally been removed; it relates only to PDP-11 code.

kern_mman.c *Chksize* takes an extra argument so that data and bss expansion can be checked separately to avoid problems with overflow.

kern_proc.c The *spgrp* routine has been corrected. An attempt to optimize its $O(n^2)$ algorithm (multiple scans of the process table) did so incorrectly; it now uses the child and sibling pointers in the proc table to find all descendents in linear time. *Pqinit* is called at initialization time to set up the process queues and free all process slots.

kern_prot.c A number of changes were needed to reflect the type changes of the user and group ID’s. The *getgroups* and *setgroups* routines pass groups as arrays of integers and thus must convert. All scans of the groups array look for an explicit NOGROUP terminator rather than any negative group. For consistency, the *setreuid* call sets the process *p_uid* to the new effective user ID instead of the real ID as before. This prevents the anomaly of a process not being allowed to send signals to itself.

kern_resource.c Attempts to change resource limits for process sizes are checked against the maximum segment size that the swap map supports, *maxdmap*. The error returned when attempting to change another user’s priority was changed from EACCESS to EPERM.

kern_sig.c The *sigmask* macro is now used throughout the kernel. The treatment of the *sigvec* flag has been expanded to include the SV_INTERRUPT option. *Kill* and *killpg* have been rewritten, and the errors returned are now closer to those of System V. In particular, unprivileged users may broadcast signals with no error if they managed to kill something, and an attempt to signal process group 0 (one’s own group) when no group is set receives an ESRCH instead of an EINVAL. SIGWINCH joins the class of signals whose default action is to ignore. When a process stops under *ptrace*, its parent now receives a SIGCHLD.

kern_synch.c The CPU overhead of *schedcpu* has been reduced as much as possible by removing loop invariants and by ignoring processes that have not run since the last calculation. When long-sleeping processes are awakened, their priority is recomputed to consider their sleep time. *Schedcpu* need not remove processes with new priorities from their run queues and reinsert them unless they are moving to a new queue. The sleep queues are now treated as circular (FIFO) lists, as the old LIFO behavior caused problems for some programs queued for locks. *Sleep* no longer allows context switches after a panic, but simply drops the processor priority momentarily then returns; this converts sleeps during the filesystem update into busy-waits.

kern_time.c *Gettimeofday* returns the microsecond time on hardware supporting it, including the VAX. It is now possible to set the timezone as well as the time with *settimeofday*. A system call, *adjtime*, has been added to correct the time by a small amount using gradual skew rather than discontinuous jumps forward or backward.

kern_xxx.c The 4.1-compatible *signal* entry sets the signal SV_INTERRUPT option as well as the per-process SOUSIG, which now controls only the resetting of signal action to default upon invocation of a caught signal.

subr_log.c This new file contains routines that implement a kernel error log device. Kernel messages are placed in the message buffer as before, and can be read from there through the log device */dev/klog*.

subr_mcount.c The kernel profiling buffers are allocated with *calloc* instead of *wmemall* to avoid the dramatic decrease in user virtual memory that could be supported after allocation of a large section of *usrpt*.

subr_prf.c Support was added for the kernel error log. The *log* routine is similar to *printf* but does not print on the console, thereby suspending system operation. *Log* takes a priority as well as a format, both of which are read from the log device by the system error logger *syslogd*. *Uprintf* was modified to check its terminal output queue and to block rather than to use all of the system clists; it is now even less appropriate for use from interrupt level. *Tprintf* is similar to *uprintf* but prints to the tty specified as an argument rather than to that of the current user. *Tprintf* does not block if the output queue is overfull, but logs only to the error log; it may thus be used from interrupt level. Because of these changes, *putchar* and *printn* require an additional argument specifying the destination(s) of the character. The *tablefull* error routine was changed to use *log* rather than *printf*.

subr_rmap.c An off-by-one error in *rmget* was corrected.

sys_generic.c The *select* call may now be used with more than 32 file descriptors, requiring that the masks be treated as arrays. The result masks are returned to the user if and only if no error (including EINTR) occurs. A select bug that caused processes to disappear was fixed; *selwakeup* needed to handle stopped processes differently than sleeping processes.

sys_inode.c Problems occurring after an interrupted close were corrected by forcing *ino_close* to return to *closef* even after an interrupt; otherwise, *f_count* could be cleared too early or twice. The code to unhash text pages being overwritten needed to be protected from memory allocations at interrupt level to avoid a bogus ‘‘panic: munhash.’’ The internal routine implementing *flock* was reworked to avoid several bad assumptions and to allow restarts after an interruption.

sys_process.c *Procxmt* uses the new *ptrace.h* header file; hopefully, the next release will have neither *ptrace* nor *procxmt*. The text XTRC flag is set when modifying a pure text image, protecting it from sharing and overwriting.

sys_socket.c The socket involved in an interface *ioctl* is passed to *ifioctl* so that it can call the protocol if necessary, as when setting the interface address for the protocol. It is now possible to be notified of pending out-of-band data by selecting for exceptional conditions.

syscalls.c The system call names here have been made to agree with reality.

3.7. Changes in the terminal line disciplines

tty.c The kernel maintains the terminal or window size in the tty structure and provides *ioctl*s to set and get these values. The window size is cleared on final close. The sizes include rows and columns in characters and may include X and Y dimensions in pixels where that is meaningful. The kernel makes no use of these values, but they are stored here to provide a consistent way to determine the current size. When a new value is set, a SIGWINCH signal is sent to the process group associated with the terminal.

The notions of line discipline exit and final close have been separated. *Ttyclose* is used only at final close, while *ttylclose* is provided for closing down a discipline. Modem control transitions are handled more cleanly by moving the common code from the terminal hardware drivers into the line disciplines; the *l_modem* entry in the *linesw* is now used for this purpose. *Ttymodem* handles carrier transitions for the standard disciplines; *nullmodem* is provided for disciplines with minimal requirements.

A new mode, LPASS8, was added to support 8-bit input in normal modes; it is the input analog of LLITOUT. An entry point, *checkoutq*, has been added to enable internal output operations (*uprintf*, *tprintf*) to check for output overflow and optionally to block to wait for space. Certain operations are handled more carefully than before: the use of the TIOCSTI *ioctl* requires read permission on the terminal, and SPGRP is disallowed if the group corresponds with another user's process. *Ttread* and *ttwrite* both check for carrier drop when restarting after a sleep. An off-by-one consistency check of *uio_iovcnt* in *ttwrite* was corrected. A bug was fixed that caused data to be flushed when opening a terminal that was already open when using the ''old'' line discipline. *Select* now returns true for reading if carrier has been lost. While changing line disciplines, interrupts must be disabled until the change is complete or is backed out. If changing to the same discipline, the close and reopen (and probable data flush) are avoided. The *t_delct* field in the tty structure was not used and has been deleted.

tty_conf.c The line discipline close entries that used *ttyclose* now use *ttylclose*. The two tablet disciplines have been combined. A new entry was added for a Serial-Line link-layer encapsulation for the Internet Protocol, SLIPDISC.

tty_pty.c Large sections of the pseudo-tty driver have been reworked to improve performance and to avoid races when one side closed, which subsequently hung pseudo-terminals. The line-discipline modem control routine is called to clean up when the master closes. Problems with REMOTE mode and non-blocking I/O were fixed by using the raw queue rather than the cannonicalized queue. A new mode was added to allow a small set of commands to be passed to the pty master from the slave as a rudimentary type of *ioctl*, in a manner analogous to that of PKT mode. Using this mode or PKT mode, a *select* for exceptional conditions on the master side of a pty returns true when a command operation is available to be read. *Select* for writing on the master side has been corrected, and now uses the same criteria as *ptcwrite*. As the pty driver depends on normal operation of the tty queues, it no longer permits changes to non-tty line disciplines.

tty_subr.c The *clist* support routines have been modified to use block moves instead of *getc/putc* wherever possible.

tty_tablet.c The two line disciplines have been merged and a number of new tablet types are supported. Tablet type and operating mode are now set by *ioctl*s. Tablets that continuously stream data are now told to stop sending on last close.

4. Changes in the filesystem

The major change in the filesystem was the addition of a name translation cache. A table of recent name-to-inode translations is maintained by *namei*, and used as a lookaside cache when translating each component of each file pathname. Each *namecache* entry contains the parent directory's device and inode, the length of the name, and the name itself, and is hashed on the name. It also contains a pointer to the inode for the file whose name it contains. Unlike most inode pointers, which hold a ''hard'' reference by incrementing the reference count, the name cache holds a ''soft'' reference, a pointer to an inode that may be reused. In order to validate the inode from a name cache reference, each inode is assigned a unique ''capability'' when it is brought into memory. When the inode entry is reused for another file, or when the name of the file is changed, this capability is changed. This allows the inode cache to be handled normally, releasing inodes at the head of the LRU list without regard for name cache references, and allows multiple names for the same inode to be in the cache simultaneously without complicating the invalidation procedure. An additional feature of this scheme is that when opening a file, it is possible to determine whether the file was previously open. This is useful when beginning execution of a file, to check whether the file might be open for writing, and for similar situations.

Other changes that are visible throughout the filesystem include greater use of the ILOCK and IUNLOCK macros rather than the subroutine equivalents. The inode times are updated on each *irele*, not only when the reference count reaches zero, if the IACC, IUPD or ICHG flags are set. This is accomplished with the ITIMES macro; the inode is marked as modified with the new IMOD flag, that causes it to be written to disk when released, or on the next sync.

