

THIS VERSION WAS PRODUCED BY REVERTING THE SEVENTH  
EDITION KERNEL SOURCE CODE AND A PROGRAM WRITTEN TO  
GENERATE THE INDEX AND CROSS REFERENCE  
BY BRIAN S. WALDEN           WH 3A-327           AUGUST 1988

# UNIX OPERATING SYSTEM SOURCE CODE LEVEL SIX

This booklet has been produced for students at the University of New South Wales taking courses 6.602B and 6.657G.

It contains a specially edited selection of the UNIX Operating System source code, such as might be used on a typical PDP11/40 computer installation.

The UNIX Software System was written by K. Thompson and D. Ritchie of Bell Telephone Laboratories, Murray Hill, NJ. It has been made available to the University of New South Wales under a licence from the Western Electric Company.

J. Lions  
Department of Computer Science  
The University of New South Wales.  
June, 1977



6746 access	3472 getgid	2855 nosys	3439 setuid
6956 alloc	6181 getmdev	4999 notavil	8201 sgtty
0734 aretu:	3480 getpid	1771 nseg	3949 signal
1012 backup:	3413 getswit	6577 nulldev	2066 sleep
7040 badblock	3452 getuid	2864 nullsys	3595 smdate
4856 bawrite	4136 grow	5765 open	6086 smount
6585 bcopy	3420 gtime	5804 open1	1293 spl0:
4836 bdwrite	8165 gtty	6702 openi	1297 spl1:
5229 bflush	7067 ialloc	6791 owner	1302 spl4:
5055 binit	1284 idle:	2416 panic	1303 spl5:
6415 bmap	7134 ifree	6517 passc	1308 spl6:
4754 bread	7276 iget	8669 pcclose	1313 spl7:
4773 breada	6922 iinit	8763 pcleader	3614 ssig
4869 brelse	4899 incore	8648 pcopen	5979 sslep
4809 bwrite	0895 incupc:	8748 pcoutput	6028 stat
8274 canon	5018 iodone	8739 pcprint	6045 stat1
3538 chdir	6364 iomove	8682 pcread	3428 stime
3560 chmod	4982 iowait	8719 pcrint	4016 stop
3575 chown	7344 iput	8710 pcstart	8183 stty
8234 cinit	3991 issig	8701 pcwrite	0827 subyte:
0676 clearseg:	7414 itrunc	5259 physio	0826 suibyte:
3725 clock	7374 iupdat	7723 pipe	0860 suiword:
5846 close	3630 kill	7862 plock	6144 sumount
6643 closef	8055 klclose	2433 prdev	1739 sureg
6672 closei	8023 klopen	7882 prele	6811 suser
5038 clrbuf	8062 klread	2340 printf	0861 suword:
1244 copyin:	8078 klrint	2369 printn	5196 swap
1252 copyout:	8090 klsgtty	4204 procxmt	2178 swtch
0696 copyseg:	8066 klwrite	3667 profil	3486 sync
4094 core	8070 klxint	4043 psig	3845 timeout
6542 cpass	1393 ldiv:	3963 psignal	3656 times
5781 creat	5909 link	4164 ptrace	2693 trap
2447 deverror	8879 lpcanon	0967 putc:	2841 trap1
5096 devstart	8863 lpclose	2386 putchar	8535 ttread
0890 display:	8976 lpint	5731 rdwr	8486 ttrstrt
1319 dpadd:	8850 lpopen	5711 read	8505 ttstart
1327 dpcmp:	8986 lpoutput	6221 readi	8550 ttwrite
6069 dup	8967 lpstart	7758 readp	8333 ttyinput
1650 estabur	8870 lpwrite	0740 retu:	8373 ttyoutput
3020 exec	1401 lrem:	3205 rexit	8577 ttystty
3219 exit	1410 lshift:	5123 rhstart	7689 uchar
2268 expand	1550 main	5420 rkaddr	6824 ufalloc
6847 falloc	7455 maknode	5451 rkintr	3510 unlink
8252 flushtty	2528 malloc	5476 rkread	7201 update
3322 fork	5156 mapalloc	5440 rkstart	3270 wait
7000 free	5182 mapfree	5389 rkstrategy	2113 wakeup
6014 fstat	6326 max	5483 rkwrite	7477 wdir
0815 fubyte:	2556 mfree	0889 savfp:	8217 wflushtty
0814 fuibyte:	6339 min	0725 savu:	5720 write
0844 fuiword:	5952 mknod	3354 sbreak	6276 writei
0845 fuword:	9016 mmread	7679 schar	7805 writep
4921 getblk	9042 mmwrite	1940 sched	4433 xalloc
0930 getc:	7518 namei	5861 seek	4490 xccdec
5336 geterror	1826 newproc	3460 setgid	4398 xfree
6619 getf	3493 nice	2156 setpri	4368 xswap
7167 getfs	6566 nodev	2134 setrun	

File param.h	File prf.c	4136 grow	5861 seek	7679 schar
File systm.h	2340 printf	4164 ptrace	5909 lnk	7689 uchar
File seg.h	2369 printn	4204 procmxt	5952 mknod	File pipe.c
File proc.h	File text.h	File text.c	5979 sslp	7723 p1pe
File user.h	2416 panic	File text.c	File sys3.c	7758 readp
File low.s	2433 prdev	4368 xswap	6014 tsat	7805 wrtdep
File m40.s	2447 deverror	4398 xtree	6028 stat	7862 plck
File malloc.c	File malloc.c	4433 xalloc	6045 statl	7882 prele
File copyseg:	2528 malloc	4490 xccdec	6069 dup	File tty.h
File savu:	2556 mtree	File buf.h	6086 smount	File kl.c
File aretu:	File reg.h	File conf.h	6144 smount	8023 klopen
File retu:	File trap.c	File conf.c	6181 getmdev	8055 klclose
File ttybtye:	2693 trap	File rdwrt.c	6221 readl	8062 klread
File ttybtye:	2841 trapl	4754 bread	6221 readl	8066 klwrtite
File ttybtye:	2855 noys	4773 breada	6276 wrtlet	8070 klxkint
File subtye:	2864 nullsys	4809 bwrtite	6326 max	8078 klrint
File ttyword:	File syst.c	4836 bdwrtite	6339 min	8090 klsgtty
File ttyword:	File sys4.c	4856 bwrtite	6364 lomove	File tty.c
File exec	3020 exec	4869 brelse	File subr.c	8165 gtty
File suword:	3205 rexit	4899 incore	6415 bmap	8183 stty
File savfp:	3219 exlt	4921 getblk	6517 passc	8201 sgtty
File dispalay:	3270 wait	4982 lowait	6542 cpass	8217 wflushetty
File incupc:	3322 fork	4999 notavll	6566 nodev	8234 cmlt
File getc:	3354 sbreak	5018 lodone	6577 nulldev	8252 flushetty
Fileputc:	File sys4.c	5038 clrbuf	6585 bcopy	8274 canon
File backup:	3413 getswit	5055 binit	File flo.c	8333 ttyinput
File copyin:	3420 gtime	5096 devstart	6619 getf	8373 tlycountput
File idie:	3439 setuid	5123 rstart	6643 closef	8486 ttrstrt
File sp10:	3452 getuid	5156 mapalloc	6672 closef	8505 ttrstart
File sp11:	3460 segtd	5182 maptree	6702 openl	8535 tthead
File sp14:	3472 getgid	5196 swap	6746 access	8550 ttrwrtite
File sp15:	3480 getpid	5229 bflush	6791 owner	8577 ttystty
File sp16:	3486 sync	5259 physio	6811 suer	File pc.c
File sp17:	3493 nice	5336 geterror	6824 ufallloc	8648 pccopen
File dpadd:	3510 unlnk	File rk.c	6847 fallloc	8669 pcclose
File dpcomp:	3538 chdir	5389 rksrategy	File alloc.c	8682 pccread
File ldv:	3560 chmod	5420 rkaddr	6922 tinit	8701 pcwrtite
File lshft:	3575 chown	5440 rkstart	6956 alloc	8710 pcstart
File main.c	3614 sig	5451 rkintr	7000 tree	8719 pcrint
File main.h	3630 klll	5476 rkread	7040 badblock	8739 pcprint
File estabur	3656 times	File file.h	7067 fallloc	8748 pcountput
File sureg	3667 profll	File file.h	7134 tree	8763 pcleader
File nseg	File clock.c	File ino.h	7167 gets	File lp.c
File slp.c	3725 clock	File inode.h	7201 update	8850 lpcopen
File newproc	3845 timeout	File sys2.c	7276 lget	8870 lpcanon
File sched	File sig.c	5711 read	7344 lput	8879 lpcanon
File sleep	3949 signal	5720 wrtite	7374 lupdate	8967 lpcstart
File wakeup	3963 psignal	5731 rdwr	7414 ltrunc	8976 lprint
File setrun	3991 lssig	5765 open	7455 maknode	8986 lpcountput
File setprt	4016 stop	5781 creat	7477 wdir	File mem.c
File switch	4043 psig	5846 close	File namt.c	9016 mmread
File expand	4094 core	5846 close	7518 namef	9042 mmwrtite





