Changes to the Kernel in 4.2BSD

July 25, 1983

Samuel J. Leffler

Computer Systems Research Group
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, California 94720
(415) 642-7780

This document summarizes the changes to the kernel between the September 1981 4.1BSD release and the July 1983 4.2BSD distribution. The information is presented in both overall terms (e.g. organizational changes), and as specific comments about individual files. See the source code itself for more details.

The system has undergone too many changes to detail everything. Instead the major areas of change will pointed out, followed by a brief description of the contents of files present in the 4.1BSD release. Where important changes and/or bug fixes were applied they are described. The networking support is not discussed in this document, refer to ‘‘4.2BSD Networking Implementation Notes’’ for a discussion of the internal structure of the network facilities.

Major changes include:

- organizational changes to isolate VAX specific portions of the system
- changes to support the new file system organization
- changes to support the new interprocess communication facilities
- changes for the new networking support; in particular, the DARPA standard Internet protocols TCP, UDP, IP, and ICMP, and the *network interface drivers* which provide hardware support
- changes for the new signal facilities
- changes for the new time and interval timer facilities
- changes to eliminate references to global variables; in particular, the global variables *u.u_base*, *u.u_offset*, *u.u_segflg*, and *u.u_count* have been almost completely replaced by *uio* structures which are passed by reference; the *u.u_error* variable has not been completely purged from low level portions of the system, but is in many places now returned as a function value; the *uio* changes were necessitated by the new scatter-gather i/o facilities
- changes for the new disk quota facilities
- changes for more flexible configuration of the disk space used for paging and swapping

1. Carrying over local software

With the massive changes made to the system, both in organization and in content, it may take some time to understand how to carry over local software. The majority of this document is devoted to describing the contents of each important source file in the system. If you have local software other than device drivers to incorporate in the system you should first read this document completely, then study the source code to more fully understand the changes as they affect you.

Locally written device drivers will need to be converted to work in the new system. The changes required of device drivers are:

1) The calling convention for the driver *ioctl* routine has changed. Any data copied in or out of the system is now done at the highest level inside *ioctl* (). The third parameter to the driver *ioctl* routine is a data buffer passed by reference. Values to be returned by a driver must be copied into the associated buffer from which the system then copies them into the user address space.

2) The *read*, *write*, and *ioctl* entry points in device drivers must return 0 or an error code from <*errno.h*>.

3) The *read* and *write* entry points should no longer reference global variables out of the user area. A new *uio* parameter is passed to these routines which should, in turn, be passed to the *physio* () routine if the driver supports raw i/o.

4) Disk drivers which are to support swapping/paging must have a new routine which returns the size, in sectors, of a disk partition. This value is used in calculating the size of swapping/paging areas at boot time.

5) Code which previously used the *iomove*, *passc*, or *cpass* routines will have to be modified to use the new *uiomove*, *ureadc*, and *uwritec* routines. The new routines all use a *uio* structure to communicate the i/o base, offset, count, and segflag values previously passed globally in the user area.

6) Include files have been rearranged and new ones have been created. Common machine-dependent files such as *mtpr.h*, *pte.h*, *reg.h*, and *psl.h* are no longer in the ‘‘h’’ directory; see below under organizational changes.

7) The handling of UNIBUS resets has changed. The reset routine should no longer deallocate UNIBUS resources allocated to pending i/o requests (this is done in the *ubareset* routine). For most drivers this means the reset routine simply needs to invalidate any *ub_info* values stored in local data structures to insure new UNIBUS resources will be allocated the next time the ‘‘device start’’ routine is entered.