The remainder of this section describes the filesystem changes that are localized to individual files.

ufs_alloc.c The algorithm for extending file fragments was changed to take advantage of the observation that fragments that were once extended were frequently extended again, that is, that the file was being written in fragments. Therefore, the first time a given fragment is allocated, a best-fit strategy is used. Thereafter, when this fragment is to be extended, a full-sized block is allocated, the fragment removed from it, and the remainder freed for use in subsequent expansion. As this policy may result in increased fragmentation, it is not used when the filesystem becomes excessively fragmented (i.e. when the number of free fragments falls to 2% of the minfree value); the policy is stored in the superblock and may be changed with *tunefs*. The *fserr* routine was converted to use *log* rather than *printf*.

ufs_bio.c I/O operations traced now include the size where relevant.

ufs_inode.c The size of the buffer hash table was increased substantially and changed to a power of two to allow the modulus to be computed with a mask operation. *Iget* invalidates the capability in each inode that is flushed from the inode cache for reuse. The new *igrab* routine is used instead of *iget* when fetching an inode from a name cache reference; it waits for the inode to be unlocked if necessary, and removes it from the free list if it was free. The caller must check that the inode is still valid after the *igrab*. A bug was fixed in *itrunc* that allowed old contents to creep back into a file. When truncating to a location within a block, *itrunc* must clear the remainder of the block. Otherwise, if the file is extended by seeking past the end of file and then writing, the old contents reappear.

ufs_mount.c The *mount* system call was modified to return different error numbers for different types of errors. *Mount* now examines the superblock more carefully before using size field it contains as the amount to copy into a new buffer. If a mount fails for a reason other than the device already being mounted, the device is closed again. When performing the name lookup for the mount point, *mount* must prevent the name translation from being left in the name cache; *umount* must flush all name translations for the device. A bug in *getmdev* caused an inode to remain locked if the specified device was not a block special file; this has been fixed.

ufs_namei.c This file was previously called ufs_nami.c. The *namei* function has a new calling convention with its arguments, associated context, and side effects encapsulated in a single structure. It has been extensively modified to implement the name cache and to cache directory offsets for each process. It may now return ENAMETOOLONG when appropriate, and returns EINVAL if the 8th bit is set on one of the pathname characters. Directories may be foreshortened if the last one or more blocks contain no entries; this is done when files are being created, as the entire directory must already be searched. An entry is provided for invalidating the entire name cache when the 32-bit prototype for capabilities wraps around. This is expected to happen after 13 months of operation, assuming 100 name lookups per second, all of which miss the cache.

A change in filesystem semantics is the introduction of ‘‘sticky’’ directories. If the ISVTX (sticky text) bit is set in the mode of a directory, files may only be removed from that directory by the owner of the file, the owner of the directory, or the superuser. This is enforced by *namei* when the lookup operation is DELETE.

ufs_subr.c The strategy for *syncip*, the internal routine implementing *fsync*, has been modified for large files (those larger than half of the buffer cache). For large files all modified buffers for the device are written out. The old algorithm could run for a very long time on a very large file, that might not actually have many data blocks. The *update* routine now saves some work by calling *iupdate* only for modified inodes. The C replacements for the special VAX instructions have been collected in this file.

ufs_syscalls.c When doing an open with flags O_CREAT and O_EXCL (create only if the file did not exist), it is now considered to be an error if the target exists and is a symbolic link, even if the symbolic link refers to a nonexistent file. This behavior is desirable for reasons of security in programs that create files with predictable names. *Rename* follows the policy of *namei* in disallowing removal of the target of a rename if the target directory is ‘‘sticky’’ and the user is not the owner of the target or the target directory. A serious bug in the open code which allowed directories and other unwritable files to be truncated has been corrected. Interrupted opens no longer lose file descriptors. The *lseek* call returns an ESPIPE error when seeking on sockets (including pipes) for backward compatibility. The error returned from *readlink* when reading something other than a symbolic link was changed from ENXIO to EINVAL. Several calls that previously failed silently on read-only filesystems (*chmod*, *chown*, *fchmod*, *fchown* and *utimes*) now return EROFS. The *rename* code was reworked to avoid several races and to invalidate the name cache. It marks a directory being renamed with IRENAME to avoid races due to concurrent renames of the same directory. *Mkdir* now sets the size of all new directories to DIRBLKSIZE. *Rmdir* purges the name cache of entries for the removed directory.

ufs_xxx.c The routines *uchar* and *schar* are no longer used and have been removed.

quota_kern.c The quota hash size was changed to a power of 2 so that the modulus could be computed with a mask.

quota_ufs.c If a user has run out of warnings and had the hard limit enforced while logged in, but has then brought his allocation below the hard limit, the quota system reverts to enforcing the soft limit, and resets the warning count; users previously were required to log out and in again to get this affect.

4.1. Changes in Interprocess Communication support

uipc_domain.c The skeletal support for the PUP-1 protocol has been removed. A domain for Xerox NS is now in use. The per-domain data structure allows a per-domain initialization routine to be called at boot time.

The *pffindproto* routine, used in creating a socket to support a specified protocol, takes an additional argument, the type of the socket. It checks both the protocol and type, useful when the same protocol implements multiple socket types. If the type is SOCK_RAW and no exact match is found, a *protosw* entry for raw support and a wildcard protocol (number zero) will be used. This allows for a generic raw socket that passes through packets for any given protocol.

The second argument to *pfctlinput*, the generic error-reporting routine, is now declared as a *sockaddr* pointer.

uipc_mbuf.c The mbuf support routines now use the *wait* flag passed to *m_get* or MGET. If M_WAIT is specified, the allocator may wait for free memory, and the allocation is guaranteed to return an mbuf if it returns. In order to prevent the system from slowly going to sleep after exhausting the mbuf pool by losing the mbufs to a leak, the allocator will panic after creating the maximum allocation of mbufs (by default, 256K). Redundant *spl*’s have been removed; most internal routines must be called at *splimp*, the highest priority at which mbuf and memory allocation occur.

When copying mbuf chains *m_copy* now preserves the type of each mbuf. There were problems in *m_adj*, in particular assumptions that there would be no zero-length mbufs within the chain; this was corrected by changing its *n*-pass algorithm for trimming from the tail of the chain to either one- or two-pass, depending on whether the correction was entirely within the last mbuf. In order to avoid return business, *m_pullup* was changed to pull additional data (MPULL_EXTRA, defined in *mbuf.h*) into the contiguous area in the first mbuf, if convenient. *m_pullup* will use the first mbuf of the chain rather then a new one if it can avoid copying.

uipc_pipe.c This ‘‘temporary’’ file has been removed; pipe now uses *socketpair*.

uipc_proto.c New entries in the protocol switch for externalization and disposal of access rights are initialized for the Unix domain protocols.

uipc_socket.c The *socreate* function uses the new interface to *pffindproto* described above if the protocol is specified by the caller. The *soconnect* routine will now try to disconnect a connected socket before reconnecting. This is only allowed if the protocol itself is not connection oriented. Datagram sockets may connect to specify a default destination, then later connect to another destination or to a null destination to disconnect. The *sodisconnect* routine never used its second argument, and it has been removed.

The *sosend* routine, which implements write and send on sockets, has been restructured for clarity. The old routine had the main loop upside down, first emptying and then filling the buffers. The new implementation also makes it possible to send zero-length datagrams. The maximum length calculation was simplified to avoid problems trying to account for both mbufs and characters of buffer space used. Because of the large improvement in speed of data handling when large buffers are used, *sosend* will use page clusters if it can use at least half of the cluster. Also, if not using nonblocking I/O, it will wait for output to drain if it has enough data to fill an mbuf cluster but not enough space in the output queue for one, instead of fragmenting the write into small mbufs. A bug allowing access rights to be sent more than once when using scatter-gather I/O (*sendmsg*) was fixed. A race that occurred when *uiomove* blocked during a page fault was corrected by allowing the protocol send routines to report disconnection errors; as with disconnection detected earlier, *sosend* returns EPIPE and sends a SIGPIPE signal to the process.

The receive side of socket operations, *soreceive*, has also been reworked. The major changes are a reflection of the way that datagrams are now queued; see uipc_socket2.c for further information. The MSG_PEEK flag is passed to the protocol's *usrreq* routine when requesting out-of-band data so that the protocol may know when the out-of-band data has been consumed. Another bug in access-rights passing was corrected here; the protocol is not called to externalize the data when PEEKing.

The *sosetopt* and *sogetopt* functions have been expanded considerably. The options that existed in 4.2BSD all set some flag at the socket level. The corresponding options in 4.3BSD use the value argument as a boolean, turning the flag off or on as appropriate. There are a number of additional options at the socket level. Most importantly, it is possible to adjust the send or receive buffer allocation so that higher throughput may be achieved, or that temporary peaks in datagram arrival are less likely to result in datagram loss. The linger option is now set with a structure including a boolean (whether or not to linger) and a time to linger if the boolean is true. Other options have been added to determine the type of a socket (eg, SOCK_STREAM, SOCK_DGRAM), and to collect any outstanding error status. If an option is not destined for the socket level itself, the option is passed to the protocol using the *ctloutput* entry. *Getopt*'s last argument was changed from *mbuf* * to *mbuf* ** for consistency with *setopt* and the new *ctloutput* calling convention.

Select for exceptional conditions on sockets is now possible, and this returns true when out-of-band data is pending. This is true from the time that the socket layer is notified that the OOB data is on its way until the OOB data has been consumed. The interpretation of socket process groups in 4.2BSD was inconsistent with that of ttys and with the *fcntl* documentation. This was corrected; positive numbers refer to processes, negative numbers to process groups. The socket process group is used when posting a SIGURG to notify processes of pending out-of-band data.

uipc_socket2.c Signal-driven I/O now works with sockets as well as with ttys; *sorwakeup* and *sowwakeup* call the new routine *sowakeup* which calls *sbwakeup* as before and also sends SIGIO as appropriate. Process groups are interpreted in the same manner as for SIGURG.

Larger socket buffers may be used with 4.3BSD than with 4.2BSD; socket buffers (*sockbuf*s) have been modified to use unsigned short rather than short integers for character counts and mbuf counts. This increases the maximum buffer size to 64K−1. These fields should really be unsigned longs, but a socket would no longer fit in an mbuf. So that as much as possible of the allotment may be used, *sbreserve* allows the high-water mark for data to be set as high as 80% of the maximum value (64K), and sets the high-water mark on mbuf allocation to the smaller of twice the character limit and 64K.