	6052 6124 6125 6371	b_resid	4533 5322	chmod	2927 3560	cpass	6388 6542 8558 8705
	6437 6473 6491 6931	B_WANTED	4581 4876 4878 4879	chown	2928 3575		8874 9057
	6935 6974 7017 7098		4887 4942 4954 5030	cinit	1613 8234	cputype	0208 1459 1461 1571
	7174 7212 7220 7328		5166 5187 5203 5216	CINTR	7958 8344 8345		1655 1746 1756 5133
	7387 7427 7432 7433		5219 5296 5318 5321	CKILL	7956 8049		5162
	7636	b_wcount	4528 4762 4784 4794	cl	8637 8832	CQUIT	7957 8344
B_ASYNC	4584 4793 4820 4862		4818 5108 5137 5208	clearseg	0675 0676 1566 3134	CRDELAY	7976
	4887 4962 5027 5239		5310		3395 4155	creat	2920 5781
b_back	4524 4556 4967 4968	B_WRITE	4572 5486 6306 6373	clist	7908 7928 7929 7930	cret	1429 1430
	4970 4971 5062 5068		6386		8634 8643 8644	CRMOD	7970 8047 8342 8412
	5070 5080	b_xmem	4530 5110 5134 5139	clock	0569 0570 3725	csv	1419 1420
b_blkno	2454 4531 4908 4938		5173 5178 5211 5308	CLOCK1	1509 1601	CTLRDY	5374 5462
	4974 5209 5309 5402	c1	8881 8883 8885 8886	CLOCK2	1510 1603	ctype	8379 8424 8426 8440
	5428 6442 6450 6470		8887 8911 8915 8928	cloop	7542 7667		8441 8445 8452 8453
	6484 6498		8929 8930 8959	close	2918 5846		8468 8469 8472
B_BUSY	4576 4887 4941 4966	c2	8881 8890 8894 8898	CLOSED	8609 8653 8675	curpri	0222 2141 2165 2224
	5010 5072 5165 5169		8902 8906 8909	closef	3230 5854 6643	c_arg	0263 3770 3776 3866
	5202 5206 5219 5295	call	0555 0558 0561 0564	closei	6656 6672		3871
	5299 5321		0567 0570 0574 0577	clrbuf	5038 6982	c_cc	7910 8074 8223 8349
B_DELWRI	4586 4817 4823 4847		0752 0776 2669 2771	CMAPSIZ	0141 0203		8543 8544 8560
	4961 5237	call1	0762 0771	colp	8378 8400 8401 8402	c_cf	7911
b_dev	2453 4527 4819 4843	callo	0260 3727 3847		8404 8423 8429 8435	c_cl	7912
	4883 4908 4938 4973	callout	0265 3748 3750 3767		8436 8442 8443 8448	c_func	0264 3748 3751 3769
	5066 5207 5238 5300		3768 3773 3853		8454 8458 8459 8475		3770 3774 3855 3861
	5399 5429 5431	callp	2696 2754 2755 2761	com	5102 5109 5112 5114		3865 3870
B_DONE	4574 4759 4782 4790		2762 2765 2771		5115 5129 5138 5141	c_next	8141 8241
	4817 4847 4989 5026	CANBSIZ	0140 0202 8316		5142 5143	c_time	0262 3751 3753 3767
	5214 5315	canon	8274 8543	cont	7106 7110		3769 3775 3855 3856
b_error	4532	canonb	0202 8291 8300 8316	copsu	1245 1253 1264		3859 3864 3869
B_ERROR	4575 4817 4882 5220		8320	copyin	1243 1244 6374	data	1457
b_error	5311	CAP	8840 8884	copyout	1243 1252 1630 6376	dev	2433 2436 2693 2700
B_ERROR	5342	CARR_ON	7990 8046 8285 8541	copyseg	0695 0696 1915 2292		2702 2718 3725 4754
b_error	5343		8556		3380 3392 4152		4758 4763 4776 4778
B_ERROR	5403 5467 7323	cblock	8140 8141 8146 8149	core	4076 4094		4780 4781 4788 4789
b_flags	4522 4759 4761 4782		8237	coreaddr	5196 5210 5211		4799 4901 4905 4908
	4783 4790 4793 4816	cc	8635 8731 8743 8754	coremap	0203 1568 1896 1982		4921 4927 4931 4934
	4817 4847 4862 4876		8830 8981 8988		2278 2282 2293 3241		4938 4973 5229 5238
	4878 4879 4882 4887	ccc	8835 8910 8918 8935	count	4383 4497		5259 5300 5476 5479
	4941 4942 4954 4961		8937 8941 8942 8946		2668 2762 2765 5196		5483 5486 6676 6679
	4962 4966 4989 5010	ccp	8950 8954 8955 8962	cp	5208 6585 6592		6685 6689 6706 6709
	5024 5026 5027 5030		8236 8239 8240 8244		3025 3049 3061 3072		6716 6722 6956 6961
	5072 5111 5140 5172	cdevsw	8246 8247		3153 3161 3162 3186		6970 6973 6981 6988
	5186 5200 5237 5239		4635 4641 4669 6234		3187 4018 4021 4022		7000 7004 7008 7016
	5295 5296 5299 5315		6287 6685 6716 8213		4024 4026 4028 6048		7040 7048 7067 7072
	5318 5321 5342 5397		8238 8245		6052 6059 6367 6371		7078 7097 7104 7120
	5403 5467 7323	cdp	8238 8245		6372 6374 6376 6377		7134 7138 7167 7173
b_forw	4523 4555 4907 4937	CEOT	7955 8306		6390 6394 6924 6928		7178 7276 7286 7296
	4967 4968 4969 4971	CERASE	7954 8048		6931 6933 6935 6936		7314 7319 8023 8026
	4972 5062 5069 5070	cf	8636 8831		6937 6938 6939 6940		8030 8033 8039 8040
	5071 5079	cfree	8146 8239 8240		7417 7427 7428 7429		8042 8055 8057 8062
B_MAP	4579 5024 5172 5186	cfreelist	0928 0954 0955 0977		7431 7438 7523 7570		8063 8066 8067 8070
B_PHYS	4577 5206 5299 5397		0979 0986 0988 8149		7572 7573 7576 7577		8072 8078 8081 8090
B_READ	2034 2042 4573 4761		8241 8242		7645 7646 8237 8240		8093 8648 8669 8850
	4783 4793 4817 5111	chan	2066 2076 2089 2113		8241 8242		8863 9016 9021 9031
	5140 5479 6260		2118	cp1	7480 7483 7485		9042 9047 9064
B_RELOC	4583 4966	chdir	2924 3538	cp2	7480 7484 7485	devblk	5096 5106 5123 5135





fstat	2940 6014	hibyte	0180 3456 3476 3582		7285 7345 7521 8205	6673 6678 6702 6703
fubyte	0807 0815 3058 4225		8585 8593	inta	3921 4235 4254	6708 6749 6751 6754
	6550 7693	httab	4728 4844	integ	0175 2070 2095 2391	6758 6764 6769 6771
fuibyte	0809 0814 1564 4218	HUPCL	7966		3416 3852 3872 4885	6774 6793 6796 6798
	9034	HZ	0147 3797 3800		4892 5006 5011 8262	6799 6801 6802 6959
fuiword	0813 0844 1602 1604	IACC	5681 6232 6285 7382		8266	6974 6975 6976 7002
	2734 2754 2756 2766		7391 7462 7751	IO	0641	7017 7018 7019 7069
	4220	IALLOC	5620 5687	iodone	5018 5404 5471	7078 7079 7081 7082
fun	3845 3870	ialloc	7067 7459	iomove	6260 6306 6364	7085 7091 7098 7101
func	7518 7519 7532 7536	IALLOC	7463	iowait	4764 4800 4821 4982	7203 7212 7213 7214
	7574 7579 8510 8515	ialloc	7728	ip	3024 3034 3035 3041	7217 7218 7219 7220
fuword	0811 0845 0847 2758	IALLOC	7752		3090 3105 3130 3142	7223 7224 7225 7226
	2763 3052 4227 8188	icode	1516 1630		3171 3173 3174 3176	7227 7281 7284 7293
	8189 8190	idle	1283 1284 2220 2423		3177 3182 3183 3184	7294 7295 7296 7306
FWRITE	5518 5722 5793 5795	IENABLE	5092 5109 5138 5370		3185 3189 3190 3191	7307 7309 7319 7323
	5816 5829 5832 6656		7981 8051 8052 8615		3194 3512 3519 3520	7324 7328 7332 7414
	7746		8659 8663 8692 8732		3522 3529 3530 3534	7415 7420 7423 7424
f_count	1878 5510 5836 6079		8814 8858		3540 3543 3544 3546	7426 7430 7442 7443
	6655 6657 6855 6857	IEXEC	3041 3552 5631 5698		3549 3552 3555 3556	7457 7459 7460 7462
	7739		6764 6765 7563		3562 3564 3566 3569	7463 7464 7465 7466
f_flag	5509 5739 5746 5829	IFBLK	5624 5691 6100 6189		3570 3571 3577 3579	7467 7468 7477 7478
	5869 6649 6656 7746		6242 6297 6314 6688		3581 3582 3583 3584	7482 7725 7728 7729
	7748		6719 7421		4096 4101 4102 4105	7733 7741 7747 7749
f_inode	5511 5754 5755 5830	IFCHR	5623 5690 6100 6233		4106 4109 4110 4112	7750 7751 7752 7761
	5894 5895 6021 6650		6286 6314 6684 6713		4118 4124 4126 4399	7764 7768 7772 7775
	6656 7747 7749 7764		7421 8209		4405 4406 4410 4411	7776 7777 7778 7786
	7810 8208	IFDIR	3522 3546 5622 5689		4433 4434 4446 4454	7787 7789 7790 7797
f_offset	5512 5751 5752 5756		5818 5921 7559		4464 4470 5767 5770	7799 7807 7810 7815
	5889 5890 5901 5902	IFMT	3041 3522 3546 4110		5771 5774 5783 5786	7817 7825 7826 7835
	6858 6859 7772 7773		5621 5688 5818 5921		5787 5790 5791 5793	7836 7837 7838 7845
	7774 7796 7798		6189 6233 6242 6286		5795 5804 5805 5811	7848 7849 7850 7851
getblk	3040 3237 4758 4781		6297 6682 6711 7559		5911 5914 5915 5917	7852 7862 7863 7867
	4789 4921 6123 6304		8209		5921 5926 5935 5940	7882 7883 7887 8205
	6928 6981 7016 7216	ifree	7134 7355		5941 5942 5945 5954	8208 8209 8213
getc	0926 0930 8258 8259	iget	1616 1618 3519 7078		5958 5959 5966 5967	7280 7328 7331 7378
	8264 8292 8520 8544		7276 7534 7664		5969 5972 6030 6033	7387 7390 7392 7393
	8673 8688 8714 8971	iinit	1615 6922		6034 6036 6037 6045	7395 7397 7398
geterror	4824 4992 5323 5336	ILARG	5625 5692 6427 6444		6046 6050 6051 6052	7279 7329 7330 7331
getf	5736 5850 5866 6018		7425 7445		6053 6055 6089 6097	7378 7388 7389 7390
	6073 6619 8206	ILOCK	1617 1619 5679 5926		6098 6100 6121 6130	3939 4181 4182 4183
getfs	6754 6961 7004 7072		7224 7225 7287 7303		6131 6137 6147 6161	4184 4185 4186 4189
	7138 7167 7383		7316 7351 7868 7872		6162 6167 6168 6169	4190 4191 4192 4194
getgid	2959 3472		7888		6170 6172 6183 6186	4195 4209 4211 4212
getmdev	6093 6151 6181	IMOUNT	5682 6130 6168 7292		6187 6189 6191 6192	4213 4218 4220 4225
getpid	2932 3480	incore	4780 4788 4899		6194 6227 6229 6232	4227 4232 4235 4240
getswit	2950 3413	incupc	0894 0895 3791		6233 6234 6242 6243	4242 4247 4249 4254
getuid	2936 3452	IND	8844 8857 8936		6248 6250 6252 6255	4264 4266 4268 4273
gid	3462 3464 3465 3466	info	8142		6259 6282 6284 6285	4282
	3467	ino	7070 7077 7078 7095		6286 6287 6297 6298	IPCPRI 3914 4182 4190
GO	5095 5109 5138 5368		7100 7105 7107 7134		6300 6302 6312 6314	iput 3194 3232 3533 3534
	5461		7143 7276 7286 7297		6315 6316 6318 6415	3549 3554 3571 3584
grow	2813 4056 4136		7315 7319 7328		6416 6422 6427 6439	4126 4411 5839 5931
gtime	2925 3420	inode	5605 5659 5675 6147		6440 6442 6444 6447	5936 5945 5972 6037
gtty	2944 8165		6161 6222 6227 6277		6451 6452 6456 6466	6137 6169 6194 6691
gword	0818 0830 0848 0851		6282 6416 6793 7104		6467 6470 6646 6650	6802 7091 7325 7344
hbcom	5096 5109		7105 7203 7223 7278		6651 6652 6653 6672	7490 7663 7670 7733