2. Organizational changes

The directory organization and file names are very different from 4.1BSD. The new directory layout breaks machine-specific and network-specific portions of the system out into separate directories. A new file, *machine* is a symbolic link to a directory for the target machine, e.g. *vax*. This allows a single set of sources to be shared between multiple machine types (by including header files as ‘‘../machine/file’’). The directory naming conventions, as they relate to the network support, are intended to allow expansion in supporting multiple ‘‘protocol families’’. The following directories comprise the system sources for the VAX:

/sys/h	machine independent include files
/sys/sys	machine independent system source files
/sys/conf	site configuration files and basic templates
/sys/net	network independent, but network related code
/sys/netinet	DARPA Internet code
/sys/netimp	IMP support code
/sys/netpup	PUP-1 support code
/sys/vax	VAX specific mainline code
/sys/vaxif	VAX network interface code
/sys/vaxmba	VAX MASSBUS device drivers and related code
/sys/vaxuba	VAX UNIBUS device drivers and related code

Files indicated as *machine independent* are shared among 4.2BSD systems running on the VAX and Motorola 68010. Files indicated as *machine dependent* are located in directories indicative of the machine on which they are used; the 4.2BSD release from Berkeley contains support only for the VAX. Files marked *network independent* form the ‘‘core’’ of the networking subsystem, and are shared among all network software; the 4.2BSD release from Berkeley contains complete support only for the DARPA Internet protocols IP, TCP, UDP, and ICMP.

3. Bug fixes and changes

This section contains a brief description of each file which is not part of the network subsystem, and also indicates important changes and bug fixes applied to the source code contained in the file.

3.1. /sys/h

Files residing here are intended to be machine independent. Consequently, the header files for device drivers which were present in this directory in 4.1BSD have been moved to other directories; e.g. /sys/vaxuba. Many files which had been duplicated in /usr/include are now present only in /sys/h. Further, the 4.1BSD /usr/include/sys directory is now normally a symbolic link to this directory. By having only a single copy of these files the ‘‘multiple update’’ problem no longer occurs. (It is still possible to have /usr/include/sys be a copy of the /sys/h for sites where it is not feasible to allow the general user community access to the system source code.)

The following files are new to /sys/h in 4.2BSD:

domain.h describes the internal structure of a communications domain; part of the new ipc facilities

errno.h had previously been only in /usr/include; the file /usr/include/errno.h is now a symbolic link to this file

fs.h replaces the old filsys.h description of the file system organization

gprof.h describes various data structures used in profiling the kernel; see *gprof*(1) for details

kernel.h is an offshoot of systm.h and param.h; contains constants and definitions related to the logical UNIX ‘‘kernel’’

mbuf.h describes the memory managment support used mostly by the network; see ‘‘4.2BSD Networking Implementation Notes’’ for more information

mman.h contains definitions for planned changes to the memory management facilities (not implemented in 4.2BSD)

nami.h defines various structures and manifest constants used in conjunctions with the *namei* routine (part of this file reflects future plans for changes to *namei* rather than current use)

protosw.h contains a description of the protocol switch table and related manifest constants and data structures use in communicating with routines located in the table

quota.h contains definitions related to the new disk quota facilities

resource.h contains definitions used in the *getrusage*, *getrlimit*, and *getpriority* system calls (among others)

socket.h contains user-visible definitions related to the new socket ipc facilities

socketvar.h contains implementation definitions for the socket ipc facilities

ttychars.h contains definitions related to tty character handling; in particular, manifest constants for the system standard erase, kill, interrupt, quit, etc. characters are stored here (all the appropriate user programs use these manifest definitions)

ttydev.h contains definitions related to hardware specific portions of tty handling (such as baud rates); to be expanded in the future

uio.h contains definitions for users wishing to use the new scatter-gather i/o facilities; also contains the kernel *uio* structure used in implementing scatter-gather i/o

un.h contains user-visible definitions related to the ‘‘unix’’ ipc domain

unpcb.h contains the definition of the protocol control block used in the ‘‘unix’’ ipc domain

wait.h contains definitions used in the *wait* and *wait3*(2) system calls; previously in /usr/include/wait.h