In 4.2BSD, datagrams queued in sockbufs were linked through the mbuf *m_next* field, with *m_act* set to 1 in the last mbuf of each datagram. Also, each datagram was required to have one mbuf to contain an address, another to contain access rights, and at least one additional mbuf of data. In 4.3BSD, the mbufs comprising a datagram are linked through *m_next*, and different datagrams are linked through the *m_act* field of the first mbuf in each. No mbuf is used to represent missing components of a datagram, but the ordering of the mbufs remains important. The components are distinguished by the mbuf type. Any address must be in the first mbuf. Access rights follow the address if present, otherwise they may be first. Data mbufs follow; at least one data buffer will be present if there is no address or access rights. The routines *sbappend*, *sbappendaddr*, *sbappendrights* and *sbappendrecord* are used to add new data to a sockbuf. The first of these appends to an existing record, and is commonly used for stream sockets. The other three begin new records with address, optional rights, and data (*sbappendaddr*), with rights and data (*sbappendrights*), or data only (*sbappendrecord*). A new internal routine, *sbcompress*, is used by these functions to compress and append data mbufs to a record. These changes improve the functionality of this layer and in addition make it faster to find the end of a queue.

An occasional ‘‘panic: sbdrop’’ was due to zero-length mbufs at the end of a chain. Although these should no longer be found in a sockbuf queue, *sbdrop* was fixed to free empty buffers at the end of the last record. Similarly, *sbfree* continues to empty a sockbuf as long as mbufs remain, as zero-length packets might be present. *Sbdroprecord* was added to free exactly one record from the front of a sockbuf queue.

uipc_syscalls.c Errors reported during an *accept* call are cleared so that subsequent *accept* calls may succeed. A failed attempt to *connect* returns the error once only, and SOISCONNECTING is cleared, so that additional connect calls may be attempted. (Lower level protocols may or may not allow this, depending on the nature of the failure.) The *socketpair* system call has been fixed to work with datagram sockets as well as with streams, and to clean up properly upon failure. Pipes are now created using *connect2*. An additional argument, the type of the data to be fetched, is passed to *sockargs*.

uipc_usrreq.c The binding and connection of Unix domain sockets has been cleaned up so that *recvfrom* and *accept* get the address of the peer (if bound) rather than their own. The Unix-domain connection block records the bound address of a socket, not the address of the socket to which it is connected. For stream sockets, back pressure to implement flow control is now handled by adjusting the limits in the send buffer without overloading the normal count fields; the flow control information was moved to the connection block. Access rights are checked now when connecting; the connected-to socket must be writable by the caller, or the connection request is denied. In order to test one previously unused routine, the Unix domain stream support was modified to support the passage of access rights. Problems with access-rights garbage collection were also noted and fixed, and a count is kept of rights outstanding so that garbage collection is done only when needed. Garbage collection is triggered by socket shutdown now rather than file close; in 4.2BSD, it happened prematurely. The PRU_SENSE *usrreq* entry, used by *stat*, has been added. It returns the write buffer size as the ‘‘blocksize,’’ and generates a fake inode number and device for the benefit of those programs that use *fstat* information to determine whether file descriptors refer to the same file. Unimplemented requests have been carefully checked to see that they properly free mbufs when required and never otherwise. Larger buffers are allocated for both stream and datagram sockets. A number of minor bugs have been corrected: the back pointer from an inode to a socket needed to be cleared before release of the inode when detaching; sockets can only be bound once, rather than losing inodes; datagram sockets are correctly marked as connected and disconnected; several mbuf leaks were plugged. A serious problem was corrected in *unp_drop*: it did not properly abort pending connections, with the result that closing a socket with unaccepted connections would cause an infinite loop trying to drop them.

4.2. Changes in the virtual memory system

The virtual memory system in 4.3BSD is largely unchanged from 4.2BSD. The changes that have been made were in two areas: adapting the VM substem to larger physical memories, and optimization by simplifying many of the macros.

Many of the internal limits on the virtual memory system were imposed by the *cmap* structure. This structure was enlarged to increase those limits. The limit on physical memory has been changed from 8 megabytes to 64 megabytes, with expansion space provided for larger limits, and the limit of 15 mounted file systems has been changed to 255. The maximum file system size has been increased to 8 gigabytes, number of processes to 65536, and per-process size to 64 megabytes of data and 64 megabytes of stack. Configuration parameters and other segment size limits were converted from pages to bytes. Note that most of these are upper bounds; the default limits for these quantities are tuned for systems with 4-8 megabytes of physical memory. The process region sizes may be adjusted with kernel configuration file options; for example,

```
options        MAXDSIZ=33554432
```

increases the data segment to 32 megabytes. With no option, data segments receive a hard limit of roughly 17Mb and a soft limit of 6Mb (that may be increased with the csh limit command).

The global clock page replacement algorithm used to have a single hand that was used both to mark and to reclaim memory. The first time that it encountered a page it would clear its reference bit. If the reference bit was still clear on its next pass across the page, it would reclaim the page. (On the VAX, the reference bit was simulated using the valid bit.) The use of a single hand does not work well with large physical memories as the time to complete a single revolution of the hand can take up to a minute or more. By the time the hand gets around to the marked pages, the information is usually no longer pertinent. During periods of sudden shortages, the page daemon will not be able to find any reclaimable pages until it has completed a full revolution. To alleviate this problem, the clock hand has been split into two separate hands. The front hand clears the reference bits, and the back hand follows a constant number of pages behind, reclaiming pages that have have not been referenced since the front hand passed. While the code has been written in such a way as to allow the distance between the hands to be varied, we have not yet found any algorithms suitable for determining how to dynamically adjust this distance. The parameters determining the rate of page scan have also been updated to reflect larger configurations. The free memory threshold at which *pageout* begins was reduced from one-fourth of memory to 512K for machines with more than 2 megabytes of user memory. The scan rate is now independent of memory size instead of proportional to memory size.

The text table is now managed differently. Unused entries are treated as a cache, similar to the usage of the inode table. Entries with reference counts of 0 are placed in an LRU cache for potential reuse. In effect, all texts are ‘‘sticky,’’ except that they are flushed after a period of disuse or overflow of the table. The sticky bit works as before, preventing entries from being freed and locking text files into the cache. The code to prevent modification of running texts was cleaned up by keeping a pointer to the text entry in the inode, allowing texts to be freed when unlinking files without linear searches.

The swap code was changed to handle errors a bit better (*swapout* doesn’t do *swkill*s, it just reflects errors to the caller for action there). During swapouts, interrupts are now blocked for less time after freeing the pages of the user structure and page tables (as explained by the old comment from *swapout*, ‘‘XXX hack memory interlock’’), and this is now done only when swapping out the current process. The same situation existed in *exit*, but had not yet been protected by raised priority.

Various routines that took page numbers as arguments now take *cmap* pointers instead to reduce the number of conversions. These include *mlink*, *munlink*, *mlock*, *munlock*, and *mwait*. *Mlock* and *munlock* are generally used in their macro forms.

The remainder of the section details the other changes according to source file.

vm_mem.c Low-level support for mapped files was removed, as the descriptor field in the page table entry was too small. Callers of *munhash* must block interrupts with *splimp* between checking for the presence of a block in the hash list and removing it with *munhash* in order to avoid reallocation of the page and a subsequent panic.

vm_page.c When filling a page from the text file, *pagein* uses a new routine, *fodkluster*, to bring in additional pages that are contiguous in the filesystem. If errors occur while reading in text pages, no page-table change is propagated to other users of the shared image, allowing them to retry and notice the error if they attempt to use the same page. Virtual memory initialization code has been collected into *vminit*, which adjusts swap interleaving to allow the configured size limits, set up the parameters for the clock algorithm, and set the initial virtual memory-related resource limits. The limit to resident-set size is set to the size of the available user memory. This change causes a single large process occupying most of memory to begin random page replacement as memory resources run short. Several races in *pagein* have been detected and fixed. Most of the *pageout* code was moved to *checkpage* in implementing the two-handed clock algorithm.

vm_proc.c The *setjmp* in *procdup* was changed to *savectx*, which saves all registers, not just those needed to locate the others on the stack.

vm_pt.c The *setjmp* call in *ptexpand* was changed to *savectx* to save all registers before initiating a swapout. *Vrelu* does an *splimp* before freeing user-structure pages if running on behalf of the current process. This had been done by *swapout* before, but not by *exit*.

vm_sched.c The swap scheduler looks through the *allproc* list for processes to swap in or out. A call to *remrq* when swapping sleeping processes was unnecessary and was removed. If swapouts fail upon exhaustion of swap space, *sched* does not continue to attempt swapouts.

vm_subr.c The *ptetov* function and the unused *vtopte* function were recoded without using the usual macros in order to fold the similar cases together.

vm_sw.c The error returned by *swapon* when the device is not one of those configured was changed from ENODEV to EINVAL for accuracy. The search for the specified device begins with the first entry so that the error is correct (EBUSY) when attempting to enable the primary swap area.

vm_swap.c The *swapout* routine now leaves any *swkill* to its caller. This avoids killing processes in a few situations. It uses *xdetach* instead of *xccdec*. Several unneeded *spl*'s were deleted.

vm_swp.c The *swap* routine now consistently returns error status. *Physio* was modified to do scatter-gather I/O correctly.

vm_text.c The text routines use a text free list as a cache of text images, resulting in numerous changes throughout this file. *Xccdec* now works only on locked text entries, and is replaced by *xdetach* for external callers. *Xumount* frees unused swap images from all devices when called with NODEV as argument. It is no longer necessary to search the text table to find any text associated with an inode in *xrele*, as the inode stores a pointer to any text entry mapping it. Statistics are gathered on the hit rate of the cache and its cost.

5. Machine specific support

The next several sections describe changes to the VAX-specific portion of the kernel whose sources reside in **/sys/vax**.