	7211 7212 7216	NMOUNT	0133 0277 6103 6154		4495 4935 5737 5771		3776 3778 3847 3860
mpid	0216 1841 1842 1843		7172 7210 7294		5787 5791 5827 5835		3861 3862 3863 3864
	1849 1867	NODEV	0105 3040		5851 5853 5867 5915		3865 3866 3867
MTC	1373	nodev	4659 4660 4661 4662		5929 5959 5967 6019	pad	5575
m_addr	2518 2536 2537 2541		4663 4664 4665 4673		6034 6074 6098 6102	panic	1605 1853 2051 2416
	2564 2565 2567 2571		4675 4677 4678 4679		6104 6108 6111 6155		2719 3236 3521 4377
	2576 2577 2580 2581		4680 4681 4682 4684		6171 6187 6435 6436		4381 4451 4458 4928
m_bufp	0275 6104 6123 6124		4686 4687 4688 4689		6448 6468 6469 6480		4936 6930 7184 7300
	6125 6155 6170 6171		4690 4691		6482 6497 6627 6631	panicstr	2328 2419
	6933 7173 7174 7211	NODEV	5238 6123		6796 6797 6803 6829	partab	7947 8424 8522
	7212	nodev	6566		6853 6864 6990 7079	passc	6394 6517 8544 8695
m_dev	0274 6105 6122 6155	NODEV	6928 7230		7080 7122 7173 7211		9038
	6934 7173 7216 7296	nofault	0757 0766 0854 0855		7284 7306 7309 7312	pc	2693 2734 2754 2756
m_inodp	0276 6121 6167 7295		0871 0872 0876 0881		7326 7460 7461 7590		2757 2766 2767 3725
m_size	2517 2534 2535 2538		0909 0910 0918 1224		7601 7610 7623 7655	pc11	8641 8645 8653 8657
	2542 2564 2565 2566		1225 1228 1232 1259		7665 7666 7671 7729		8658 8673 8675 8688
	2568 2569 2572 2576		1267 1273 1277 1465		7732 7738 7740 8206		8689 8693 8714 8721
	2578 2583 2584		1466	nulldev	4658 4682 4684 6577		8724 8726 8728 8730
n1	7170 7175 7177	NOFILE	0139 0438 1876 3227	nullsys	2864 2912 2942		8731 8734 8743 8744
n2	7170 7176 7177		6624 6828	o1	2447 2454		8754 8755 8756
na	3022 3050 3053 3154	nospace	6966 6969 6986	o2	2447 2454	PCADDR	8607 8659 8663 8674
	3156 3158	nosys	2855 2939 2941 2945	ODDP	7972		8691 8692 8714 8715
namei	3034 3515 3543 4101		2951 2952 2957 2961	ok	4256 4259 4261		8722 8727 8730 8732
	5770 5786 5914 5928		2962 2963 2964 2965	on	6225 6240 6241 6260		8750
	5958 6033 6097 6186		2966 2967 2968 2969		6280 6295 6296 6306	pcclose	4673 8669
	6796 7518		2970 2971 2972 2973		9018 9025 9034 9044	PCIHWAT	8624 8731
nb	5265 5278 5283 5292		2974 2975		9056 9067	pcin	0564 8643 8673 8688
	5306 6419 6447 6448	notavail	4948 4960 5240	open	2917 5765		8693 8730 8731 8734
	6450 6451 6457 6466	notavil	4999	OPEN	8843 8853 8857	PCIPRI	8620 8660 8693
	6472 6479 6488 6497	NPROC	0144 0376 1846 1960	open1	5774 5793 5795 5804	pcleader	8664 8678 8763
	6498 6499 6507		1991 2006 2120 2203	openi	5832 6702	PCOHWAT	8623 8754
nblkdev	4631 4927 5084 6192		2206 3246 3250 3277	os	4368 4373 4374 4380	PCOLWAT	8622 8743
	6720		3327 3639 3810 3953		4383	pcopen	4673 8648
nbp	6420 6480 6484 6488		4023 4172	out	2735 2779 2814 2820	PCOPRI	8621 8755
	6490 6497 6498 6500	nps	2693 3725		3331 3346 3523 3532	pcou	0567
NBUF	0130 4535 4720 5064	NRK	5364		3749 3760 3787 4449	pcout	8644 8714 8743 8744
nc	3022 3051 3062 3063	NRKBLK	5365 5402		4474 5823 5828 5838		8754 8755 8756
	3071 3073 3154 3157	ns	1650 1657 1660 1662		5919 5922 5934 5938	pcoutput	8706 8748 8769
NCALL	0143 0265		1703 1704 1706 1710		5944 5961 5971 6101	pcpbuf	8630 8715
nchrdev	4647 6714 8247		1711		6106 6112 6115 6134	pcpcsr	8629 8663 8714 8750
NCLIST	0146 8146 8240	nseg	1657 1660 1771 3366		7539 7549 7561 7564	pcpint	0566 0567 8739
nd	1650 1657 1660 1662	NSIG	0113 0447 3183 3225		7581 7605 7613 7659	pcrbuf	8628 8730
	1683 1687 1689 1690		3619 3968		7669 8690 8696	pcrcsr	8627 8659 8674 8691
	1692	nswap	0232 1583 4698	out1	6119 6136		8692 8722 8727 8732
NDL11	8012 8015 8026	nt	1650 1657 1660 1662	owner	3564 3579 6791	pcread	4673 8682
newproc	1627 1826 3334		1667 1671 1673 1674	p1	1942 1963 1977 2010	pcrint	0563 0564 8719
newsize	2268 2275 2277 2278	NTEXT	0145 4314 4441		2015 2032 2041 3324	pcstart	8710 8742 8758
	2282	NULL	0104 1752 1833 1847		3326 3335 3727 3768	pcstate	8642 8653 8657 8658
NEXEC	0134 3037 3196		1852 1877 1879 1902		3769 3770 3771 3774		8675 8689 8721 8724
NFILE	0132 5513 6854		1979 1982 2032 2184		3775 3776 3777 3847		8726 8728
nice	2946 3493		2198 2218 2283 3035		3853 3855 3856 3857	pcwrite	4673 8701
NINODE	0131 5675 6161 7103		3229 3235 3284 3328		3859 3860 3863 3869	physio	5259 5479 5486
	7223 7285		3516 3520 3544 3564		3870 3871	PINOD	0155 6963 7007 7074
NKL11	8011 8015 8026 8042		3579 4102 4106 4376	p2	3324 3327 3328 3344		7289
	8043		4401 4402 4407 4440		3727 3750 3751 3752	pipe	2954 7723
NLDELAY	7974		4442 4443 4451 4457		3753 3773 3774 3775	PIPSIZ	7715 7835 7846



	6655 6656 6657		2207 2208 2209 2210	savfp	0888 0889 2698		2143 2208 4385
RHRCOM	5121 5141		2211 2219 2223 2228	savu	0724 0725 1889 1905	SLOCK	0393 1992 2007 4379
rhstart	5123		2240 2241 3966 3970		2189 2281 2284 2846		4385 4466 4468 5312
RHWCOC	5120 5142		3971 3972 3973 3974		4476 4477		5317
rip	1831 1859 1860 1863		3975 3976 4046 4048	sbreak	2929 3354	sloop	1953 2004 2014
	1864 1865 1866 1868		4049 4050 4370 4372	schar	1552 4097 4101 7679	slp0	2022
	1876 1877 1892 1893		4374 4375 4378 4379	sched	1637 1940	slp6	1990
	1894 1903 1908 1917		4380 4383 4384 4385	SCHMAG	3707 3814 3815	SMAPSIZ	0142 0204
	5808 5811 5815 5817		4386 4437 4440 4443	seek	2931 5861	smount	2933 6086
	5818 5825 5826 5830		4444 4451 4465 4466	sep	1650 1654 1677 1698	smp	6090 6102 6108 6109
	5832 5839 6675 6678		4467 4468 4469 4470		1714 3023 3094 3100		6111 6121 6122 6123
	6679 6680 6681 6682		4471 4472 4493 4495		3118 3151		6124 6125 6126 6127
	6691 6705 6708 6709		4496 4497 7347 7349	SETD	2660 2734		6128
	6710 6711		7350 7351 7352 7353	setgid	2958 3460	sp	2693 2811 3725 4136
RKADDR	5363		7354 7355 7357 7358	setpri	2156 2823 3818 3828		4137 4141 4143
rkaddr	5420		7359 7360 7362 7363	setreg	1089 1099 1117 1120	spl0	1292 1293 1976 2079
RKADDR	5447		7378 7381 7382 7383		1196		2092 4944 4947 4956
rkaddr	5447		7385 7386 7388 7389	setrun	2123 2134 3254 3310		4959 4991 5170 5218
RKADDR	5459 5460 5461 5462		7391 7396 7417 7420		3976 4188		5245 5320 5416 5996
rkba	5381		7421 7423 7425 7426	setuid	2935 3439		8228 8289 8565 8676
rkcs	5379 5459 5461 5462		7430 7431 7435 7438	sgtty	8171 8191 8201		8697 8759 8993 9037
rkda	5382 5447		7442 7445 7446 7447	si	4139 4143 4144 4146		9070
rkds	5377 5460		7448 7761 7763 7764		4148 4152 4154 4156	spl1	1292 1297 3803
rker	5378 5460		7772 7773 7774 7796	SIDL	0385 1903	spl4	1292 1302 8672 8686
rkintr	0576 0577 5451		7798 7807 7809 7810	sig	3949 3955 3963 3968		8757 8991
rkio	0577		7865 7867 7868 7869		3972	spl5	1292 1303 3766 5408
rkread	4684 5476		7870 7872 7885 7887	SIGBUS	0123 2722 4072		8222 8263 8283 8559
rkstart	5415 5440 5464 5472		7888 7889 7890 7891	SIGEMT	0120 2748 4070	spl6	1292 1308 1958 2075
rkstrategy	4658 5389 5479 5486	rpp	1830 1846 1847 1848	SIGFPT	0121 2793 2797 4071		2088 4886 4940 4952
rktab	4658 5386 5409 5410		1849 1852 1861 1862	SIGHUP	0114		4988 5007 5164 5201
	5412 5413 5414 5444		1863 1864 1865 1866	SIGINS	0117 2734 2736 4053		5213 5234 5294 5314
	5446 5455 5457 5458		1867 1868 1869 1877		4067	spl7	1292 1313 3854 5983
	5463 5469 5470		1878 1879 1880 1881	SIGINT	0115 8345		9028 9061
rkwc	5380		1890 1891 1895 1904	SIGIOT	0119 2744 4069	sps	4873 4885 4892 5003
rkwrite	4684 5483		1906 1907 1913	SIGKIL	0122 3619 3971		5006 5011 8256 8262
RO	0315 1668 1674	RPS	2613 2679 4057 4060	signal	3949 8345		8266
rootdev	0228 1616 1618 4695		4262	SIGPIPE	0126 7828	SRUN	0384 1591 1861 1908
	6926 6927 6934 7728	rrkbuf	5387 5479 5486	SIGQUIT	0116 4066 8345		1961 2008 2140 2208
rootdir	0206 1616 1617 7533	rsr	2315	SIGSEG	0124 2815 4073	ssig	2960 3614
ROOTINO	0106 1616 1618 7297	rtp	8377 8381 8386 8390	SIGSYS	0125 2781 4074	SSIZE	0137 3118 3131 3150
rp	1741 1745 1748 1750		8392 8393 8399 8403	SIGTRC	0118 2740 4053 4068	SSLEEP	0382 2008 2090
	1751 1755 1758 1760		8412 8413 8414 8423	SINCR	0138 4143	sslep	2947 5979
	1761 1762 1763 1943		8440 8452 8463 8468	size	2528 2535 2537 2538	ssr	0759 0760 1013 1016
	1960 1961 1962 1963		8478		2556 2566 2567 2576		1021 1023 1028 1050
	1964 1977 1978 1979	runin	0218 1954 1955 2080		2577 2578 2579 2584		1150 1171 1465 1467
	1980 1981 1991 1992		2081 2082 3820 3821		2586	SSR0	0613 0647 0759 0761
	1993 2006 2007 2008		3822	sleep	1955 1968 2066 3038		0765 1354
	2009 2010 2011 2015	runout	0219 1967 1968 2143		3314 4182 4190 4943	SSR2	0760
	2023 2024 2032 2033		2144 2145 4387 4388		4955 4990 5167 5204	SSTART	7988 8514
	2034 2036 2037 2039		4389		5215 5297 5316 5994	SSTOP	0387 1993 3253 3301
	2041 2042 2044 2045	runrun	0220 0770 0788 2142		6963 7007 7074 7289		4026 4173
	2046 2047 2068 2071		2166 2196 3807		7790 7838 7870 8225	SSWAP	0394 1907 2240 2241
	2076 2077 2078 2089	RW	0317 1684 1690 1707		8287 8563 8660 8693		2286 4479
	2090 2091 2136 2138		1711		8755 8989	SSYS	0392 1592 1992 2007
	2139 2140 2141 2143	rw	5259 5299 6672 6685	SLOAD	0391 1592 1862 1961	start	0521 0522 0611 0612
	2182 2197 2205 2206		6689 6702 6716 6722		1992 2007 2023 2046		0614