The following files have undergone significant change:

buf.h reflects the changes made to the buffer cache for the new file system organization – buffers are variable sized with pages allocated to buffers on demand from a pool of pages dedicated to the buffer cache; one new structure member has been added to eliminate overloading of a commonly unreferenced structure member; a new flag B_CALL, when set, causes the function *b_iodone* to be called when i/o completes on a buffer (this is used to wakeup the pageout daemon); macros have been added for manipulating the buffer queues, these replace the previous subroutines used to insert and delete buffers from the queues

conf.h reflects changes made in the handling of swap space and changes made for the new *select* (2) system call; the block device table has a new member, *d_psize*, which returns the size of a disk partition, in sectors, given a major/minor value; the character device table has a new member, *d_select*, which is passed a *dev_t* value and an FREAD (FWRITE) flag and returns 1 when data may be read (written), and 0 otherwise; the *swdevt* structure now includes the size, in sectors, of a swap partition

dir.h is completely different since directory entries are now variable length; definitions for the user level interface routines described in *directory* (3) are also present

file.h has a very different *file* structure definition and definitions for the new *open* and *flock* system calls; symbolic definitions for many constants commonly supplied to *access* and *lseek*, are also present

inode.h reflects the new hashed cacheing scheme as well additions made to the on-disk and in-core inodes; on-disk inodes now contain a count of the actual number of disk blocks allocated a file (used mostly by the disk quota facilities), larger time stamps (for planned changes), more direct block pointers, and room for future growth; in-core inodes have new fields for the advisory locking facilities, a back pointer to the file system super block information (to eliminate lookups), and a pointer to a structure used in implementing disk quotas.

ioctl.h has all request codes constructed from _IO, _IOR, _IOW, and _IOWR macros which encode whether the request requires data copied in, out, or in and out of the kernel address space; the size of the data parameter (in bytes) is also encoded in the request, allowing the *ioctl* () routine to perform all user-kernel address space copies

mount.h the *mount* structure has a new member used in the disk quota facilities

param.h has had numerous items deleted from it; in particular, many definitions logically part of the ‘‘kernel’’ have been moved to kernel.h, and machine-dependent values and definitions are now found in param.h files located in machine/param.h; contains a manifest constant, NGROUPS, which defines the maximum size of the group access list

proc.h has changed extensively as a result of the new signals, the different resource usage structure, the disk quotas, and the new timers; in addition, new members are present to simplify searching the process tree for siblings; the SDLYU and SDETACH bits are gone, the former is replaced by a second parameter to *pagein*, the latter is no longer needed due to changes in the handling of open’s on /dev/tty by processes which have had their controlling terminal revoked with *vhangup*

signal.h reflects the new signal facilities; several new signals have been added: SIGIO for signal driven i/o; SIGURG for notification when an urgent condition arises; and SIGPROF and SIGVTALRM for the new timer facilities; structures used in the *sigvec* (2) and *sigstack* (2) system calls, as well as signal handler invocations are defined here

stat.h has been updated to reflect the changes to the inode structure; in addition a new field *st_blksize* contains an ‘‘optimal blocking factor’’ for performing i/o (for files this is the block size of the underlying file system)

systm.h has been trimmed back a bit as various items were moved to kernel.h

time.h contains the definitions for the new time and interval timer facilities; time zone definitions for the half dozen time zones understood by the system are also included here

tty.h reflects changes made to the internal structure of the terminal handler; the ‘‘local’’ structures have been merged into the standard flags and character definitions though the user interface is virtually identical to that of 4.1BSD; the TTYHOG value has been changed from 256 to 255 to account for a counting problem in the terminal handler on input buffer overflow

user.h has been extensively modified; members have been grouped and categorized to reflect the ‘‘4.2BSD System Manual’’ presentation; new members have been added and existing members changed to reflect: the new groups facilities, changes to resource accounting and limiting, new timer facilities, and new signal facilities

vmmac.h has had many macro definitions changed to eliminate assumptions about the hardware virtual memory support; in particular, the stack and user area page table maps are no longer assumed to be adjacent or mapped by a single page table base register

vmparam.h now includes machine-dependent definitions from a file machine/vmparam.h.

vmsystm.h has had several machine-dependent definitions moved to machine/vmparam.h

3.2. /sys/sys

This directory contains the ‘‘mainstream’’ kernel code. Files in this directory are intended to be shared between 4.2BSD implementations on all machines. As there is little correspondence between the current files in this directory and those which were present in 4.1BSD a general overview of each files’s contents will be presented rather than a file-by-file comparison.