5.1. Autoconfiguration

The data structures and top level of autoconfiguration have been generalized to support the VAX 8600 and machines whose main I/O busses are not similar to an SBI. The *percpu* structure has been broken into three structures. The *percpu* structure itself contains only the CPU type, an approximate value for the speed of the CPU, and a pointer to an array of I/O bus descriptions. Each of these, in turn, contain general information about one I/O bus that must be configured and a pointer to the private data for its configuration routine. The third new structure that has been defined describes the SBI and the other interconnects that emulate it. At boot time, *configure* calls *probeio* to configure the I/O bus(ses). *Probeio* looks through the array of bus descriptions, indirecting to the correct routine to configure each bus. For the VAXen currently supported, the main bus is configured by either *probe_Abus* (on the 8600 and 8650) or by *probenexi*, that is used on anything resembling an SBI. Multiple SBI adaptors on the 8600 are handled by multiple calls to *probenexi*. (Although the code has been tested with a second SBI, there were no adaptors installed on the second SBI.) This structure is easily extensible to other architectures using the BI bus, Q bus, or any combination of busses.

The CPU speed value is used to scale the DELAY macro so that autoconfiguration of old devices on faster CPU's will continue to work. The units are roughly millions of instructions per second (MIPS), with a value of 1 for the 780, although fractional values are not used. When multiple CPU's share the same CPU type, the largest value for any of them is used.

UNIBUS autoconfiguration has been modified to accommodate UNIBUS memory devices correctly. A new routine, *ubameminit,* is used to configure UNIBUS memory before probing other devices, and is also used after a UNIBUS reset to remap these memory areas. The device probe or attach routines may then allocate and hold UNIBUS map registers without interfering with these devices.

5.2. Memory controller support

The introduction of the MS780-E memory controller for the VAX 780 made it necessary to configure the memory controller(s) on a VAX separately from the CPU. During autoconfiguration, the types of the memory controllers are recorded in an array. Memory error routines that must know the type of controller then use this information rather than the CPU type. The MS780-E controller is listed as two controllers, as each half reports errors independently. Both 1Mb and 4Mb boards using 64K and 256K dRAM chips are supported.

Locore.c For *lint*'s sake, *Locore.c* has been updated to include the functions provided by *inline* and the new functions in *locore.s*.

autoconf.c Most of the changes to autoconfiguration are described above. Other minor changes: UNIBUS controller probe routines are now passed an additional argument, a pointer to the *uba_ctlr* structure, and similarly device probe routines are passed a pointer to the *uba_device* structure. *Ubaaccess* and *nxaccess* were combined into a single routine to map I/O register areas. A logic error was corrected so that swap device sizes that were initialized from information in the machine configuration file are used unmodified. *Dumplo* is set at configuration time according to the sizes of the dump device and memory.

conf.c Several new devices have been added and old entries have been deleted. A number of devices incorrectly set unused UNIBUS reset entries to *nodev*; these were changed to *nulldev*. An entry was added for the new error log device. Additional device numbers have been reserved for local use.

cons.h New definitions have been added for the 8600 console.

crl.h,crl.c New files for the VAX 8600 console RL02 (our third RL02 driver!).

flp.c It was discovered that not all VAXen that are not 780's are 750's; the console floppy driver for the 780 now checks for cpu == 780, not cpu != 750. An error causing the floppy to be locked in the busy state was corrected.

genassym.c Several new structure offsets were needed by the assembly language routines.

in_cksum.c It was discovered that the instruction used to clear the carry in the checksum loops did not actually clear carry. As the carry bit was always off when entering the checksum loop, this was never noticed.

inline This directory contains the new *inline* program used to edit the assembly language output by the compiler.

locore.s The assembly language support for the kernel has a number of changes, some of which are VAX specific and some of which are needed on all machines. They are simply enumerated here without distinction.

The *doadump* routine sometimes faulted because it changed the page table entry for the *rpb* without flushing the translation buffer. In order to reconfigure UNIBUS memory devices again after UNIBUS resets, *badaddr* was reimplemented without the need to modify the system control block. The machine check handler catches faults predicted by *badaddr*, cleans up and then returns to the error handler. The interrupt vectors have each been modified to count the number of interrupts from their respective devices, so that it is possible to account for software interrupts and UBA interrupts, and to determine which of several similar devices is generating unexpected interrupt loads. The *config* program generates the definitions for the indices into this interrupt count table. Software clock interrupts no longer call timer entries in the dz and dh drivers. The processing of network software interrupts has been reordered so that new interrupts requested during the protocol interrupt routine are likely to be handled before return from the software interrupt. Additional map entries were added to the network buffer and user page table page maps, as both use origin-1 indexing. The memory size limit and the offsets into the coremap are both obtained from *cmap.h* instead of inline constants. The signal trampoline code is all new and uses the *sigreturn* system call to reset signal masks and perform the *rei* to user mode. The initialization code for process 1, *icode*, was moved to this file to avoid hand assembly; it has been changed to exit instead of looping if */etc/init* cannot be executed, and to allow arguments to be passed to *init*. The routines that are called with *jsb* rather than *calls* use a new entry macro that allows them to be profiled if profiling is enabled.

Several new routines were added to move data from address space to address space a character string at a time; they are *copyinstr*, *copyoutstr*, and *copystr*. *Copyin* and *copyout* now receive their arguments in registers. *Setjmp* and *longjmp* are now similar to the user-level routines; *setjmp* saves the stack and frame pointers and PC only (all implemented in line), and *longjmp* unwinds the stack to recover the other registers. This optimizes the common case, *setjmp*, and allows the same semantics for register variables as for stack variables. For swaps and alternate returns using *u.u_save*, however, all registers must be saved as in a context switch, and *savectx* is provided for that purpose.

Redundant context switches were caused by two bugs in *swtch*. First, *swtch* cleared *runrun* before entering the idle loop. Once an interrupt caused a *wakeup*, *runrun* would be set, requesting another context switch at system call exit. Also, the use of the VAX AST mechanism caused a similar problem, posting AST's to one process that would then *swtch* (or might already be in the idle loop), only to catch the AST after being rescheduled and completing its system service. The AST is no longer marked in the process control block and is cancelled during the context switch. The idle loop has been separated from *swtch* for profiling.

machdep.c The *startup* code to calculate the core map size and the limit to the buffer cache's virtual memory allocation was corrected and reworked. The number of buffer pages was reduced for larger memories (10% of the first 2 Mb of physical memory is used for buffers, as before, and 5% thereafter). The default number of buffers or buffer pages may be overridden with configuration-file options. If the number of buffers must be reduced to fit the system page table, a warning message is printed. Buffers are allocated after all of the fully dense data structures, allowing the other tables allocated at boot time to be mapped by the identity map once again. The new signal stack call and return mechanisms are implemented here by *sendsig* and *sigreturn*; *sigcleanup* remains for compatibility with 4.2BSD's *longjmp*. There are a number of modifications for the VAX 8600, particularly in the machine check and memory error handlers and in the use of the console flags. On the VAX-11/750 more translation-buffer parity faults are considered recoverable. The *reboot* routine flushes the text cache before initiating the filesystem update, and may wait longer for the update to complete. The time-of-day register is set, as any earlier time adjustments are not reflected there yet. The *microtime* function was completed and is now used; it is careful not to allow time to appear to reverse during time corrections. An *initcpu* routine was added to enable caches, floating point accelerators, etc.

machparam.h The file *vax/param.h* was renamed to avoid ambiguity when including *‘‘param.h’’*.

ns_cksum.c This new file contains the checksum code for the Xerox NS network protocols.

pcb.h The *aston*() and *astoff*() macros no longer set an AST in the process control block (see *locore.s*).

pte.h The *pg_blkno* field was increased to 24 bits to correspond with the *cmap* structure; the *pg_fileno* field was reduced to a single bit, as it no longer contains a file descriptor.

swapgeneric.c *Dumpdev* and *argdev* are initialized to NODEV, preventing accidents should they be used before configuration completes. DEL is now recognized as an erase character by the kernel *gets*.

tmscp.h A new file which contains definitions for the Tape Mass Storage Control Protocol.

trap.c Syscall 63 is no longer reserved by *syscall* for out-of-range calls. In order to make *wait3* restartable, *syscall* must not clear the carry bit in the program status longword before beginning a system call, but only after successful completion.

tu.c There were several important fixes in the console TU58 driver.

vm_machdep.c The *chksize* routine requires an additional argument, allowing it to check data size and bss growth separately without overflow.

vmparam.h The limits to user process virtual memory allow nondefault values to be defined by configuration file options. The definition of DMMAX here now defines only the maximum value; it will be reduced according to the definition of MAXDSIZ. The space allocated to user page tables was increased substantially. The free-memory threshold at which *pageout* begins was changed to be at most 512K.

6. Network

There have been many changes in the kernel network support. A major change is the addition of the Xerox NS protocols. During the course of the integration of a second major protocol family to the kernel, a number of Internet dependencies were removed from common network code, and structural changes were made to accommodate multiple protocol and address families simultaneously. In addition, there were a large number of bug fixes and other cleanups in the general networking code and in the Internet protocols. The skeletal support for PUP that was in 4.2BSD has been removed.

The link layer drivers were changed to save an indication of the incoming interface with each packet received, and this information was made available to the protocol layer. There were several problems that could be corrected by taking advantage of this change. The IMP code needed to save error packets for software interrupt-level processing in order to fix a race condition, but it needed to know which interface had received the packet when decoding the addresses. ICMP needed this information to support information requests and (newly added) network mask requests properly, as these request information about a specific network. IP was able to take advantage of this change to implement redirect generation when the incoming and outgoing interfaces are the same.