3295	3296	3297	3304		6230	6239	6240	6241	ufalloc	6076	6824	6852		3141	3208	3297	3364	
3305	3314	3317	3326		6244	6262	6290	6294	uid	3441	3443	3444	3445	3568	3569	3581	3582	
3330	3335	3336	3337		6295	6296	6307	6309		3446	3447			3618	3624	3649	3661	
3338	3339	3340	3341		6313	6315	6316	6319	UISA	0306	1563	1599	1745	3662	3670	3671	3672	
3344	3347	3364	3365		6372	6374	6376	6378		1750	1763	5306	9026	3673	4075	4079	4168	
3366	3369	3370	3371		6381	6382	6383	6424		9029	9032	9035	9059	4174	4185	4186	4439	
3373	3376	3378	3388		6521	6522	6523	6524		9062	9065	9068		4455	4461	5743	5744	
3389	3416	3423	3424		6527	6528	6529	6530	UISA0	0678	0680	0690	0698	5756	5758	5773	5774	
3432	3433	3443	3444		6531	6546	6548	6549		0701	0719			5790	5873	5875	5876	
3445	3446	3447	3455		6550	6551	6554	6555	UISA1	0699	0702	0718		5880	5927	5966	5969	
3456	3464	3465	3466		6556	6557	6569	6626	UISD	0304	1561	1600	1755	6021	6036	6096	6113	
3467	3475	3476	3482		6630	6727	6755	6759		1760	1763	9027	9030	6128	8174	8187	8188	
3497	3502	3519	3524		6763	6769	6771	6778		9036	9060	9063	9069	8189	8190	8590		
3525	3526	3527	3547		6798	6814	6816	6829	UISD0	0681	0682	0689	0703	u_base	0425	3085	3139	3525
3554	3555	3567	3568		6830	6833	6856	6863		0705	0717			4115	4121	4463	5269	
3569	3581	3582	3618		6929	6989	7121	7311	UISD1	0704	0706	0716		5743	6372	6374	6376	
3620	3623	3624	3625		7459	7465	7466	7482	UMODE	2659	2699	3706	3788	6381	6522	6523	6530	
3626	3637	3638	3646		7483	7484	7486	7487		3824				6549	6550	6557	7488	
3649	3652	3660	3661		7488	7489	7490	7531	unlink	2922	3510			9050				
3662	3670	3671	3672		7538	7548	7560	7570	up	1741	1744	1747	1751	u_cdir	0428	1618	1619	1883
3673	3789	3790	3791		7571	7572	7576	7580		1752	1753	1754	1757	3232	3554	3555	7531	
3793	3794	3825	3828		7585	7586	7587	7589		1761	1829	1860	1879	u_count	0426	3086	3141	3526
3996	4003	4021	4048		7600	7606	7608	7612		1892	2156	2160	8168	4116	4122	4461	5273	
4051	4052	4054	4055		7622	7626	7636	7638		8174	8175	8176	8177	5291	5310	5322	5744	
4057	4058	4059	4060		7639	7640	7642	7645		8185	8187	8188	8189	5756	5758	6230	6241	
4061	4075	4079	4099		7646	7664	7682	7693		8190				6262	6290	6296	6319	
4100	4103	4111	4113		7695	7736	7740	7744	update	2420	3489	6150	7201	6383	6527	6531	6546	
4114	4115	4116	4117		7745	7795	7796	7798	updlock	0234	1559	7207	7209	6554	7486	7589	7600	
4119	4121	4122	4123		7811	7818	7827	7828		7229				7639	7811	7818	7846	
4127	4141	4143	4146		7844	7845	7846	7847	user	0413				7847	9048	9049		
4148	4149	4150	4156		8027	8031	8032	8172	USER	2662	2700	2721	2733	u_cstime	0451	3291	3292	3293
4168	4169	4174	4175		8174	8187	8188	8189		2739	2743	2747	2751	3294	3336	3337		
4177	4184	4185	4186		8190	8206	8210	8590		2796	2810			u_cutime	0450	3294	3295	3296
4191	4193	4209	4235		8654	8751	8854	9024	USIZE	0103	0636	0646	0662	3339	3340			
4254	4255	4258	4262		9025	9038	9048	9049		1560	1590	1628	1662	u_dbuf	0429	7484	7570	7572
4273	4401	4402	4439		9050	9051	9055	9056		1682	3129	3131	3133	7576	7645	7646		
4448	4455	4461	4462		9057					3370	4116	4119	4233	u_dent	0434	3519	3525	3527
4463	4465	4476	4477	u0	1067	1096				4459	4467	4473		7482	7483	7488	7636	
4478	4479	5269	5273	u1	1067	1189			u_ar0	0452	2701	2812	3155	7640	7646	7664		
5275	5276	5291	5292	u2	1067	1190				3187	3188	3208	3281	u_dirp	0430	2770	4100	5927
5306	5309	5310	5312	u3	1067	1191				3297	3304	3305	3335	6096	7682	7693		
5317	5322	5326	5343	u4	1067	1087				3344	3347	3416	3423	u_dsize	0442	3149	3152	3369
5344	5736	5740	5743	u5	1067	1071	1075	1097		3424	3432	3433	3443	3371	3373	4146	5291	
5744	5745	5751	5752	u6	1067	1069				3455	3456	3464	3475	u_error	0419	1728	2752	2773
5756	5758	5773	5774	u7	1067	1192				3476	3482	3497	3623	2774	2775	2777	2857	
5788	5790	5819	5822	ub	6045	6055	6056	6059		3637	3825	4055	4057	3064	3092	3102	3106	
5831	5833	5835	5850		6060					4058	4059	4060	4061	3317	3330	3547	3620	
5853	5866	5870	5873	UBMAP	0311	1573	1574	5175		4079	4184	4191	4258	3652	4052	4099	4103	
5875	5876	5880	5918		5177					4262	5736	5758	5831	4127	4177	4193	5326	
5927	5930	5933	5935	uchar	3026	3034	3513	3515		5850	5853	5866	5986	5343	5344	5740	5788	
5936	5937	5960	5964		3541	3543	5768	5770		6018	6073	6830	7736	5819	5822	5833	5870	
5966	5969	5986	6018		5784	5786	5912	5914		7744	7745	8206		5918	5930	5933	5937	
6021	6036	6073	6078		5928	5955	5958	6031	u_arg	0440	2763	2766	2770	5960	5964	6094	6114	
6094	6096	6113	6114		6033	6091	6097	6184		3052	3056	3085	3095	6117	6135	6152	6157	
6117	6128	6135	6152		6186	6794	6796	7689		3096	3097	3099	3101	6163	6190	6193	6262	
6157	6163	6190	6193	UDSA	0308	5306				3105	3116	3117	3140	6307	6319	6378	6424	





# 1

**Initialization**  
**Process Initialization**



```

0100 /* fundamental constants: do not change */
0101
0102
0103 #define USIZE 16 /* size of user block (*64) */
0104 #define NULL 0
0105 #define NODEV (-1)
0106 #define ROOTINO 1 /* i number of all roots */
0107 #define DIRSIZ 14 /* max characters per directory */
0108
0109
0110 /* signals: do not change */
0111
0112
0113 #define NSIG 20
0114 #define SIGHUP 1 /* hangup */
0115 #define SIGINT 2 /* interrupt (rubout) */
0116 #define SIGQUIT 3 /* quit (FS) */
0117 #define SIGILL 4 /* illegal instruction */
0118 #define SIGTRC 5 /* trace or breakpoint */
0119 #define SIGIOT 6 /* iot */
0120 #define SIGEMT 7 /* emt */
0121 #define SIGFPE 8 /* floating point exception */
0122 #define SIGKIL 9 /* kill */
0123 #define SIGBUS 10 /* bus error */
0124 #define SIGSEGV 11 /* segmentation violation */
0125 #define SIGSYS 12 /* sys */
0126 #define SIGPIPE 13 /* end of pipe */
0127
0128 /* tunable variables */
0129
0130 #define NBUF 15 /* size of buffer cache */
0131 #define NINODE 100 /* number of in core inodes */
0132 #define NFILE 100 /* number of in core file structures */
0133 #define NMOUNT 5 /* number of mountable file systems */
0134 #define NEXEC 3 /* number of simultaneous exec's */
0135 #define MAXMEM (64*32) /* max core per process;
0136 first number is kw */
0137 #define SSIZE 20 /* initial stack size (*64 bytes) */
0138 #define SINCR 20 /* increment of stack (*64 bytes) */
0139 #define NOFILE 15 /* max open files per process */
0140 #define CANBSIZ 256 /* max size of typewriter line */
0141 #define CMAPSIZ 100 /* size of core allocation area */
0142 #define SMAPSIZ 100 /* size of swap allocation area */
0143 #define NCALL 20 /* max simultaneous time callouts */
0144 #define NPROC 50 /* max number of processes */
0145 #define NTEXT 40 /* max number of pure texts */
0146 #define NCLIST 100 /* max total clist size */
0147 #define HZ 60 /* Ticks/second of the clock */
0148
0149

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

0150
0151 /* priorities: do not alter much */
0152
0153
0154 #define PSWP -100
0155 #define PINOD -90
0156 #define PRIBIO -50
0157 #define PPIPE 1
0158 #define PWAIT 40
0159 #define PSLEP 90
0160 #define PUSER 100
0161
0162 /* Certain processor registers */
0163
0164 #define PS 0177776
0165 #define KL 0177560
0166 #define SW 0177570
0167
0168 /* ----- */
0169
0170 /* structures to access integers : */
0171
0172 /* single integer */
0173
0174 struct { int integ; };
0175
0176
0177 /* in bytes */
0178
0179 struct { char lobyte; char hibyte; };
0180
0181
0182 /* as a sequence */
0183
0184 struct { int r[]; };
0185
0186
0187
0188 /* ----- */
0189
0190
0191
0192
0193
0194
0195
0196
0197
0198
0199

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

0200 /* various global variables */
0201
0202 char canonb[CANBSIZ]; /* buffer for erase and kill */
0203 int coremap[CMAPSIZ]; /* space for core allocation */
0204 int swapmap[SMAPSIZ]; /* space for swap allocation */
0205
0206 int *rootdir; /* pointer to inode of root directory */
0207
0208 int cputype; /* type of cpu =40, 45, or 70 */
0209
0210 int execont; /* number of processes in exec */
0211
0212 int lboot; /* time of day in 60th not in time */
0213 int time[2]; /* time in sec from 1970 */
0214 int tout[2]; /* time of day of next sleep */
0215
0216 int mpd; /* generic for unique process id's */
0217
0218 char runin; /* scheduling flag */
0219 char runout; /* scheduling flag */
0220 char runrun; /* scheduling flag */
0221
0222 char curpri; /* more scheduling */
0223
0224 int maxmem; /* actual max memory per process */
0225
0226 int *lks; /* pointer to clock device */
0227
0228 int rootdev; /* dev of root see conf.c */
0229 int swapdev; /* dev of swap see conf.c */
0230
0231 int swplo; /* block number of swap space */
0232 int nswap; /* size of swap space */
0233
0234 int updlock; /* lock for sync */
0235 int rablok; /* block to be read ahead */
0236
0237 char regloc[]; /* locs. of saved user registers
0238 (see trap.c) */
0239
0240
0241 /* ----- */
0242
0243
0244
0245
0246
0247
0248
0249
0250
0251 /* ----- */
0252
0253 /* The callout structure is for a routine
0254 * arranging to be called by the clock interrupt
0255 * (see clock.c), with a specified argument,
0256 * within a specified amount of time.
0257 * It is used, for example, to time tab delays
0258 * on teletypes. */
0259
0260 struct callo
0261 {
0262     int c_time; /* incremental time */
0263     int c_arg; /* argument to routine */
0264     int (*c_func)(); /* routine */
0265 } callout[NCALT];
0266 /* ----- */
0267
0268 /* Mount structure: used to locate
0269 * the super block of a mounted file.
0270 */
0271
0272 struct mount
0273 {
0274     int m_dev; /* device mounted */
0275     int m_buftp; /* pointer to superblock */
0276     int m_inodp; /* pointer to mounted on inode */
0277 } mount[NMOUNT];
0278 /* ----- */
0279
0280
0281
0282
0283
0284
0285
0286
0287
0288
0289
0290
0291
0292
0293
0294
0295
0296
0297
0298
0299