Files in the *sys* directory are named with prefixes which indicate their placement in the internal system layering. The following table summarizes these naming conventions.

init_	system initialization
kern_	kernel (authentication, process management, etc.)
quota_	disk quotas
sys_	system calls and similar
tty_	terminal handling
ufs_	file system
uipc_	interprocess communication
vm_	virtual memory

3.2.1. Initialization code

init_main.c contains system startup code

init_sysent.c contains the definition of the *sysent* table − the table of system calls supported by 4.2BSD

3.2.2. Kernel-level support

kern_acct.c contains code used in per-process accounting

kern_clock.c contains code for clock processing; work was done here to minimize time spent in the *hardclock* routine; support for kernel profiling and statistics collection from an alternate clock source have been added; a bug which caused the system to lose time has been fixed; the code which drained terminal multiplexor silos has been made the default mode of operation and moved to locore.s

kern_descrip.c contains code for management of descriptors; descriptor related system calls such as *dup* and *close* (the upper-most levels) are present here

kern_exec.c contains code for the *exec* system call

kern_exit.c contains code for the *exit* system call

kern_fork.c contains code for the *fork* (and *vfork*) system call

kern_mman.c contains code for memory management related calls; the contents of this file is expected to change when the revamped memory management facilities are added to the system

kern_proc.c contains code related to process management; in particular, support routines for process groups

kern_prot.c contains code related to access control and protection; the notions of user ID, group ID, and the group access list are implemented here

kern_resource.c code related to resource accounting and limits; the *getrusage* and ‘‘get’’ and ‘‘set’’ resource limit system calls are found here

kern_sig.c the signal facilities; in particular, kernel level routines for posting and processing signals

kern_subr.c support routines for manipulating the *uio* structure: *uiomove*, *ureadc*, and *uwritec*

kern_synch.c code related to process synchonization and scheduling: *sleep* and *wakeup* among others

kern_time.c code related to processing time; the handling of interval timers and time of day

kern_xxx.c miscellaneous system facilities and code for supporting 4.1BSD compatibility mode (kernel level)

3.2.3. Disk quotas

quota_kern.c ‘‘kernel’’ of disk quota suppport

quota_subr.c miscellaneous support routines for disk quotas

quota_sys.c disk quota system call routines

quota_ufs.c portions of the disk quota facilities which interface to the file system routines

3.2.4. General subroutines

subr_mcount.c code used when profiling the kernel

subr_prf.c *printf* and friends; also, code related to handling of the diagnostic message buffer

subr_rmap.c subroutines which manage resource maps

subr_xxx.c miscellaneous routines and code for routines implemented with special VAX instructions, e.g. bcopy

3.2.5. System level support

sys_generic.c code for the upper-most levels of the ‘‘generic’’ system calls: *read*, *write*, *ioctl*, and *select*; a ‘‘must read’’ file for the system guru trying to shake out 4.1BSD bad habits

sys_inode.c code supporting the ‘‘generic’’ system calls of sys_generic.c as they apply to inodes; the guts of the byte stream file i/o interface

sys_process.c code related to process debugging: *ptrace* and its support routine *procxmt*; this file is expected to change as better process debugging facilities are developed

sys_socket.c code supporting the ‘‘generic’’ system calls of sys_generic.c as they apply to sockets

3.2.6. Terminal handling

tty.c the terminal handler proper; both 4.1BSD and version 7 terminal interfaces have been merged into a single set of routines which are selected as line disciplines; a bug which caused new line delays past column 127 to be calculated incorrectly has been fixed; the high water marks for terminals running in tandem mode at 19.2 or 38.4 kilobaud have been upped

tty_bk.c the old Berknet line discipline (defunct)

tty_conf.c initialized data structures related to terminal handling;

tty_pty.c support for pseudo-terminals; actually two device drivers in one; additions over 4.1BSD pseudo-terminals include a simple ‘‘packet protocol’’ used to support flow-control and output flushing on interrupt, as well as a ‘‘transparent’’ mode used in programs such as emacs

tty_subr.c c-list support routines

tty_tb.c two line disciplines for supporting RS232 interfaces to Genisco and Hitachi tablets

tty_tty.c trivial support routines for ‘‘/dev/tty’’