6.1. Network common code

The changes in the common support routines for networking, located in **/sys/net**, are described here.

if_arp.h This new file contains the definitions for the Address Resolution Protocol (ARP) that are independent of the protocols using ARP.

if.c Most of the *if_ifwith** functions that returned pointers to *ifnet* structures were converted to *ifa_with** equivalents that return pointers to *ifaddr* structures. The old *if_ifonnetof* function is no longer provided, as there is no concept of network number that is independent of address family. A new routine, *ifa_ifwithdstaddr*, is provided for use with point-to-point interfaces. Interface *ioctl*s that set interface addresses are now passed to the appropriate protocol using the PRU_CONTROL request of the *pr_usrreq* entry. Additional *ioctl* operations were added to get and set interface metrics and to manipulate the ARP table (see *netinet/if_ether.c*).

if.h In 4.2BSD, the per-interface structure *ifnet* held the address of the interface, as well as the host and network numbers. These have all been moved into a new structure, *ifaddr*, that is managed by the address family. The *ifnet* structure for an interface includes a pointer to a linked list of addresses for the interface. The IFF_ROUTE flag was also removed. The software loopback interface is distinguished with a new flag. Each interface now has a routing metric that is stored by the kernel but only interpreted by user-level routing processes. Additional interface *ioctl* operations allow the metric or the broadcast address to be read or set. When received packets are passed to the receiving protocol, they include a reference to the incoming interface; a variant of the IF_DEQUEUE macro, IF_DEQUEUEIFP, dequeues a packet and extracts the information about the receiving interface.

if_loop.c The software loopback driver now supports Xerox NS and Internet protocols. It was modified to provide information on the incoming interface to the receiving protocol. The loopback driver's address(es) must now be set with *ifconfig*.

if_sl.c This file was added to support a customized line discipline for the use of an asynchronous serial line as a network interface. Until the encapsulation is changed the interface supports only IP traffic.

raw_cb.c Raw sockets record the socket's protocol number and address family in a *sockproto* structure in the raw connection block. This allows a wildcard raw protocol entry to support raw sockets using any single protocol.

raw_cb.h A *sockproto* description and a hook for protocol-specific options were added to the raw protocol control block.

raw_usrreq.c A bug was fixed that caused received packet return addresses to be corrupted periodically; an mbuf was being used after it was freed. Routing is no longer done here, although the raw socket protocol control block includes a routing entry for use by the transport protocol. The SO_DONTROUTE flag now works correctly with raw sockets.

route.c The routing algorithm was changed to use the first route found in the table instead of the one with the lowest use count. This reduces routing overhead and makes response more predictable. The load-sharing effect of the old algorithm was minimal under most circumstances. Several races were fixed. The hash indexes have been declared as unsigned; negative indices worked for the network route hash table but not for the host hash table. (This fix was included on most 4.2BSD tapes.) New routes are placed at the front of the hash chains instead of at the end. The redirect handling is more robust; redirects are only accepted from the current router, and are not used if the new gateway is the local host. The route allocated while checking a redirect is freed even if the redirect is disbelieved. Host redirects cause a new route to be created if the previous route was to the network. Routes created dynamically by redirects are marked as such. When adding new routes, the gateway address is checked against the addresses of point-to-point links for exact matches before using another interface on the appropriate network. *Rtinit* takes arguments for flags and operation separately, allowing point-to-point interfaces to delete old routes.

route.h The size of the routing hash table has been changed to a power of two, allowing unsigned modulus operations to be performed with a mask. The size of the table is expanded if the GATEWAY option is configured.

7. Internet network protocols

There are numerous bug fixes and extensions in the Internet protocol support (**/sys/netinet**). This section describes some of the more important changes with very little detail. As many of the changes span several source files, and as it is very difficult to merge this code with earlier versions of these protocols, it is strongly recommended that the 4.3BSD network be adopted intact, with local hacks merged into it only if necessary.

7.1. Internet common code

By far, the most important change in IP and the shared Internet support layer is the addition of subnetwork addressing. This facility is used (and required) by a number of large university and other networks that include multiple physical networks as well as connections with the DARPA Internet. Subnet support allows a collection of interconnected local networks to share a single network number, hiding the complexity of the local environment and routing from external hosts and gateways. The subnet support in 4.3BSD conforms with the Internet standard for subnet addressing, RFC-950. For each network interface, a network mask is set along with the address. This mask determines which portion of the address is the network number, including the subnet, and by default is set according to the network class (A, B, or C, with 8, 16, or 24 bits of network part, respectively). Within a subnetted network each subnet appears as a distinct network; externally, the entire network appears to be a single entity.

Another important change in IP addressing is a change to the default IP broadcast address. The default broadcast address is the address with a host part of all ones (using the definition INADDR_BROADCAST), in conformance with RFC-919. In 4.2BSD, the broadcast address was the address with a host part of all zeros (INADDR_ANY). To facilitate the conversion process, and to help avoid breaking networks with forwarded broadcasts, 4.3BSD allows the broadcast address to be set for each interface. IP recognizes and accepts network broadcasts as well as subnet broadcasts when subnets are enabled. Such broadcasts normally originate from hosts that do not know about subnets. IP also accepts old-style (4.2) broadcasts using a host part of all zeros, either as a network or subnet broadcast. An address of all ones is recognized as ‘‘broadcast on this network,’’ and an address of all zeros is accepted as well. The latter two are sometimes used in broadcast information requests or network mask requests in the course of starting a diskless workstation. ICMP includes support for the Network Mask Request and Response. A new routine, *in_broadcast*, was added for the use of link layer output routines to determine whether an IP packet should be broadcast.

Network numbers are now stored and used unshifted to minimize conversions and reduce the overhead associated with comparisons. 4.2BSD shifted network numbers to the low-order part of the word. The structure defining Internet addresses no longer includes the old IMP-host fields, but only a featureless 32-bit address.

in.h The definitions of Internet port numbers in this file were deleted, as they have been superceded by the *getservicebyname* interface. A definition was added for the single option at the IP level accessible through *setsockopt*, IP_OPTIONS.

in_pcb.h The Internet protocol control block includes a pointer to an optional mbuf containing IP options.

in_var.h This new header file contains the declaration of the Internet variety of the per-interface address information. The *in_ifaddr* structure includes the network, subnet, network mask and broadcast information.

in.c The *if_** routines which manipulate Internet addresses were renamed to *in_**. *in_netof* and *in_lnaof* check whether the address is for a directly-connected network, and if so they use the local network mask to return the subnet/net and host portions, respectively. *in_localaddr* determines whether an address corresponds to a directly-connected network. By default, this includes any subnet of a local network; a configuration option, SUBNETSARELOCAL=0, changes this to return true only for a directly-connected subnet or non-subnetted network. Interface *ioctl*s that get or set addresses or related status information are forwarded to *in_control*, which implements them. *in_iaonnetof* replaces *if_ifonnetof* for Internet addresses only.

in_pcb.c The destination address of a *connect* may be given as INADDR_ANY (0) as a shorthand notation for ‘‘this host.’’ This simplifies the process of connecting to local servers such as the name-domain server that translates host names to addresses. Also, the short-hand address INADDR_BROADCAST is converted to the broadcast address for the primary local network; it fails if that network is incapable of broadcast. The source address for a connection or datagram is selected according to the outgoing interface; the initial route is allocated at this time and stored in the protocol control block, so that it may be used again when actually sending the packet(s). The *in_pcbnotify* routine was generalized to apply any function and/or report an error to all connections to a destination; it is used to notify connections of routing changes and other non-error situations as well as errors. New entries have been added to this level to invalidate cached routes when routing changes occur, as well as to report possible routing failures detected by higher levels.

in_proto.c The protocol switch table for Internet protocols includes entries for the *ctloutput* routines. ICMP may be used with raw sockets. A raw wildcard entry allows raw sockets to use any protocol not already implemented in the kernel (e.g., EGP).

7.2. IP

Support was added for IP source routing and other IP options (partly derived from BBN's implementation). On output, IP options such as strict or loose source route and record may be set by a client process using TCP, UDP or raw IP sockets. IP properly updates source-route and record-route options when forwarding (and leaves them in the packet, unlike 4.2 which stripped them out after updating). IP input preserves any source-routing information in an incoming packet and passes it up to the receiving protocol upon request, reversing it and arranging it in the same way as user-supplied options. Both TCP and ICMP retrieve incoming source routes for use in replies. Most of the option-handling code has been converted to use *bcopy* instead of structure assignments when copying addresses, as the alignment in the incoming packet may not be correct for the host. This is not required on the VAX, but is needed on most other machines running 4.2BSD.

ip.h The IP time-to-live field is decremented by one when forwarding; in 4.2BSD this value was five.

ip_var.h Data structures and definitions were added for storing IP options. New fields have been added to the structure containing IP statistics.

ip_input.c The changes to save and present incoming IP source-routing information to higher level protocols are in this file. The identity of the interface that received the packet is also determined by *ip_input* and passed to the next protocol receiving the packet. To avoid using uninitialized data structures, IP must not begin receiving packets until at least one Internet address has been set. A bug in the reassembly of IP packets with options has been corrected. Machines with only a single network interface (in addition to the loopback interface) no longer attempt to forward received IP packets that are not destined for them; they also do not respond with ICMP errors unless configured with the GATEWAY option. This change prevents large increases in network activity which used to result when an IP packet that was broadcast was not understood as a broadcast. A one-element route cache was added to the IP forwarding routine. When a packet is forwarded using the same interface on which it arrived, if the source host is on the directly-attached network, an ICMP redirect is sent to the source. If the route used for forwarding was a route to a host or a route to a subnet, a host redirect is used, otherwise a network redirect is sent. The generation of redirects may be disabled by a configuration option, IPSENDREDIRECTS=0. More statistics are collected, in particular on traffic and fragmentation. The *ip_ctlinput* routine was moved to each of the upper-level protocols, as they each have somewhat different requirements.

ip_output.c The IP output routine manages a cached route in the protocol control block for each TCP, UDP or raw IP socket. If the destination has changed, the route has been marked down, or the route was freed because of a routing change, a new route is obtained. The route is not used if the IP_ROUTETOIF (aka SO_DONTROUTE or MSG_DONTROUTE) option is present. Preformed IP options passed to *ip_output* are inserted, changing the destination address as required. The *ip_ctloutput* routine allows options to be set for an individual socket, validating and internalizing them as appropriate.

raw_ip.c The type-of-service and offset fields in the IP header are set to zero on output. The SO_DONTROUTE flag is handled properly.