```

```

0300
0301 /* kt-11 addresses and bits */
0302
0303
0304 #define UISD 0177600 /* first user I-space descriptor
0305                      register */
0306 #define UISA 0177640 /* first user I-space address
0307                      register */
0308 #define UDSA 0177660 /* first user D-space address
0309                      register */
0310
0311 #define UBMAP 0170200 /* access to 11/70 unibus map */
0312
0313
0314
0315 #define RO 02 /* access abilities */
0316 #define WO 04
0317 #define RW 06
0318 #define ED 010 /* expand segment downwards */
0319
0320 /* ----- */
0321
0322 int *ka6; /* 11/40 KISA6; 11/45 KDSA6 */
0323
0324
0325
0326
0327
0328
0329
0330
0331
0332
0333
0334
0335
0336
0337
0338
0339
0340
0341
0342
0343
0344
0345
0346
0347
0348
0349

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

0350 /*
0351 * One structure allocated per active
0352 * process. It contains all data needed
0353 * about the process while the
0354 * process may be swapped out.
0355 * Other per process data (user.h)
0356 * is swapped with the process.
0357 */
0358 struct proc
0359 {
0360 char p_stat;
0361 char p_flag;
0362 char p_pri; /* priority, negative is high */
0363 char p_sig; /* signal number sent to this process */
0364 char p_uid; /* user id, used to direct tty signals */
0365 char p_time; /* resident time for scheduling */
0366 char p_cpu; /* cpu usage for scheduling */
0367 char p_nice; /* nice for scheduling */
0368 int p_ttyp; /* controlling tty */
0369 int p_pid; /* unique process id */
0370 int p_ppid; /* process id of parent */
0371 int p_addr; /* address of swappable image */
0372 int p_size; /* size of swappable image (*64 bytes) */
0373 int p_wchan; /* event process is awaiting */
0374 int *p_textp; /* pointer to text structure */
0375
0376 } proc[NPROC];
0377 /* ----- */
0378
0379 /* stat codes */
0380
0381 /* null 0 not assigned */
0382 #define SSLEEP 1 /* sleeping on high priority */
0383 #define SWAIT 2 /* sleeping on low priority */
0384 #define SRUN 3 /* running */
0385 #define SIDL 4 /* process being created */
0386 #define SZOMB 5 /* process being terminated */
0387 #define SSTOP 6 /* process being traced */
0388
0389 /* flag codes */
0390
0391 #define SLOAD 01 /* in core */
0392 #define SSYS 02 /* scheduling process */
0393 #define SLOCK 04 /* process cannot be swapped */
0394 #define SSWAP 010 /* process is being swapped out */
0395 #define STRC 020 /* process is being traced */
0396 #define SWTED 040 /* another tracing flag */
0397
0398
0399

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

0400 /*
0401 * The user structure.
0402 * One allocated per process.
0403 * Contains all per process data
0404 * that doesn't need to be referenced
0405 * while the process is swapped.
0406 * The user block is USIZE*64 bytes
0407 * long; resides at virtual kernel
0408 * loc 14000; contains the system
0409 * stack per user; is cross referenced
0410 * with the proc structure for the
0411 * same process.
0412 */
0413 struct user
0414 {
0415     int u_rsav[2]; /* save r5,r6 when exchanging stacks */
0416     int u_fsav[25]; /* save fp registers */
0417     /* rsav and fsav must be first in structure */
0418     char u_segflg; /* flag for IO; user or kernel space */
0419     char u_error; /* return error code */
0420     char u_uid; /* effective user id */
0421     char u_gid; /* effective group id */
0422     char u_ruid; /* real user id */
0423     char u_rgid; /* real group id */
0424     int u_procp; /* pointer to proc structure */
0425     char *u_base; /* base address for IO */
0426     char *u_count; /* bytes remaining for IO */
0427     char *u_offset[2]; /* offset in file for IO */
0428     int *u_cdir; /* pointer to inode for current directory */
0429     char u_dbuf[DIRSZ]; /* current pathname component */
0430     char *u_dirp; /* current pointer to inode */
0431     struct {
0432         int u_ino;
0433         char u_name[DIRSZ];
0434     } u_dent;
0435     int *u_dirp; /* inode of parent directory of dirp */
0436     int u_nisa[16]; /* prototype segmentation addresses */
0437     int u_nisd[16]; /* prototype segmentation descriptors */
0438     int u_offile[NOFIL]; /* pointers to file structures of
0439     open files */
0440     int u_arg[5]; /* arguments to current system call */
0441     int u_tsize; /* text size (*64) */
0442     int u_dsize; /* data size (*64) */
0443     int u_ssize; /* stack size (*64) */
0444     int u_sep; /* flag for I and D separation */
0445     int u_gsav[2]; /* label variable for quits & interrupts */
0446     int u_ssav[2]; /* label variable for swapping */
0447     int u_signal[NSIG]; /* disposition of signals */
0448     int u_utime; /* this process user time */
0449     int u_stime; /* this process system time */
};
0450 int u_cstime[2]; /* sum of child's utimes */
0451 int u_cstime[2]; /* sum of child's stimes */
0452 int *u_ar0; /* address of users saved R0 */
0453 int u_prof[4]; /* profile arguments */
0454 char u_inflg; /* catch intr from sys */
0455 /* kernel stack per user
0456 * extends from u + USIZE*64
0457 * backward not to reach here
0458 */
0459 } u;
0460 /* ----- */
0461
0462 /* u_error codes */
0463
0464 /* See section "INTR0(II)" of
0465 * the UNIX Programmer's manual
0466 * for the meanings of these codes. */
0467 #define EFAULT 106
0468 #define ESRCH 3
0469 #define EINTR 4
0470 #define EIO 5
0471 #define ENXIO 6
0472 #define E2BIG 7
0473 #define ENOEXEC 8
0474 #define EBADF 9
0475 #define ECHILD 10
0476 #define EAGAIN 11
0477 #define ENOMEM 12
0478 #define EACCES 13
0479 #define ENOTBLK 15
0480 #define EBUSY 16
0481 #define EXIST 17
0482 #define EXDEV 18
0483 #define ENODEV 19
0484 #define ENODIR 20
0485 #define EISDIR 21
0486 #define EINVAL 22
0487 #define ENFILE 23
0488 #define EMFILE 24
0489 #define ENOTTY 25
0490 #define ETXTBSY 26
0491 #define ERFBG 27
0492 #define ENOSPC 28
0493 #define ESPICE 29
0494 #define EROFS 30
0495 #define EMLINK 31
0496 #define EPIPE 32
0497
0498
0499

```

```

0500 / low core
0501
0502 br4 = 200
0503 br5 = 240
0504 br6 = 300
0505 br7 = 340
0506
0507 . = 0^.
0508   br      1f
0509     4
0510
0511 / trap vectors
0512 trap; br7+0.      / bus error
0513 trap; br7+1.      / illegal instruction
0514 trap; br7+2.      / bpt-trace trap
0515 trap; br7+3.      / iot trap
0516 trap; br7+4.      / power fail
0517 trap; br7+5.      / emulator trap
0518 trap; br7+6.      / system entry
0519
0520 . = 40^.
0521 .globl   start, dump
0522 1: jmp    start
0523     jmp    dump
0524
0525 . = 60^.
0526   klin; br4
0527   klou; br4
0528
0529 . = 70^.
0530   pcin; br4
0531   pcou; br4
0532
0533 . = 100^.
0534   kwlp; br6
0535   kwlp; br6
0536
0537 . = 114^.
0538 trap; br7+7.      / 11/70 parity
0539
0540 . = 200^.
0541   lpou; br4
0542
0543 . = 220^.
0544   rkio; br5
0545
0546 . = 240^.
0547 trap; br7+7.      / programmed interrupt
0548 trap; br7+8.      / floating point
0549 trap; br7+9.      / segmentation violation

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

0550
0551 ////////////////////////////////////////////////////
0552 /           interface code to C
0553 ////////////////////////////////////////////////////
0554
0555 .globl   call, trap
0556
0557 .globl   _klrint
0558 klin:   jsr    r0,call; _klrint
0559
0560 .globl   _klxint
0561 klou:   jsr    r0,call; _klxint
0562
0563 .globl   _pccint
0564 pcin:   jsr    r0,call; _pccint
0565
0566 .globl   _pcpint
0567 pcou:   jsr    r0,call; _pcpint
0568
0569 .globl   _clock
0570 kwlp:   jsr    r0,call; _clock
0571
0572
0573 .globl   _lpint
0574 lpou:   jsr    r0,call; _lpint
0575
0576 .globl   _rkintr
0577 rkio:   jsr    r0,call; _rkintr
0578
0579
0580
0581
0582
0583
0584
0585
0586
0587
0588
0589
0590
0591
0592
0593
0594
0595
0596
0597
0598
0599

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

0600 / machine language assist
0601 / for 11/40
0602
0603 / non-UNIX instructions
0604 mtpi = 6500^tst
0605 mtpi = 6600^tst
0606 wait = 1
0607 rtf = 6
0608 reset = 5
0609
0610 / * ----- *
0611 .globl start,_end,_edata,_main
0612 start:
0613 bit $1,SSR0
0614 bne start
0615 / loop if restart
0616
0617 / initialize systems segments
0618
0619 mov $KISA0,r0
0620 mov $KISD0,r1
0621 mov $200,r4
0622 clr r2
0623 mov $6,r3
0624 l:
0625 mov r2,(r0) +
0626 mov $77406,(r1) +
0627 add r4,r2
0628 sob r3,lb
0629 / initialize user segment
0630
0631 mov $_end+63,r2
0632 ash $-6,r2
0634 bic $1777,r2
0635 mov r2,(r0) +
0636 mov $USIZE-1<8|6,(r1) +
0637 / initialize io segments
0638 / set up counts on supervisor segments
0640
0641 mov $IO,(r0) +
0642 mov $77406,(r1) +
0643 / rw 4k
0644 / get a sp and start segmentation
0645
0646 mov $_u+[USIZE*64.],sp
0647 inc SSR0
0648 / clear bss
0649

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lyons, 1976

```

0650 mov $_edata,r0
0651
0652 l:
0653 clr (r0) +
0654 cmp r0,$_end
0655 blo lb
0656 / clear user block
0657
0658
0659 mov $_u,r0
0660 l:
0661 clr (r0) +
0662 cmp r0,$_u+[USIZE*64.]
0663 blo lb
0664 / set up previous mode and call main
0665
0666 / on return, enter user mode at OR
0667
0668 mov $30000,ps
0669 jsr pc,_main
0670 mov $170000,-(sp)
0671 clr -(sp)
0672 rtf
0673
0674 / * ----- *
0675 .globl _clearseg:
0676 _clearseg:
0677 mov ps,-(sp)
0678 mov UISA0,-(sp)
0679 mov $30340,ps
0680 mov 6(sp),UISA0
0681 mov UISD0,-(sp)
0682 mov $6,UISD0
0683 clr r0
0684 mov $32,r1
0685 l:
0686 clr -(sp)
0687 mtpi (r0) +
0688 sob r1,lb
0689 mov (sp)+,UISD0
0690 mov (sp)+,UISA0
0691 mov (sp)+,ps
0692 rts
0693
0694 / * ----- *
0695 .globl _copyseg
0696 _copyseg:
0697 mov ps,-(sp)
0698 mov UISA0,-(sp)
0699 mov UISA1,-(sp)