3.2.7. File system support

ufs_alloc.c code which handles allocation and deallocation of file system related resources: disk blocks, on-disk inodes, etc.

ufs_bio.c block i/o support; the buffer cache proper; see description of buf.h and ‘‘A Fast File System for UNIX’’ for information

ufs_bmap.c code which handles logical file system to logical disk block number mapping; understands structure of indirect blocks and files with holes; handles automatic extension of files on write

ufs_dsort.c sort routine implementing prioritized seek sort algorithm for disk i/o operations

ufs_fio.c code handling file system specific issues of access control and protection

ufs_inode.c inode management routines; in-core inodes are now hashed and cached; inode synchronization has been revamped since 4.1BSD to eliminate race conditions present in 4.1

ufs_mount.c code related to demountable file systems

ufs_nami.c the *namei* routine (and related support routines) – the routine that maps pathnames to inode numbers

ufs_subr.c miscellaneous subroutines: this code is shared with certain user programs such as *fsck* (8); for a good time look at the *bufstats* routine in this file

ufs_syscalls.c file system related system calls, everything from *open* to *unlink*; many new system calls are found here: *rename, mkdir, rmdir, truncate,* etc.

ufs_tables.c static tables used in block and fragment accounting; this file is shared with user programs such as *fsck* (8)

ufs_xxx.c miscellaneous routines and 4.1BSD compatibility code; all of the code which still understands the old inode format is in here

3.2.8. Interprocess communication

uipc_domain.c code implementing the ‘‘communication domain’’ concept; this file must be augmented to incorporate new domains

uipc_mbuf.c memory management routines for the ipc and network facilities; refer to the document ‘‘4.2BSD Networking Implementation Notes’’ for a detailed description of the routines in this file

uipc_pipe.c leftover code for connecting two sockets into a pipe; actually a special case of the code for the *socketpair* system call

uipc_proto.c UNIX ipc communication domain configuration definitions; contains UNIX domain data structure initialization

uipc_socket.c top level socket support routines; these routines handle the interface to the protocol request routines, move data between user address space and socket data queues, understand the majority of the logic in process synchronization as it relates to the ipc facilities

uipc_socket2.c lower level socket support routines; provide nitty gritty bit twiddling of socket data structures; manage placement of data on socket data queues

uipc_syscalls.c user interface code to ipc system calls: *socket*, *bind*, *connect*, *accept*, etc.; concerned exclusively with system call argument passing and validation

uipc_usrreq.c UNIX ipc domain support; user request routine and supporting utility routines

3.2.9. Virtual memory support

The code in the virtual memory subsystem has changed very little from 4.1BSD; changes made in these files were either to gain portability, handle the new swap space configuration scheme, or fix bugs.

vm_drum.c code for the management of disk space used in paging and swapping

vm_mem.c management of physical memory; the ‘‘core map’’ is implemented here as well as the routines which lock down pages for physical i/o (the latter will have to change when the memory management facilities are modified to support sharing of pages); a sign extension bug on block numbers extracted from the core map has been fixed (this caused the system to crash with certain disk partition layouts on RA81 disks)

vm_mon.c support for virtual memory monitoring; code in this file is included in the system only if the PGINPROF and/or TRACE options are configured

vm_page.c the code which handles and processes page faults: *pagein*; race conditions in accessing pages in transit and requests to lock pages for raw i/o have been fixed in this code; a major path through *pagein* whose sole purpose was to implement the software simulated reference bit has been ‘‘parallel coded’’ in assembly language (this appears to decrease system time by at least 5% when a system is paging heavily); *pagein* now has a second parameter indicating if the page to be faulted in should be left locked (this eliminated the need for the SDLYU flag in the *proc* structure)

vm_proc.c mainly code to manage virtual memory allocation during process creation and destruction (the virtual memory equivalent of ‘‘passing the buck’’ is done here).