7.3. ICMP

There have been numerous fixes and corrections to ICMP. Length calculations have been corrected, allowing most ICMP packet lengths to be received and allowing errors to be sent about smaller input packets. ICMP now uses information about the interface on which a message was received to determine the correct source address on returned error packets and replies to information requests. Support was added for the Network Mask Request. Responses to source-routed requests use the reversed source route for the return trip. Timestamps are created with *microtime*, allowing 1-millisecond resolution. The *icmp_error* routine is capable of sending ICMP redirects. When processing network redirects, the returned source address is converted to a network address before passing it to the routing redirect handler. The translation of ICMP errors to Unix error returns was updated.

7.4. TCP

In addition to bug fixes, several performance changes have been made to TCP. Several of these address overall network performance and congestion avoidance, while others address performance of an individual connection. The most important changes concern the TCP send policy. First, the sender silly-window syndrome avoidance strategy was fixed. In 4.2BSD, the amount that could be sent was compared to the offered window, and thus small amounts could still be sent if the receiver offered a silly window. Once this was fixed, there were problems with peers that never offered windows large enough for a maximum segment, or at least 512 bytes (e.g., the peer is a TAC or an IBM PC). Code was then added to maintain estimates of the peer's receive and send buffer sizes. The send policy will now send if the offered window is at least one-half of the receiver's buffer, as well as when the window is at least a full-sized segment. (When the window is large enough for all data that is queued, the data will also be sent.) The send buffer size estimate is not yet used, but is desired for a new delayed-acknowledgement scheme that has yet to be tested. Another problem that was exposed when the silly-window avoidance was fixed was that the persist code didn't expect to be used with a non-zero window. The persist now lasts only until the first timeout, at which time a packet is sent of the largest size allowed by the window. If this packet is not acknowledged, the output routine must begin retransmission rather than returning to the persist state.

Another change related to the send policy is a strategy designed to minimize the number of small packets outstanding on slow links. This is an implementation of an algorithm proposed by John Nagle in RFC-896. The algorithm is very simple: when there is outstanding, unacknowledged data pending on a connection, new data are not sent unless they fill a maximum-sized segment. This allows bulk data transfers to proceed, but causes small-packet traffic such as remote login to bundle together data received during a single round-trip time. On high-bandwidth, low-delay networks such as a local Ethernet, this change seldom causes delay, but over slow links or across the Internet, the number of small packets can be reduced considerably. This algorithm does interact poorly with one type of usage, however, as demonstrated by the X window system. When small packets are sent in a stream, such as when doing rubber-banding to position a new window, and when no echo or other acknowledgement is being received from the other end of the connection, the round-trip delay becomes as large as the delayed-acknowledgement timer on the remote end. For such clients, a TCP option may be set with *setsockopt* to defeat this part of the send policy.

For bulk-data transfers, the largest single change to improve performance is to increase the size of the send and receive buffers. The default buffer size in 4.3BSD is 4096 bytes, double the value in 4.2BSD. These values allow more outstanding data and reduce the amount of time waiting for a window update from the receiver. They also improve the utility of the delayed-acknowledgement strategy. The delayed acknowledgment strategy withholds acknowledgements until a window update would uncover at least 35% of the window; in 4.2BSD, with 1024-byte packets on an Ethernet and 2048-byte windows, this took only a single packet. With 4096-byte windows, up to 50% of the acknowledgements may be avoided.

The use of larger buffers might cause problems when bulk-data transfers must traverse several networks and gateways with limited buffering capacity. The source-quench ICMP message was provided to allow gateways in such circumstances to cause source hosts to slow their rate of packet injection into the network. While 4.2BSD ignored such messages, the 4.3BSD TCP includes a mechanism for throttling back the sender when a source quench is received. This is done by creating an artificially small window (one which is 80% of the outstanding data at the time the quench is received, but no less than one segment).

This artificial congestion window is slowly opened as acknowledgements are received. The result under most circumstances is a slow fluctuation around the buffering limit of the intermediate gateways, depending on the other traffic flowing at the same time.

A final set of changes designed to improve network throughput concerns the retransmission policy. The retransmission timer is set according to the current round-trip time estimate. Unfortunately, the round-trip timing code in 4.2BSD had several bugs which caused retransmissions to begin much too early. These bugs in round trip timing have been corrected. Also, the retransmission code has been tuned, using a faster backoff after the first retransmission. On an initial connection request where there is no round-trip time estimate, a much more conservative policy is used. When a slow link intervenes between the sender and the destination, this policy avoids queuing large numbers of retransmitted connection requests before a reply can be received. It also avoids saturation when the destination host is down or nonexistent. During a connection, when the retransmission timer expires, only a single packet is sent. When only a single packet has been lost, this avoids resending data that was successfully received; when a host has gone down or become unreachable, it avoids sending multiple packets at each timeout. Once another acknowledgement is received, the transmission policy returns to normal.

4.2BSD offered a maximum receive segment size of 1024 for all connections, and accepted such offers whenever made. However, that size was especially poor for the Arpanet and other 1822-based IMP networks (sorry, make that PSN networks) where the maximum packet size is 1007 bytes. This was compounded by a bug in the LH/DH driver that did not allow space for an end-of-packet bit in the receive buffer, and thus maximum size packets that were received were split across buffers. This, in turn, aggravated a hardware problem causing small packets following a segmented packet to be concatenated with the previous packet. The result of this set of conditions was that performance across the Arpanet was sometimes abominably slow. The maximum size segment selected by 4.3BSD is chosen according to the destination and the interface to be used. The segment size chosen is somewhat less than the maximum transmission unit of the outgoing interface. If the destination is not local, the segment size is a convenient small size near the default maximum size (512 bytes). This value is both the maximum segment size offered to the sender by the receive side, and the maximum size segment that will be sent. Of course, the send size is also limited to be no more than the receiver has indicated it is willing to receive.

The initial sequence number prototype for TCP is now incremented much more quickly; this has exposed two bugs. Both the window-update receiving code and the urgent data receiving code compared sequence numbers to 0 the first time they were called on a connection. This fails if the initial sequence number has wrapped around to negative numbers. Both are now initialized when the connection is set up. This still remains a problem in maintaining compatibility with 4.2BSD systems; thus an option, TCP_COMPAT_42, was added to avoid using such sequence numbers until 4.2 systems have been upgraded.

Additional changes in TCP are listed by source file:

tcp_input.c The common case of TCP data input, the arrival of the next expected data segment with an empty reassembly queue, was made into a simplified macro for efficiency. *Tcp_input* was modified to know when it needed to call the output side, reducing unnecessary tests for most acknowledgement-only packets. The receive window size calculation on input was modified to avoid shrinking the offered window; this change was needed due to a change in input data packaging by the link layer. A bug in handling TCP packets received with both data and options (that are not supposed to be used) has been corrected. If data is received on a connection after the process has closed, the other end is sent a reset, preventing connections from hanging in CLOSE_WAIT on one end and FIN_WAIT_2 on the other. (4.2BSD contained code to do this, but it was never executed because such input packets had already been dropped as being outside of the receive window.) A timer is now started upon entering FIN_WAIT_2 state if the local user has closed, closing the connection if the final FIN is not received within a reasonable time. Half-open connections are now reset more reliably; there were circumstances under which one end could be rebooted, and new connection requests that used the same port number might not receive a reset. The urgent-data code was modified to remember which data had already been read by the user, avoiding possible confusion if two urgent-data signals were received close together. Another change was made specifically for connections with a TAC. The TAC doesn't fill in the window field on its initial packet (SYN), and the apparent window is random. There is some question as to the validity of the window field if the packet does not have ACK set, and therefore TCP was changed to ignore the window information on those packets.

tcp_output.c The advertised window is never allowed to shrink, in correspondence with the earlier change in the input handler. The retransmit code was changed to check for shrinking windows, updating the connection state rather than timing out while waiting for acknowledgement. The modifications to the send policy described above are largely within this file.

tcp_timer.c The timer routines were changed to allow a longer wait for acknowledgements. (TCP would generally time out before the routing protocol had changed routes.)

7.5. UDP

An error in the checksumming of output UDP packets was corrected. Checksums are now checked by default, unless the COMPAT_42 configuration option is specified; it is provided to allow communication with the 4.2BSD UDP implementation, which generates incorrect checksums. When UDP datagrams are received for a port at which no process is listening, ICMP unreachable messages are sent in response unless the input packet was a broadcast. The size of the receive buffer was increased, as several large datagrams and their attached addresses could otherwise fill the buffer. The time-to-live of output datagrams was reduced from 255 to 30. UDP uses its own *ctlinput* routine for handling of ICMP errors, so that errors may be reported to the sender without closing the socket.

7.6. Address Resolution Protocol

The address resolution protocol has been generalized somewhat. It was specific for IP on 10 Mb/s Ethernet; it now handles multiple protocols on 10 Mb/s Ethernet and could easily be adapted to other hardware as well. This change was made while adding ARP resolution of trailer protocol addresses. Hosts desiring to receive trailer encapsulations must now indicate that by the use of ARP. This allows trailers to be used between cooperating 4.3 machines while using non-trailer encapsulations with other hosts. The negotiation need not be symmetrical: a VAX may request trailers, for example, and a SUN may note this and send trailer packets to the VAX without itself requesting trailers. This change requires modifications to the 10 Mb/s Ethernet drivers, which must provide an additional argument to *arpresolve*, a pointer for the additional return value indicating whether trailer encapsulations may be sent. With this change, the IFF_NOTRAILERS flag on each interface is interpreted to mean that trailers should not be requested. Modifications to ARP from SUN Microsystems add *ioctl* operations to examine and modify entries in the ARP address translation table, and to allow ARP translations to be ‘‘published.’’ When future requests are received for Ethernet address translations, if the translation is in the table and is marked as published, they will be answered for that host. Those modifications superceded the ‘‘oldmap’’ algorithmic translation from IP addresses, which has been removed. Packets are not forwarded to the loopback interface if it is not marked up, and a bug causing an mbuf to be freed twice if the loopback output fails was corrected. ARP complains if a host lists the broadcast address as its Ethernet address. The ARP tables were enlarged to reflect larger network configurations now in use. A new function for use in driver messages, *ether_sprintf*, formats a 48-bit Ethernet address and returns a pointer to the resulting string.