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lyons, 1976



```

0700 mov    $30340,PS
0701 mov    10(sp),UISA0
0702 mov    12(sp),UISA1
0703 mov    UISD0,-(sp)
0704 mov    UISD1,-(sp)
0705 mov    $6,UISD0
0706 mov    $6,UISD1
0707 mov    r2,-(sp)
0708 clr    r0
0709 mov    $8192.,r1
0710 mov    $32.,r2
0711 1:
0712 mfpi   (r0)+
0713 mtpi   (r1)+
0714 sob   r2,1b
0715 mov    (sp)+,r2
0716 mov    (sp)+,UISD1
0717 mov    (sp)+,UISD0
0718 mov    (sp)+,UISA1
0719 mov    (sp)+,UISA0
0720 mov    (sp)+,PS
0721 rts    pc
0722
0723 /* ----- */
0724 .globl  _savu, _retu, _aretu
0725 _savu:
0726 bis    $340,PS
0727 mov    (sp)+,r1
0728 mov    (sp),r0
0729 mov    sp,(r0)+
0730 mov    r5,(r0)+
0731 bic    $340,PS
0732 jmp    (r1)
0733
0734 _aretu:
0735 bis    $340,PS
0736 mov    (sp)+,r1
0737 mov    (sp),r0
0738 br    1f
0739
0740 _retu:
0741 bis    $340,PS
0742 mov    (sp)+,r1
0743 mov    (sp),r0
0744 mov    $_u,r0
0745 1:
0746 mov    (r0)+,sp
0747 mov    (r0)+,r5
0748 bic    $340,PS
0749 jmp    (r1)

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

0750
0751 /* ----- */
0752 .globl  trap, call
0753 /* ----- */
0754 .globl  _trap
0755 trap:
0756 mov    PS,-4(sp)
0757 tst    nofault
0758 bne    1f
0759 mov    SSR0,ssr
0760 mov    SSR2,ssr+4
0761 mov    $1,SSR0
0762 jsr    r0,call1; _trap
0763 / no return
0764 1:
0765 mov    $1,SSR0
0766 mov    nofault,(sp)
0767 rtt
0768
0769 /* ----- */
0770 .globl  _runrun, _swtch
0771 call1:
0772 tst    -(sp)
0773 bic    $340,PS
0774 br    1f
0775
0776 call:
0777 mov    PS,-(sp)
0778 1:
0779 mov    r1,-(sp)
0780 mfpi   sp
0781 mov    4(sp),-(sp)
0782 bic    $!37,(sp)
0783 bit    $30000,PS
0784 beq    1f
0785 jsr    pc,* (r0)+
0786 2:
0787 bis    $340,PS
0788 tstb   _runrun
0789 beq    2f
0790 bic    $340,PS
0791 jsr    ps,_swtch
0792 br    2b
0793 2:
0794 tst    (sp)+
0795 mtpi   sp
0796 br    2f
0797 1:
0798 bis    $30000,PS
0799 jsr    pc,* (r0)+

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

0800      cmp      (sp) +, (sp) +
0801 2:
0802      mov      (sp) +, r1
0803      tst      (sp) +
0804      mov      (sp) +, r0
0805      rtt
0806 /* -----
0807      .globl  _fubyte, _subyte
0808 /* -----
0809      .globl  _fubyte, _subyte
0810 /* -----
0811      .globl  _fword, _sword
0812 /* -----
0813      .globl  _fword, _sword
0814      _fubyte:
0815      _fubyte:
0816      mov      2 (sp), r1
0817      bfc     $1, r1
0818      jsr     pc, gword
0819      cmp     r1, 2 (sp)
0820      beq     lf
0821      swab   r0
0822 1:
0823      bfc     $1377, r0
0824      rts
0825      pc
0826      _subyte:
0827      _subyte:
0828      mov      2 (sp), r1
0829      bfc     $1, r1
0830      jsr     pc, gword
0831      mov     r0, - (sp)
0832      cmp     r1, 4 (sp)
0833      beq     lf
0834      movb   6 (sp), 1 (sp)
0835      br     zr
0836 1:
0837      movb   6 (sp), (sp)
0838 2:
0839      mov     (sp) +, r0
0840      jsr     pc, pword
0841      clr     r0
0842      rts
0843      pc
0844      _fword:
0845      _fword:
0846      mov     2 (sp), r1
0847      fword:
0848      jsr     pc, gword
0849      rts

```

Copyright, J. Lions, 1976

Reproduced under license from the Western Electric Company, NY

```

0850      gword:
0851      mov     ps, - (sp)
0852      mov     $340, ps
0853      bts     $340, ps
0854      mov     nofault, - (sp)
0855      mov     $err, nofault
0856      mfpj   (r1)
0857      mov     (sp) +, r0
0858      br     lf
0859      _sword:
0860      mov     2 (sp), r1
0861      _sword:
0862      mov     2 (sp), r1
0863      mov     4 (sp), r0
0864      sword:
0865      jsr     pc, pword
0866      rts
0867      pc
0868      pword:
0869      mov     ps, - (sp)
0870      bts     $340, ps
0871      mov     nofault, - (sp)
0872      mov     $err, nofault
0873      mov     r0, - (sp)
0874      mfpj   (r1)
0875 1:
0876      mov     (sp) +, nofault
0877      mov     (sp) +, ps
0878      rts
0879      pc
0880      err:
0881      mov     (sp) +, nofault
0882      mov     (sp) +, ps
0883      tst     (sp) +
0884      mov     $-1, r0
0885      rts
0886      pc
0887 /* -----
0888      .globl  _savfp, _display
0889      _savfp:
0890      _display:
0891      rts
0892      pc
0893 /* -----
0894      .globl  _incpc
0895      _incpc:
0896      mov     r2, - (sp)
0897      mov     6 (sp), r2
0898      mov     4 (sp), r0
0899      sub     4 (r2), r0
0900      / offset

```

Copyright, J. Lions, 1976

Reproduced under license from the Western Electric Company, NY

```

0900  clc
0901  ror    r0
0902  mul    6(r2),r0      / scale
0903  ashc   $-14.,r0
0904  inc    r1
0905  bic    $1,r1
0906  cmp    r1,2(r2)     / length
0907  bhis   1f
0908  add    (r2),r1      / base
0909  mov    nofault,-(sp)
0910  mov    $2f,nofault
0911  mfpi   (r1)
0912  inc    (sp)
0913  mtpi   (r1)
0914  br     3f
0915  2:
0916  clr    6(r2)
0917  3:
0918  mov    (sp)+,nofault
0919  1:
0920  mov    (sp)+,r2
0921  rts    pc
0922
0923  / Character list get/put
0924
0925  /* ----- */
0926  .globl _getc, _putc
0927  /* ----- */
0928  .globl _cfreelist
0929
0930  _getc:
0931  mov    2(sp),r1
0932  mov    PS,-(sp)
0933  mov    r2,-(sp)
0934  bis    $340,PS
0935  bic    $100,PS      / spl 5
0936  mov    2(r1),r2     / first ptr
0937  beq    9f          / empty
0938  movb   (r2)+,r0     / character
0939  bic    $1377,r0
0940  mov    r2,2(r1)
0941  dec    (r1)+       / count
0942  bne    1f
0943  clr    (r1)+
0944  clr    (r1)+       / last block
0945  br     2f
0946  1:
0947  bit    $7,r2
0948  bne    3f
0949  mov    -10(r2),(r1) / next block

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

0950  add    $2,(r1)
0951  2:
0952  dec    r2
0953  bic    $7,r2
0954  mov    _cfreelist,(r2)
0955  mov    r2,_cfreelist
0956  3:
0957  mov    (sp)+,r2
0958  mov    (sp)+,PS
0959  rts    pc
0960  9:
0961  clr    4(r1)
0962  mov    $-1,r0
0963  mov    (sp)+,r2
0964  mov    (sp)+,PS
0965  rts    pc
0966
0967  _putc:
0968  mov    2(sp),r0
0969  mov    4(sp),r1
0970  mov    PS,-(sp)
0971  mov    r2,-(sp)
0972  mov    r3,-(sp)
0973  bis    $340,PS
0974  bic    $100,PS      / spl 5
0975  mov    4(r1),r2     / last ptr
0976  bne    1f
0977  mov    _cfreelist,r2
0978  beq    9f
0979  mov    (r2),_cfreelist
0980  clr    (r2)+
0981  mov    r2,2(r1)     / first ptr
0982  br     2f
0983  1:
0984  bit    $7,r2
0985  bne    2f
0986  mov    _cfreelist,r3
0987  beq    9f
0988  mov    (r3),_cfreelist
0989  mov    r3,-10(r2)
0990  mov    r3,r2
0991  clr    (r2)+
0992  2:
0993  movb   r0,(r2)+
0994  mov    r2,4(r1)
0995  inc    (r1)         / count
0996  clr    r0
0997  mov    (sp)+,r3
0998  mov    (sp)+,r2
0999  mov    (sp)+,PS

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

1000 rts pc
1001 9:
1002 mov pc,r0
1003 mov (sp)+,r3
1004 mov (sp)+,r2
1005 mov (sp)+,r5
1006 rts pc
1007
1008 /*-----*/
1009 .globl _backup
1010 /*-----*/
1011 .globl _regloc
1012 _backup:
1013 mov 2(sp),ssr+2
1014 mov r2,-(sp)
1015 jsr pc,backup
1016 mov r2,ssr+2
1017 mov (sp)+,r2
1018 movb jflg,r0
1019 bne zf
1020 mov 2(sp),r0
1021 movb ssr+2,r1
1022 jsr pc,lf
1023 movb ssr+3,r1
1024 jsr pc,lf
1025 movb _regloc+7,r1
1026 asl r1
1027 add r0,r1
1028 mov ssr+4,(r1)
1029 clr r0
1030 2:
1031 rts pc
1032 1:
1033 mov r1,-(sp)
1034 asr (sp)
1035 asr (sp)
1036 asr (sp)
1037 bfc $17,r1
1038 movb _regloc(r1),r1
1039 asl r1
1040 add r0,r1
1041 sub (sp)+,(r1)
1042 rts pc
1043 / hard part
1044 / simulate the ssr2 register missing on 11/40
1047 backup:
1048 clr r2
1049 mov $1,bflg / cirs jflg
1050
1050 mov ssr+4,r0
1051 jsr pc,fetch
1052 mov r0,r1
1053 ash $-11,r0
1054 bfc $136,r0
1055 jmp *0f(r0)
1056 0:
1057 t00; t01; t02; t03; t04; t05; t06; t07
1058 t10; t11; t12; t13; t14; t15; t16; t17
1059 t00:
1060 clrb bflg
1061 t10:
1062 t10:
1063 mov r1,r0
1064 swab r0
1065 bfc $16,r0
1066 jmp *0f(r0)
1067 0:
1068 u0; u1; u2; u3; u4; u5; u6; u7
1069 u6: / single op, m[tf]p[.], sxt, illegal
1070 bit $400,r1
1071 beq u5 / all but m[tf], sxt
1072 bit $200,r1
1073 beq lf / mfp[.]
1074 bit $100,r1
1075 bne u5 / sxt
1076 r1
1077 / simulate mfp[.] with double (sp)+,dd
1078 bfc $400,r1 / turn instr into (sp)+
1079 br t01
1080
1081 / simulate mfp[.] with double ss,-(sp)
1082 1:
1083 ash $6,r1
1084 bfc $46,r1 / -(sp)
1085 br t01
1086 u4:
1087 jsr / jsr
1088 mov r1,r0
1089 jsr pc,secreg / assume no fault
1090 bits $173000,r2 / -2 from sp
1091 rts pc
1092 t07: / EIS
1093 t07:
1094 clrb bflg
1095
1096 u0: / jmp, swab
1097 u5: / single op
1098 mov r1,r0
1099 br setreg