vm_pt.c code for manipulating process page tables; knowledge of the user area is found here as it relates to the user address space page tables

vm_sched.c the code for process 0, the scheduler, lives here; other routines which monitor and meter virtual memory activity (used in implementing high level scheduling policies) also are present; this code has been better parameterized to isolate machine-dependent heuristics used in the scheduling policies

vm_subr.c miscellaneous routines: some for manipulating accessability of virtual memory, others for mapping virtual addresses to logical segments (text, data, stack)

vm_sw.c indirect driver for interleaved, multi-controller, paging area; modified to support interleaved partitions of different sizes

vm_swap.c code to handle process related issues of swapping

vm_swp.c code to handle swap i/o

vm_text.c code to handle shared text segments – the ‘‘text’’ table

3.3. /sys/conf

This directory contains files used in configuring systems. The format of configuration files has changed slightly; it is described completely in a new document ‘‘Building 4.2BSD UNIX Systems with Config’’. Several new files exist for use by the *config*(8) program, and several old files have had their meaning changed slightly.

LINT a new configuration file for use in linting kernels

devices.vax maps block device names to major device numbers (on the VAX)

files now has only files containing machine-independent code

files.*xxx* (where *xxx* is a system name) optional, *xxx*-specific *files* files

files.vax new file describing files which contain machine-dependent code

makefile.vax makefile template specific to the VAX

param.c updated calculations of *ntext* and *nfile* to reflect network requirements; new quantities added for disk quotas

3.3.1. /sys/vaxuba

This directory contains UNIBUS device drivers and their related include files. The latter have moved from /sys/h in an effort to isolate machine-dependent portions of the system. The following device drivers were not present in the 4.1BSD release.

ad.c a driver for the Data Translation A/D converter

ik.c an Ikonas frame buffer graphics interphase; user access to the device is implemented by mapping the device registers directly into the virtual address space of a user (the routines to map memory are included in uba.c only if an Ikonas is configured in the system)

kgclock.c a driver for a DL11-W or KL11-W used as an auxiliary real-time clock source for kernel profiling and/or statistics gathering; if this device is present, the system will automatically collect its i/o statistics (and if profiling, pc samples) off the secondary clock; very useful in kernel profiling as the second clock source eliminates most of the statistical anomalies and shows the true time spent in the clock routine

ps.c driver for an Evans and Sutherland Picture System 2

rl.c driver for RL11 controller with RL02 cartridge disks; does not support RL01 disks though it should only require additions to disk geometry and partition tables

rx.c driver for RX211 floppy disk controller; provides both block and character device interfaces; *ioctl* calls support floppy disk formatting and ‘‘deleted data mark’’ sensing and writing; makes a great paging device

ut.c driver for tape controllers which emulate a TU45 on the UNIBUS; in particular, the System Industries Model 9700 triple density tape drive

uu.c driver for dual UNIBUS TU58 cartridge tape cassettes accessed through a DL11 serial line; uses assembly language code in locore.s which provides pseudo-DMA on input (necessary to avoid data overruns); using this driver while the system runs multi-user degrades response severely (developed at Berkeley exclusively to produce distribution TU58 cassettes)

In addition to the above device drivers, many drivers present in 4.1BSD now sport corresponding include files which contain device register definitions. For example, the DH11 driver is now broken into three files: dh.c, dhreg.h, and dmreg.h.

The following drivers have been significantly modified, or had bugs fixed in them, since the 4.1BSD release:

dh.c changes to reflect the revised tty data organization

dmf.c a bug where device register accesses caused unwitting modification of certain status bits has been fixed; modem control has been fixed; a remnant of the DH11 include file which caused incorrect definitions for even/odd parity has been fixed; changes to reflect the revised tty data organization

dz.c now supports the DZ32; changes to reflect the revised tty data organization

lp.c now takes a non-zero flags value specified in the configuration file as the printer width (default is 132 columns); thus, to configure an 80 column printer, include ‘‘flags 80’’ in the device specification

rk.c a race condition has been fixed where a seek finishing on one drive appeared as an i/o transfer completeing on another (this bug actually was present in all UNIBUS disk drivers); changes for *uio* and swap space configuration

tm.c a typo which made the system crash with multiple slaves on a single controller has been fixed; an incorrect priority level change in the watchdog timer routine which caused the system to crash when a device operation timed out has been fixed; changes for *uio* processing of raw i/o