7.7. IMP support

The support facilities for connections to an 1822 (or X.25) IMP port (**/sys/netimp**) have had several bug fixes and one extension. Unit numbers are now checked more carefully during autoconfiguration. Code from BRL was installed to support class B and C networks. Error packets received from the IMP such as Host Dead are queued in the interrupt handler for reprocessing from a software interrupt, avoiding state transitions in the protocols at priorities above *splnet*. The host-dead timer is no longer restarted when attempting new output, as a persistent sender could otherwise prevent new output from being attempted once a host was reported down. The network number is always taken from the address configured for the interface at boot time; network 10 is no longer assumed. A timer is used to prevent blocking if RFNM messages from the IMP are lost. A race was fixed when freeing mbufs containing host table entries, as the mbuf had been used after it was freed.

8. Xerox Network Systems Protocols

4.3BSD now supports some of the Xerox NS protocols. The kernel will allow the user to send or receive IDP datagrams directly, or establish a Sequenced Packet connection. It will generate Error Protocol packets when necessary, and may close user connections if this is the appropriate action on receipt of such packets. It will respond to Echo Protocol requests. The Routing Information Protocol is executed by a user level process, and sufficient access has been left for other protocols to be implemented using IDP datagrams. It would be possible to set the additional fields required for the Packet Exchange format at user level, to provide a daemon to respond to time-of-day requests, or conduct an expanding ring broadcast to discover clearinghouses.

Wherever possible, the algorithms and data structures parallel those used in Internet protocol support, so that little extra effort should be required to maintain the NS protocols. There has not yet been much effort at tuning.

8.1. Naming

A machine running 4.3 is allowed to have only one six-byte NS host address, but is permitted to be on several networks. As in the Internet case, an address of all zeros may be used to bind the host address for an offered service. Unlike the Internet case, an address of all zeros cannot be used to contact a service on the same machine. (This should be changed.)

There is only one name space of port numbers, as opposed to the Internet case where each protocol has its own port space.

Several point-to-point connections can share the same network number. The destination of a point-to-point connection can have a different network number from the local end.

The files *ns.h*, *ns_pcb.h*, *ns.c*, *ns_pcb.c* and *ns_proto.c* are direct translations of similarly named files in the netinet directory. *Ns_pcbnotify* differs a little from *in_pcbnotify* in that it takes an extra parameter which it will pass to the ‘‘notification’’ routine argument indirectly, by stuffing it in each NS control block selected.

This header file *ns_if.h* contains the declaration of the NS variety of the per-interface address information, like *netinet/in_var.h*.

8.2. Encapsulations

The stipulation that each host is allowed exactly one 6 byte address implies that each 10 Mb/s Ethernet interface other than the first will need to reprogram its physical address. All the 10 Mb/s Ethernet drivers supplied with 4.3BSD perform this. The 3 Mb/s Ethernet driver does not perform any address resolution, but uses the 6th byte of the NS host address as a PUP host number, making it largely incompatible with altos running XNS. In a system with both 3 Mb/s and 10 Mb/s Ethernets, one should configure the 3 Mb/s network first.

The file *ns_ip.c* contains code providing a mechanism for sending XNS packets over any medium supporting IP datagrams. It builds objects that look like point-to-point interfaces from the point of view of NS, and a protocol from the point of view of IP. Each of these pseudo interface structures has extra IP data at the end (a route, source and destination), and fits exactly into an mbuf. If the *ifnet* structure grows any larger, the extra data will have to be put in a separate mbuf, or the whole scheme will have to be reworked more rationally.

8.3. Datagrams

The files *ns_input.c* and *ns_output.c* contain the base level routines which interact with network interface drivers. There is a kernel variable *idp_cksum*, which can be used to defeat checksums for all packets. (There ought to be an option per socket to do this). The NS output routine manages a cached route in the protocol control block of each socket. If the destination has changed, the route has been marked down, or the route was freed because of a routing change, a new route is obtained. The route is not used if the NS_ROUTETOIF (aka SO_DONTROUTE or MSG_DONTROUTE) option is present.

The files *idp.h*, *idp_var.h*, and *idp_usrreq.c* are the analogues of *udp.h*, *udp_var.h*, and *udp_usrreq.c*.

8.4. Error and Echo protocols

Routines for processing incoming error protocol packets are in *ns_error.c*. They call *ctlinput* routines for IDP and SPP to maintain structural similarity to the Internet implementation. The kernel will generate error messages indicating lack of a listener at a port, incorrectly received checksum, or that a packet was thrown away due to insufficient resources at the recipient (buffer full). The echo protocol is handled as a special case. If there is no listener at port number 2, then the routine that generates the ‘‘no listener’’ error message will inspect the packet to see if it was an echo request, and if so, will echo it. Thus, the user is free to construct his own echoing daemon if he so chooses.

8.5. Sequenced Packet Protocol

In general, this code employs the Internet TCP algorithms where possible. By default, a three-way handshake is used in establishing connections. There is a compile time option to employ the minimal two way handshake. Incoming connections may multiplexed by source machine and port, as in the Internet case. It will switch over ports when establishing connections if requested to do so.

The retransmission timing and strategies are much like those of TCP, though recent performance enhancements have not yet migrated here. There has not yet been much opportunity to tune this implementation. The code is intended to generate keep-alive packets, though there is some evidence this isn’t working yet. The TCP source-quench strategy hasn’t been added either. The default nominal packet size is 576 bytes, and the default amount of buffering is 2048. It is possible to raise both by setting appropriate socket options.

9. VAX Network Interface drivers

Most of the changes in the network interfaces follow common patterns that are summarized in categories. In addition, there are a number of bug fixes. The change that was made universally to the interface handlers was to remove the *ioctl* routines that set the interface address and flags, replacing them by simpler routines that merely initialize the hardware if this has not already been done. Several of the drivers notice when the IFF_UP flag is cleared and perform a hardware reset, then reinitialize the interface when IFF_UP is set again. This allows interfaces to be turned off, and also provides a mechanism to reset devices that have lost interrupts or otherwise stopped functioning. The handling of the other interface flags has been made more consistent. IFF_RUNNING is used uniformly to indicate that UNIBUS resources have been allocated and that the board has been initialized. The reset routines clear this flag before reinitializing so that both operations will be repeated.

9.1. Interface UNIBUS support

The UNIBUS common support routines for network interfaces have been modified to support multiple transmit and receive buffers per device. A set of macros provide a nearly-compatible interface for devices using a single buffer of each type. When placing received packets into mbufs, *if_ubaget* prepends a pointer to the receiving interface to the data; this requires that the interface pointer be passed to *if_ubaget* or *if_rubaget* as an additional argument. When removing the trailer header from the front of a packet, interface receive routines must move the interface pointer which precedes the header; see one of the existing drivers for an example. When received data is larger than half of an mbuf cluster, the data will be placed in an mbuf cluster rather than a chain of small mbufs. Similarly, in *if_ubaput*, clusters may be remapped instead of copied if they are at least one-half full and are the last mbuf of the chain. For devices like the DEC DEUNA that wish to perform receive operations on a transmit buffer, the transmit buffers are marked. Receive operations from transmit buffers force page mapping to be consistent before attempting to read data or swap pages from them.

9.2. 10 Mb/s Ethernet

The 10Mb/s Ethernet handlers have been modified to use the new ARP interfaces. They no longer use *arpattach*, and the call to *arpresolve* contains an additional argument for a second return, a boolean for the use of trailer encapsulations. Input and output functions were augmented to handle NS IDP packets. For hosts using Xerox NS with multiple interfaces, the drivers are able to reprogram the physical address on each board so that all interfaces use the address of the first configured interface. The hardware Ethernet addresses are printed during autoconfiguration.

9.3. Changes specific to individual drivers

if_acc.c An additional word was added to the input buffer to allow space for the end-of-message bit on a maximum-sized message without segmentation. This avoids a hardware problem that sometimes causes the next packet to be concatenated with the end-of-message segment.

if_ddn.c A new driver from ACC for the ACC DDN Standard mode X.25 IMP interface.

if_de.c A new driver for the DEC DEUNA 10 Mb/s Ethernet controller. The hardware is reset when *ifconfig*ed down and reinitialized when marked up again.

if_dmc.c The DMC-11/DMR-11 driver has been made much more robust. It now uses multiple transmit and receive buffers. A link-layer encapsulation is used to indicate the type of the packet; this driver is thus incompatible with the 4.2BSD DMC driver. (The driver is, however, compatible with current ULTRIX drivers.)

if_ec.c The handler for the 3Com 10 Mb/s Ethernet controller is now able to support multiple units. The address of the UNIBUS memory is taken from the flags in the configuration file; note that address 0 is still the default. The UNIBUS memory is configured in a separate memory-probe routine that is called during autoconfiguration and after a UNIBUS reset. This allows the 3Com interface reset to work correctly. The collision backoff algorithm was corrected so that the maximum backoff is within the specification, rather than waiting seconds after numerous collisions. The private *ecget* and *ecput* routines were modified to correspond with the *if_uba* routines. The hardware is reset when *ifconfig*ed down and reinitialized when marked up again.

if_en.c The 3 Mb/s Experimental Ethernet driver now supports NS IDP packets, using a simple algorithmic conversion of host to Ethernet addresses. The *enswab* function was corrected.

if_ex.c A new driver for the Excelan 204 10 Mb/s Ethernet controller, used as a link-layer interface.

if_hdh.c A new driver for the ACC HDH IMP interface.

if_hy.c A new version of the Hyperchannel driver from Tektronix was installed. It is untested with 4.3BSD.

if_il.c The Interlan 1010 and 1010A driver now resets the interface and checks the result of hardware diagnostics when initializing the board. The hardware is reset when *ifconfig*ed down and reinitialized when marked up again.

if_ix.c A new driver for using the Interlan NP100 10 Mb/s Ethernet controller as a link-level interface.

if_uba.c In addition to the major changes in UNIBUS support functions, there were several bug fixes made. Interfaces with no link-level header are set up properly. A variable was reused incorrectly in *if_wubaput*, and this has been corrected.

if_vv.c The driver for the Proteon proNET has been reworked in several areas. The elaborate error handling code had several problems and was simplified considerably. The driver includes support for both the 10 Mb/s and 80 Mb/s rings. The byte ordering of the trailer fields was corrected; this makes the trailer format incompatible with the 4.2BSD driver.