```

Reproduced under license from the Western Electric Company, NY

Copyright, J. Lyons, 1976

Reproduced under license from the Western Electric Company, NY

Copyright, J. Lyons, 1976

```

1100
1101 t01:      / mov
1102 t02:      / cmp
1103 t03:      / bit
1104 t04:      / bic
1105 t05:      / bis
1106 t06:      / add
1107 t16:      / sub
1108 clrb      bflg
1109
1110 t11:      / movb
1111 t12:      / cmpb
1112 t13:      / bitb
1113 t14:      / bicb
1114 t15:      / bisb
1115 mov       r1,r0
1116 ash      $-6,r0
1117 jsr      pc,setreg
1118 swab     r2
1119 mov      r1,r0
1120 jsr      pc,setreg
1121
1122 / if delta(dest) is zero,
1123 / no need to fetch source
1124
1125 bit      $370,r2
1126 beq      1f
1127
1128 / if mode(source) is R,
1129 / no fault is possible
1130
1131 bit      $7000,r1
1132 beq      1f
1133
1134 / if reg(source) is reg(dest),
1135 / too bad.
1136
1137 mov      r2,-(sp)
1138 bic     $174370,(sp)
1139 cmpb    1(sp),(sp)+
1140 beq     t17
1141
1142 / start source cycle
1143 / pick up value of reg
1144
1145 mov     r1,r0
1146 ash    $-6,r0
1147 bic    $!7,r0
1148 movb   _regloc(r0),r0
1149 asl    r0

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

1150 add      ssr+2,r0
1151 mov      (r0),r0
1152
1153 / if reg has been incremented,
1154 / must decrement it before fetch
1155
1156 bit      $174000,r2
1157 ble     2f
1158 dec     r0
1159 bit      $10000,r2
1160 beq     2f
1161 dec     r0
1162 2:
1163
1164 / if mode is 6,7 fetch and add X(R) to R
1165
1166 bit      $4000,r1
1167 beq     2f
1168 bit      $2000,r1
1169 beq     2f
1170 mov     r0,-(sp)
1171 mov     ssr+4,r0
1172 sdd     $2,r0
1173 jsr     pc,fetch
1174 add     (sp)+,r0
1175 2:
1176
1177 / fetch operand
1178 / if mode is 3,5,7 fetch *
1179
1180 jsr     pc,fetch
1181 bit     $1000,r1
1182 beq     1f
1183 bit     $6000,r1
1184 bne     fetch
1185 1:
1186 rts     pc
1187
1188 t17:    / illegal
1189 u1:    / br
1190 u2:    / br
1191 u3:    / br
1192 u7:    / illegal
1193 incb   jflg
1194 rts     pc
1195
1196 setreg:
1197 mov     r0,-(sp)
1198 bic     $!7,r0
1199 bis     r0,r2

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

1200 mov (sp)+,r0
1201 ash $-3,r0
1202 bfc $17,r0
1203 movb 0f(r0),r0
1204 tsfb bfig
1205 beq lf
1206 bft $2,r2
1207 beq 2f
1208 bft $4,r2
1209 beq 2f
1210 1:
1211 cmp r0,$20
1212 beq 2f
1213 cmp r0,$-20
1214 beq 2f
1215 asl r0
1216 2:
1217 bisb r0,r2
1218 rts pc
1219 0: .byte 0,0,10,20,-10,-20,0,0
1221 fetch:
1222 bfc $1,r0
1223 mov nofault,-(sp)
1224 mov nofault,-(sp)
1225 mov $1f,nofault
1226 mfpf (r0)
1227 mov (sp)+,r0
1228 mov (sp)+,nofault
1229 rts pc
1230
1231 1:
1232 mov (sp)+,nofault
1233 clrb r2
1234 mov $-1,r0
1235 rts pc
1236 .bss
1237 bfig: .=.+1
1238 bfig: .=.+1
1239 jfg: .=.+1
1240 .text
1241 -----
1242 /* -----
1243 .globl _copyin, _copyout
1244 _copyin:
1245 ]sr pc,copsu
1246 1:
1247 mfpf (r0)+
1248 mov (sp)+,(r1)+
1249 sob r2,1b

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lyons, 1976

```

1250 br 2f
1251
1252 _copyout:
1253 ]sr pc,copsu
1254 1:
1255 mov (r0)+,-(sp)
1256 mfpf (r1)+
1257 sob r2,1b
1258 2:
1259 mov (sp)+,nofault
1260 mov (sp)+,r2
1261 clr r0
1262 rts pc
1263
1264 copsu:
1265 mov (sp)+,r0
1266 mov r2,-(sp)
1267 mov nofault,-(sp)
1268 mov r0,-(sp)
1269 mov 10(sp),r0
1270 mov 12(sp),r1
1271 mov 14(sp),r2
1272 asr r2
1273 mov $1f,nofault
1274 rts pc
1275
1276 1:
1277 mov (sp)+,nofault
1278 mov (sp)+,r2
1279 mov $-1,r0
1280 rts pc
1281
1282 /* -----
1283 .globl _idle
1284 _idle:
1285 mov ps,-(sp)
1286 bfc $340,ps
1287 wait
1288 mov (sp)+,ps
1289 rts pc
1290
1291 /* -----
1292 .globl _sp10, _sp11, _sp14, _sp15, _sp16, _sp17
1293 _sp10:
1294 bfc $340,ps
1295 rts pc
1296
1297 _sp11:
1298 bfc $40,ps
1299 bfc $300,ps

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lyons, 1976

```

1300 rts pc
1301
1302 _spl4:
1303 _spl5:
1304 bis $340,PS
1305 bic $100,PS
1306 rts pc
1307
1308 _spl6:
1309 bis $340,PS
1310 bic $40,PS
1311 rts pc
1312
1313 _spl7:
1314 bis $340,PS
1315 rts pc
1316
1317 /* ----- */
1318 .globl _dpadd
1319 _dpadd:
1320 mov 2(sp),r0
1321 add 4(sp),2(r0)
1322 adc (r0)
1323 rts pc
1324
1325 /* ----- */
1326 .globl _dpcmp
1327 _dpcmp:
1328 mov 2(sp),r0
1329 mov 4(sp),r1
1330 sub 6(sp),r0
1331 sub 8(sp),r1
1332 sbc r0
1333 bge 1f
1334 cmp r0,$-1
1335 bne 2f
1336 cmp r1,$-512.
1337 bhi 3f
1338 2:
1339 mov $-512.,r0
1340 rts pc
1341 1:
1342 bne 2f
1343 cmp r1,$512.
1344 blo 3f
1345 2:
1346 mov $512.,r1
1347 3:
1348 mov r1,r0
1349 rts pc

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

1350
1351 /* ----- */
1352 .globl dump
1353 dump:
1354 bit $1,SSR0
1355 bne dump
1356
1357 / save regs r0,r1,r2,r3,r4,r5,r6,KIA6
1358 / starting at abs location 4
1359
1360 mov r0,4
1361 mov $6,r0
1362 mov r1,(r0)+
1363 mov r2,(r0)+
1364 mov r3,(r0)+
1365 mov r4,(r0)+
1366 mov r5,(r0)+
1367 mov sp,(r0)+
1368 mov KISA6,(r0)+
1369
1370 / dump all of core (ie to first mt error)
1371 / onto mag tape. (9 track or 7 track 'binary')
1372
1373 mov $MTC,r0
1374 mov $60004,(r0)+
1375 clr 2(r0)
1376 1:
1377 mov $-512.,(r0)
1378 inc -(r0)
1379 2:
1380 tstb (r0)
1381 bge 2b
1382 tst (r0)+
1383 bge 1b
1384 reset
1385
1386 / end of file and loop
1387
1388 mov $60007,-(r0)
1389 br .
1390
1391 /* ----- */
1392 .globl _ldiv
1393 _ldiv:
1394 clr r0
1395 mov 2(sp),r1
1396 div 4(sp),r0
1397 rts pc
1398
1399 /* ----- */

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

1400 .globl _lrem
1401 _lrem:
1402 clr r0
1403 mov 2(sp), r1
1404 div 4(sp), r0
1405 mov r1, r0
1406 rts pc
1407
1408 /* ----- */
1409 .globl _lshift
1410 _lshift:
1411 mov 2(sp), r1
1412 mov (r1)+, r0
1413 mov (r1), r1
1414 ashc 4(sp), r0
1415 mov r1, r0
1416 rts pc
1417
1418 /* ----- */
1419 .globl csw
1420 csw:
1421 mov r5, r0
1422 mov sp, r5
1423 mov r4, -(sp)
1424 mov r3, -(sp)
1425 mov r2, -(sp)
1426 jsr pc, (r0)
1427
1428 /* ----- */
1429 .globl cret
1430 cret:
1431 mov r5, r1
1432 mov -(r1), r4
1433 mov -(r1), r3
1434 mov -(r1), r2
1435 mov r5, sp
1436 mov (sp)+, r5
1437 rts pc
1438
1439 /* ----- */
1440 .globl _n
1441 n = 140000
1442 _n = 16.
1443
1444 ps = 177776
1445 SSR0 = 177572
1446 SSR2 = 177576
1447 KISA0 = 172340
1448 KISA6 = 172354
1449 KISD0 = 172300

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lyons, 1976