ts.c changes for *uio* processing of raw i/o

uba.c a new support routine for allocating UNIBUS memory for memory-mapped devices such as the 3Com Ethernet interface; the handling of UNIBUS resets has been changed, all UNIBUS resources are now reclaimed in the *ubareset* routine prior to calling individual device driver reset routines – this implies driver reset routines should no longer free up allocated UNIBUS resources; new routines for mapping UNIBUS memory into the virtual address space of a process have been added to support the Ikonas device driver; changes to fix the race condition described above in the RK07 device driver; processes awaiting UNIBUS map registers now sleep on a different event than those waiting for buffered data paths

uda.c the problem with multiplexing buffered data paths on an 11/750 has been fixed; a bug in the setup of the *ui_dk* field has been fixed; now properly defines the field indicating the disk transfer rate; changes for *uio* processing and swap space configuration

up.c now supports ECC correction and bad sector forwarding; significant changes have been made to make configuration of various disk drives simple (by probing the holding register and using the resultant value indicating the number of tracks on the disk); the race condition described under rk.c has been fixed; references to UNIBUS map registers are now done with longword instructions so the device driver does not cause the system to crash when an ECC or bad sector error occurs on a disk attached to a 730 UNIBUS; the upSDIST/upRDIST parameters which control the use of search and seek operations on controllers with multiple drives have been made drive dependent; a bug whereby the probe routine would belive certain non-existant drives were present has been fixed; changes for *uio* processing and swap space configuration

va.c has been rewritten to honor the software support for exclusive access to the UNIBUS so that the device may coexist on the same UNIBUS with RK07 disk drives; the driver now works with controllers which have a GO bit

3.3.2. /sys/vax

The following files are new in 4.2BSD:

crt0.ex edit script for creating a profiled kernel

frame.h copied from /usr/include

in_cksum.c checksum routine for the DARPA Internet protocols

param.h machine-dependent portion of /sys/h/param.h

pup_cksum.c checksum routine for PUP-I protocols

rsp.h protocol definitions for communicating with a TU58

sys_machdep.c machine-dependent portion of the ‘‘sys_*’’ files of /sys/sys

ufs_machdep.c machine-dependent portion of the ‘‘ufs_*’’ files of /sys/sys

vm_machdep.c machine-dependent portion of the ‘‘vm_*’’ files of /sys/sys

vmparam.h machine-dependent portion of /sys/h/vmparam.h

The following files have been modified for 4.2BSD:

Locore.c includes new definitions for linting the network and ipc code

asm.sed now massages *insque*, *remque*, and various routines which do byte swapping into assembly language

autoconf.c handles MASSBUS drives which come on-line after the initial autoconfiguration process; sizes and configures swap space at boot time in addition to calculating the swap area allocation parameters *dmtext*, *dmmax*, and *dmmin* (which were manifest constants in 4.1BSD); calculates the disk partition offset for system dumps at boot time to take into account variable sized swap areas; now uses the per-driver array of standard control status register addresses when probing for devices on the UNIBUS; now allows MASSBUS tapes and disks to be wildcarded across controllers

conf.c uses many ‘‘local’’ spaces for new and uncommon device drivers

genassym.c generates several new definitions for use in locore.s

locore.s includes code to vector software interrupts to protocol processing modules; assembly language assist routines for the console and UNIBUS TU58 cassette drives; a new routine, *Fastreclaim* is a fast coding of a major path through the *pagein* routine; copyin and copyout now handle greater than 64Kbyte data copies and return EFAULT on failure; understands the new signal trampoline code; now contains code for draining terminal multiplexor silos at clock time; a bug where a the translation buffer was sometimes being improperly flushed during a *resume* operation has been fixed

machdep.c a bug which caused memory errors to not be reported on 11/750’s has been fixed; has new code for handling the new signals; recovers from translation buffer parity fault machine checks apparently caused by substandard memory chips used in many 11/750’s; includes optional code to pinpoint bad memory chips on Trendata memory boards; the machine check routine now calls the *memerr* routine to print out the memory controller status registers in case the fault occurred because of a memory error