10. VAX MASSBUS device drivers

This section documents the modifications in the drivers for devices on the VAX MASSBUS, with sources in */sys/vaxmba*, as well as general changes made to all disk and tape drivers.

10.1. General changes in disk drivers

Most of the disk drivers' strategy routines were changed to report an end-of-file when attempting to read the first block after the end of a partition. Distinct errors are returned for nonexistent drives, blocks out of range, and hard I/O errors. The *dkblock* and *dkunit* macros once used to support disk interleaving were removed, as interleaving makes no sense with the current file system organization. Messages for recoverable errors, such as soft ECC's, are now handled by *log* instead of *printf*.

10.2. General changes in tape drivers

The open functions in the tape drivers now return sensible errors if a drive is in use. They save a pointer to the user's terminal when opened, so that error messages from interrupt level may be printed on the user's terminal using *tprintf*.

10.3. Modifications to individual MASSBUS device drivers

hp.c Error recovery in the MASSBUS disk driver is considerably better now than it was. The driver deals with multiple errors in the same transfer much more gracefully. Earlier versions could go into an endless loop correcting one error, then retrying the transfer from the beginning when a second error was encountered. The driver now restarts with the first sector not yet successfully transferred. ECC correction is now possible on bad-sector replacements. The correct sector number is now printed in most error messages. The code to decide whether to initiate a data transfer or whether to do a search was corrected, and the *sdist/rdist* parameters were split into three parameters for each drive: the minimum and maximum rotational distances from the desired sector between which to start a transfer, and the number of sectors to allow after a search before the desired sector. The values chosen for these parameters are probably still not optimal.

There were races when doing a retry on one drive that continued with a repositioning command (recal or seek) and when then beginning a data transfer on another drive. These were corrected by using a distinguished return value, MBD_REPOSITION, from *hpdtint* to change the controller state when reverting to positioning operations during a recovery. The remaining steps in the recovery are then managed by *hpustart*. Offset commands were previously done under interrupt control, but only on the same retries as recals (every eighth retry starting with the fourth). They are now done on each read retry after the 16th and are done by busy-waiting to avoid the race described above. The tests in the error decoding section of the interrupt handler were rearranged for clarity and to simplify the tests for special conditions such as format operations. The *hpdtint* times out if the drive does not become ready after an interrupt rather than hanging at high priority. When forwarding bad sectors, *hpecc* correctly handles partial-sector transfers; prior versions would transfer a full sector, then continue with a negative byte count, encountering an invalid map register immediately thereafter. Partial-sector transfers are requested by the virtual memory system when swapping page tables.

mba.c The top level MASSBUS driver supports the new return code from data-transfer interrupts that indicate a return to positioning commands before restarting a data transfer. It is capable of restarting a transfer after partial completion and adjusting the starting address and byte count according to the amount remaining. It has also been modified to support data transfers in reverse, required for proper error recovery on the TU78. *Mbustart* does not check drives to see that they are present, as dual-ported disks may appear to have a type of zero if the other port is using the disk; in this case, the disk unit start will return MBU_BUSY.

mt.c The TU78 driver has been extensively modified and tested to do better error recovery and to support additional operations.

11. VAX UNIBUS device drivers

This section includes changes in device drivers for UNIBUS peripherals other than network interfaces. Modifications common to all of the disk and tape drivers are listed in the previous section on MASSBUS drivers. Many of the UNIBUS drivers were missing null terminations on their lists of standard addresses; this has been corrected.

11.1. Changes in terminal multiplexor handling

There are numerous changes that were made uniformly in each of the drivers for UNIBUS terminal multiplexors (DH11, DHU11, DMF32, DMZ32, DZ11 and DZ32). The DMA terminal boards on the same UNIBUS share map registers to map the *clists* to UNIBUS address space. The initialization of *tty*s at open and changes from *ioctl*s have been made uniform; the default speed is 9600 baud. Hardware parameters are changed when local modes change; these include LLITOUT and the new LPASS8 options for 8-bit output and input respectively. The code conditional on PORTSELECTOR to accept characters while or before carrier is recognized is the same in all drivers. The processing done for carrier transitions was line discipline-specific, and has been moved into the standard *tty* code; it is called through the previously-unused *l_modem* entry to the line discipline. This routine's return is used to decide whether to drop DTR. DTR is asserted on lines regardless of the state of the software carrier flag. The drivers for hardware without silo timeouts (DH11, DZ11) dynamically switch between use of the silo during periods of high input and per-character interrupts when input is slow. The timer routines schedule themselves via timeouts and are no longer called directly from the *softclock* interrupt. The timeout runs once per second unless silos are enabled. Hardware faults such as nonexistent memory errors and silo overflows use *log* instead of *printf* to avoid blocking the system at interrupt level.

11.2. Changes in individual drivers

dmf.c The use of the parallel printer port on the DMF32 is now supported. Autoconfiguration of the DMF includes a test for the sections of the DMF that are present; if only the asynchronous serial ports or parallel printer ports are present, the number of interrupt vectors used is reduced to the minimum number. The common code for the DMF and DMZ drivers was moved to *dmfdmz.c*. Output is done by DMA. The Emulex DMF emulator should work with this driver, despite the incorrect update of the bus address register with odd byte counts. Flow control should work properly with DMA or silo output.

dmfdmz.c This file contains common code for the DMF and DMZ drivers.

dmz.c This is a new device driver for the DMZ32 terminal multiplexor.

idc.c The ECC code for the Integral Disk Controller on the VAX 11/730 was corrected.

kgclock.c The profiling clock using a DL11 serial interface can be disabled by patching a global variable in the load image before booting or in memory while running. It may thus be used for a profiling run and then turned off. The *probe* routine returns the correct value now.

lp.c A fix was made so that slow printers complete printing after device close. The *spl*'s were cleaned up.

ps.c The handler for the E & S Picture System 2 has substantial changes to fix refresh problems and clean up the code.

rk.c Missing entries in the RK07 size table were added.

rl.c A missing partition was added to the RL02 driver. Drives that aren't spun up during autoconfiguration are now discovered.

rx.c It is no longer possible to leave a floppy drive locked if no floppy is present at open. Incorrect open counts were corrected.

tm.c Hacks were added for density selection on Aviv triple-density controllers.

tmscp.c This is a new driver for tape controllers using the Tape Mass Storage Control Protocol such as the TU81.

ts.c Adjustment for odd byte addresses when using a buffered data path was incorrect and has been fixed.

uba.c The UBA_NEED16 flag is tested, and unusable map registers are not allocated for 16-bit addressing devices. Optimizations were made to improve code generation in *ubasetup*. Zero-vector interrupts on the DW780 now cause resets only when they occur at an unacceptably high rate; this is appreciated by the users who happen to be on the dialups at the time of the 250000th passive release since boot time. UNIBUS memory is now configured separately from devices during autoconfiguration by *ubameminit*, and this process is repeated after a UNIBUS reset. Devices that consist of UNIBUS memory only may be configured more easily. On a DW780, any map registers made useless by UNIBUS memory above or near them are discarded.

ubareg.h Definitions were added to include the VAX8600.

ubavar.h Modifications to the *uba_hd* structure allow zero vectors and UNIBUS memory allocation to be handled more sensibly. The *uba_driver* has a new entry for configuration of UNIBUS memory. This routine may probe for UNIBUS memory, and if no further configuration is required may signify the completion of device configuration. A macro was added to extract the UNIBUS address from the value returned by *ubasetup* and *uballoc*.

uda.c This driver is considerably more robust than the one released with 4.2BSD. It configures the drive types so that each type may use its own partition tables. The partitions in the tables as distributed are much more useful, but are mostly incompatible with the previously released driver; a configuration option, RACOMPAT, provides a combination of new and old filesystems for use during conversion. The buffered-data-path handling has been fixed. A dump routine was added.

up.c Entries were added for the Fujitsu Eagle (2351) in 48-sector mode on an Emulex SC31 controller.

vs.c This is a driver for the VS100 display on the UNIBUS.

12. Bootstrap and standalone utilities

The standalone routines in **/sys/stand** and **/sys/mdec** have received some work. The bootstrap code is now capable of booting from drives other than drive 0. The device type passed from level to level during the bootstrap operation now encodes the device type, partition number, unit number, and MASSBUS or UNIBUS adaptor number (one byte for each field, from least significant to most significant). The bootstrap is much faster, as the standalone *read* operation uses raw I/O when possible.

The formatter has been much improved. It deals with skip-sector devices correctly; the previous version tested for skip-sector capability incorrectly, and thus never dealt with it. The formatter is capable of formatting sections of the disk, track by track, and can run a variable number of passes. The error retry logic in the standalone disk drivers was corrected and parameterized so that the formatter may disable most corrections.