mem.c now has correct definitions to enable correctable memory error reporting on 11/750’s: DEC documentation incorrectly specifies use of the ICRD bit

pcb.h has changes related to the new signal trampoline code

swapgeneric.c supports more devices which can be used as a generic root device; interacts with the new swap configuration code to size the swap area properly when running a generic system; understands the special ‘‘swap on root’’ device syntax used when installing the system

trap.c can be compiled with a SYSCALLTRACE define to allow system calls to be traced when the variable *syscalltrace* is non-zero;

tu.c includes (limited) support for the TU58 console cassette on the 11/750, sufficient for use in single-user mode; supports the use of the MRSP ROM on the 11/750.

3.3.3. /sys/vaxmba

The following bug fixes and modifications have been applied to the MASSBUS device drivers:

hp.c a large number of disk drives attached to second vendor disk controllers are now automatically recognized at boot time by probing the holding register and using disk geometry information to decide what kind of drive is present; the hpSDIST/hpRDIST parameters that control seek and search operations on controllers with multiple drives have been made a per-drive parameter; a bug where the sector number reported on a hard error was off by one has been fixed; the error recovery code now searches the bad sector table when a header CRC error occurs; the error recovery code now handles bad sectors on tracks which also have skip sectors; a bug in the handling of ECC errors has been fixed; many separate driver data structures have been consolidated into the software carrier structure; the driver handles the ML-11 solid-state disk

mba.c now autoconfigures MASSBUS tapes and disks which ‘‘come on-line’’ after the initial boot

4. Standalone support

This section describes changes made to the standalone i/o facilities and the new methods used in system bootstrapping.

4.1. Disk formatting

A new disk formatting program has been developed for use with non-DEC UNIBUS and MASSBUS disk controllers. The *format*(8V) program has been tested mainly with disk drives attached to Emulex MASSBUS and UNIBUS disk controllers, but should operate with any controller which handles bad sector forwarding in an identical fashion to DEC RM03/RM05 or RM80 (but not RP06) disk controllers. The program runs standalone formatting disk headers and creating a bad sector table in the DEC standard 144 format.

4.2. Standalone i/o library

Changes to support more complex standalone i/o applications as well as changes for the new file system organization, have resulted in significant revisions to the standalone i/o library. Device drivers now support a new entry point for *ioctl* requests and library routines now return error codes a la the UNIX system calls. In addition, standalone i/o library routines now make many more internal consistency checks to verify data structures have not been corrupted by faulty device drivers and that i/o errors have not occurred when reading critical file system information. In conjunction with the new disk formatter, the *up* and *hp* standalone drivers have been rewritten to support ECC correction and bad sector handling. These drivers are used in bootstrapping from the console media on 11/780's and 11/730's thereby eliminating the requirement for error free root partitions on disks attached to *hp* and *up* controllers. Many bugs in the standalone tape drivers have been fixed.

4.3. System bootstrapping

On 11/780's and 11/730's, the console device is still used to load the ‘‘boot’’ program. This in turn loads the system image from the root file system.

The method by which the system bootstraps on 11/750's is different in 4.2BSD. The system is still bootstrapped from disk using a boot block in sector 0 of the root file system partition, but now this boot block simply reads in the next 7.5 kilobytes. The 7.5 kilobyte program is a version of the ‘‘/boot’’ program loaded only with the device driver required to read the ‘‘/boot’’ program from the root file system. The ‘‘/boot’’ program then reads in the system image, as done on 11/780's and 11/730's.

The additional level of bootstrap code was done to simplify the sector 0 boot programs and minimize the total amount of assembly language code which had to be maintained. It was also expected that 7.5 kilobytes would be sufficient to allow the new *hp* and *up* standalone drivers which support ECC correction and bad sector handling to be used. Unfortunately, the standalone system has not yet been trimmed down to allow the second level boot programs, loaded with the new drivers, to fit in the space provided. Sites which have Winchester disk drives with bad sectors in the root file system partition and which require this support should be able to trim the size of the second level boot program to make it fit.