```

1450 MTC = 172522
1451 UISA0 = 177640
1452 UISA1 = 177642
1453 UISD0 = 177600
1454 UISD1 = 177602
1455 IO = 7600
1456 pc
1457 .data
1458 /* ----- */
1459 .globl _k6, _cpuype
1460 _k6: _cpuype
1461 _cpuype:40.
1462
1463 .bss
1464 /* ----- */
1465 .globl nofault, ssr, badtrap
1466 nofault:.=.+2
1467 ssr: .+=+6
1468 badtrap:.=.+2
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478 /* ----- */
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lyons, 1976



```

1500 #
1501 #include "../param.h"
1502 #include "../user.h"
1503 #include "../system.h"
1504 #include "../proc.h"
1505 #include "../text.h"
1506 #include "../inode.h"
1507 #include "../seg.h"
1508
1509 #define CLOCK1 0177546
1510 #define CLOCK2 0172540
1511 /*
1512  * Icode is the octal bootstrap
1513  * program executed in user mode
1514  * to bring up the system.
1515  */
1516 int icode[]
1517 {
1518     0104413, /* sys exec; init; initp */
1519     0000014,
1520     0000010,
1521     0000777, /* br . */
1522     0000014, /* initp: init; 0 */
1523     0000000,
1524     0062457, /* init: </etc/init\0> */
1525     0061564,
1526     0064457,
1527     0064556,
1528     0000164,
1529 };
1530 /* ----- */
1531
1532 /*
1533  * Initialization code.
1534  * Called from m40.s or m45.s as
1535  * soon as a stack and segmentation
1536  * have been established.
1537  * Functions:
1538  * clear and free user core
1539  * find which clock is configured
1540  * hand craft 0th process
1541  * call all initialization routines
1542  * fork - process 0 to schedule
1543  * - process 1 execute bootstrap
1544  *
1545  * panic: no clock -- neither clock responds
1546  * loop at loc 6 in user mode -- /etc/init
1547  * cannot be executed
1548  */
1549

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

1550 main()
1551 {
1552     extern schar;
1553     register i, *p;
1554
1555     /*
1556      * zero and free all of core
1557      */
1558
1559     updlock = 0;
1560     i = *ka6 + USIZE;
1561     UISD->r[0] = 077406;
1562     for(;;) {
1563         UISA->r[0] = i;
1564         if(fuibyte(0) < 0)
1565             break;
1566         clearseg(i);
1567         maxmem++;
1568         mfree(coremap, 1, i);
1569         i++;
1570     }
1571     if(cputype == 70)
1572     for(i=0; i<62; i+=2) {
1573         UBMAP->r[i] = i<<12;
1574         UBMAP->r[i+1] = 0;
1575     }
1576     printf("mem = %l\n", maxmem*5/16);
1577     printf("RESTRICTED RIGHTS\n\n");
1578     printf("Use, duplication or disclosure is subject to\n");
1579     printf("restrictions stated in Contract with Western\n");
1580     printf("Electric Company, Inc.\n");
1581
1582     maxmem = min(maxmem, MAXMEM);
1583     mfree(swapmap, nswap, swplo);
1584
1585     /*
1586      * set up system process
1587      */
1588
1589     proc[0].p_addr = *ka6;
1590     proc[0].p_size = USIZE;
1591     proc[0].p_stat = SRUN;
1592     proc[0].p_flag = | SLOAD|SSYS;
1593     u.u_procp = &proc[0];
1594
1595     /*
1596      * determine clock
1597      */
1598
1599     UISA->r[7] = ka6[1]; /* io segment */

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

1600 UID5->r[7] = 077406;
1601 lks = CLOCK1;
1602 { if(fuword(lks) == -1)
1603     lks = CLOCK2;
1604     if(fuword(lks) == -1)
1605         panic("no clock");
1606     }
1607     *lks = 0115;
1608
1609 /*
1610     * set up 'known' i-nodes
1611 */
1612     cinit();
1613     binit();
1614     finit();
1615     rootdir = lget(rootdev, ROOTINO);
1616     rootdir->i_flag = &-ILOCK;
1617     rootdir->i_flag = &-ILOCK;
1618     u.n_cdir = lget(rootdev, ROOTINO);
1619     u.n_cdir->i_flag = &-ILOCK;
1620
1621 /*
1622     * make init process
1623     * enter scheduling loop
1624     * with system process
1625 */
1626     if(newproc())
1627     {
1628         expand(USIZE+1);
1629         estabur(0, 1, 0, 0);
1630         copyout(&code, 0, sizeof &code);
1631     }
1632     /*
1633     * Return goes to loc. 0 of user init
1634     * code just copied out.
1635     */
1636     return;
1637 }
1638     sched();
1639 }
1640 /*
1641     * Set up software prototype segmentation
1642     * registers to implement the 3 pseudo
1643     * text,data,stack segment sizes passed
1644     * as arguments.
1645     * The argument sep specifies if the
1646     * text and data+stack segments are to
1647     * be separated.
1648     */
1649     /

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

1650     estabur(nt, nd, ns, sep)
1651     {
1652         register a, *ap, *dp;
1653         if(sep)
1654         {
1655             if(cpu_type == 40)
1656                 goto err;
1657             if(nseg(nd)+nseg(ns) > 8)
1658                 goto err;
1659             } else
1660             if(nseg(nt)+nseg(nd)+nseg(ns) > 8)
1661                 goto err;
1662             goto err;
1663             if(nt+nd+ns+USIZE > maxmem)
1664                 goto err;
1665             ap = &u.n_utsa[0];
1666             dp = &u.n_utsd[0];
1667             while(nt <= 128)
1668             {
1669                 *dp++ = (127<<8) | RO;
1670                 a += 128;
1671                 nt -= 128;
1672             }
1673             if(nt)
1674             {
1675                 *dp++ = ((nt-1)<<8) | RO;
1676                 *ap++ = a;
1677             }
1678             if(sep)
1679             {
1680                 while(ap > &u.n_utsa[8])
1681                 {
1682                     *ap++ = 0;
1683                     *dp++ = 0;
1684                     a = USIZE;
1685                     while(nd <= 128)
1686                     {
1687                         *dp++ = (127<<8) | RW;
1688                         *ap++ = a;
1689                         a += 128;
1690                         nd -= 128;
1691                     }
1692                     if(nd)
1693                     {
1694                         *dp++ = ((nd-1)<<8) | RW;
1695                         *ap++ = a;
1696                         a += nd;
1697                     }
1698                     while(ap > &u.n_utsa[16])
1699                     {
1700                         *ap++ = 0;
1701                         *dp++ = 0;
1702                     }
1703                     if(sep)
1704                     {
1705                         while(ap > &u.n_utsa[16])
1706                         {
1707                             *ap++ = 0;
1708                             *dp++ = 0;
1709                         }
1710                     }
1711                 }
1712             }
1713         }
1714     }

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

1700     *dp++ = 0;
1701     *ap++ = 0;
1702 }
1703 a += ns;
1704 while(ns >= 128) {
1705     a -= 128;
1706     ns -= 128;
1707     *--dp = (127<<8) | RW;
1708     *--ap = a;
1709 }
1710 if(ns) {
1711     *--dp = ((128-ns)<<8) | RW | ED;
1712     *--ap = a-128;
1713 }
1714 if(!sep) {
1715     ap = &u.u_uisa[0];
1716     dp = &u.u_uisa[8];
1717     while(ap < &u.u_uisa[8])
1718         *dp++ = *ap++;
1719     ap = &u.u_uisd[0];
1720     dp = &u.u_uisd[8];
1721     while(ap < &u.u_uisd[8])
1722         *dp++ = *ap++;
1723 }
1724 sureg();
1725 return(0);
1726
1727 err:
1728     u.u_error = ENOMEM;
1729     return(-1);
1730 }
1731 /*----- */
1732
1733 /*
1734  * Load the user hardware segmentation
1735  * registers from the software prototype.
1736  * The software registers must have
1737  * been setup prior by estabur.
1738  */
1739 sureg()
1740 {
1741     register *up, *rp, a;
1742
1743     a = u.u_procp->p_addr;
1744     up = &u.u_uisa[16];
1745     rp = &UISA->r[16];
1746     if(cputype == 40) {
1747         up -= 8;
1748         rp -= 8;
1749     }

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

1750     while(rp > &UISA->r[0])
1751         *--rp = *--up + a;
1752     if((up=u.u_procp->p_textp) != NULL)
1753         a -= up->x_caddr;
1754     up = &u.u_uisd[16];
1755     rp = &UISD->r[16];
1756     if(cputype == 40) {
1757         up -= 8;
1758         rp -= 8;
1759     }
1760     while(rp > &UISD->r[0]) {
1761         *--rp = *--up;
1762         if((*rp & WO) == 0)
1763             rp[(UISA-UISD)/2] -= a;
1764     }
1765 }
1766 /* ----- */
1767
1768 /*
1769  * Return the arg/128 rounded up.
1770  */
1771 nseg(n)
1772 {
1773     return((n+127)>>7);
1774 }
1775 /* ----- */
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

1800 #
1801 /*
1802 /*
1803 #include "../param.h"
1804 #include "../user.h"
1805 #include "../proc.h"
1806 #include "../text.h"
1807 #include "../system.h"
1808 #include "../file.h"
1809 #include "../inode.h"
1810 #include "../buf.h"
1811 #include "....."
1812 /*
1813 /*
1814 * Create a new process-- the internal version of
1815 * sys fork.
1816 * It returns 1 in the new process.
1817 * How this happens is rather hard to understand.
1818 * The essential fact is that the new process is created
1819 * in such a way that it appears to have started executing
1820 * in the same call to newproc as the parent;
1821 * but in fact the code that runs is that of swtch.
1822 * The subtle implication of the return value of swtch
1823 * (see above) is that this is the value that newproc's
1824 * caller in the new process sees.
1825 */
1826 newproc()
1827 {
1828     int a1, a2;
1829     struct proc *p, *np;
1830     register struct proc *rpp;
1831     register *rtp, n;
1832     p = NULL;
1833     /*
1834     /*
1835     * First, just locate a slot for a process
1836     * and copy the useful info from this process into it.
1837     * The panic "cannot happen" because fork already
1838     * checked for the existence of a slot.
1839     */
1840     retry:
1841     mpid++;
1842     if(mpid > 0) {
1843         mpid = 0;
1844         goto retry;
1845     }
1846     for(rpp = kproc[0]; rpp > kproc[NPROC]; rpp++) {
1847         if(rpp->p_stat == NULL && p == NULL)
1848             p = rpp;
1849         if (rpp->p_pid==mpid)

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

1850     goto retry;
1851 }
1852 if (rpp == p) p=NULL;
1853 panic("no proc");
1854 /*
1855 /*
1856 * make proc entry for new proc
1857 */
1858 rtp = u.u_proc;
1859 np = rtp;
1860 srpn;
1861 rpp->p_stat = SRUN;
1862 rpp->p_flag = SLOAD;
1863 rpp->p_uid = rtp->p_uid;
1864 rpp->p_ttyp = rtp->p_ttyp;
1865 rpp->p_nice = rtp->p_nice;
1866 rpp->p_textp = rtp->p_textp;
1867 rpp->p_pid = mpid;
1868 rpp->p_ppid = rtp->p_ppid;
1869 rpp->p_time = 0;
1870 /*
1871 * make duplicate entries
1872 * where needed
1873 */
1874 for(rtp = kproc[0]; rtp < kproc[NPROC]; rtp++) {
1875     if(rpp == rtp) continue;
1876     rpp->f_count++;
1877     if(rpp == rtp) continue;
1878     rpp->f_count++;
1879     if(rpp==np->p_textp) {
1880         rpp->x_count++;
1881         rpp->x_count++;
1882     }
1883     u.u_cdir->i_count++;
1884 /*
1885     * Partially simulate the environment
1886     * of the new process so that when it is actually
1887     * created (by copying) it will look right.
1888     */
1889     savu(u.u_rsav);
1890     rpp = p;
1891     u.u_proc = rpp;
1892     rtp = np;
1893     n = rtp->p_size;
1894     a1 = rtp->p_addr;
1895     rpp->p_size = n;
1896     a2 = malloc(coremap, n);
1897     /*
1898     * If there is not enough core for the
1899     * new process, swap put the current process to

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

1900     * generate the copy.
1901     */
1902     if(a2 == NULL) {
1903         rip->p_stat = SIDL;
1904         rpp->p_addr = a1;
1905         savu(u.u_ssav);
1906         xswap(rpp, 0, 0);
1907         rpp->p_flag = | SSWAP;
1908         rip->p_stat = SRUN;
1909     } else {
1910     /*
1911     * There is core, so just copy.
1912     */
1913         rpp->p_addr = a2;
1914         while(n--)
1915             copyseg(a1++, a2++);
1916     }
1917     u.u_procp = rip;
1918     return(0);
1919 }
1920 /* ----- */
1921
1922 /*
1923 * The main loop of the scheduling (swapping)
1924 * process.
1925 * The basic idea is:
1926 * see if anyone wants to be swapped in;
1927 * swap out processes until there is room;
1928 * swap him in;
1929 * repeat.
1930 * Although it is not remarkably evident, the basic
1931 * synchronization here is on the runin flag, which is
1932 * slept on and is set once per second by the clock routine.
1933 * Core shuffling therefore take place once per second.
1934 *
1935 * panic: swap error -- IO error while swapping.
1936 * this is the one panic that should be
1937 * handled in a less drastic way. Its
1938 * very hard.
1939 */
1940 sched()
1941 {
1942     struct proc *pl;
1943     register struct proc *rp;
1944     register a, n;
1945
1946     /*
1947     * find user to swap in
1948     * of users ready, select one out longest
1949     */

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

1950
1951     goto loop;
1952
1953 loop:
1954     runin++;
1955     sleep(&runin, PSWP);
1956
1957 loop:
1958     spl6();
1959     n = -1;
1960     for(rp = &proc[0]; rp < &proc[NPROC]; rp++)
1961         if(rp->p_stat==SRUN && (rp->p_flag&SLOAD)==0 &&
1962            rp->p_time > n) {
1963             pl = rp;
1964             n = rp->p_time;
1965         }
1966     if(n == -1) {
1967         runout++;
1968         sleep(&runout, PSWP);
1969         goto loop;
1970     }
1971
1972     /*
1973     * see if there is core for that process
1974     */
1975
1976     spl0();
1977     rp = pl;
1978     a = rp->p_size;
1979     if((rp=rp->p_textp) != NULL)
1980         if(rp->x_ccount == 0)
1981             a += rp->x_size;
1982     if((a=malloc(coremap, a)) != NULL)
1983         goto found2;
1984
1985     /*
1986     * none found,
1987     * look around for easy core
1988     */
1989
1990     spl6();
1991     for(rp = &proc[0]; rp < &proc[NPROC]; rp++)
1992         if((rp->p_flag&(SSYS|SLOCK|SLOAD))==SLOAD &&
1993            (rp->p_stat == SWAIT || rp->p_stat==SSTOP))
1994             goto found1;
1995
1996     /*
1997     * no easy core,
1998     * if this process is deserving,
1999     * look around for

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

2000 * oldest process in core
2001 */
2002 {
2003     if(n < 3)
2004         goto slploop;
2005     n = -1;
2006     for(rp = &proc[0]; rp < &proc[NPROC]; rp++)
2007         if((rp->p_flag&(SYS|SLOCK|SLOAD)) == SLOAD &&
2008             (rp->p_stat == SRUN || rp->p_stat == SLEEP) &&
2009             rp->p_time < n)
2010             p1 = rp;
2011             n = rp->p_time;
2012     }
2013     if(n > 2)
2014         goto slploop;
2015     rp = p1;
2016     /*
2017     * swap user out
2018     */
2019     /*
2020     found1:
2021     slp0();
2022     rp->p_flag &= ~SLOAD;
2023     xswap(rp, 1, 0);
2024     goto slploop;
2025     /*
2026     * swap user in
2027     */
2028     /*
2029     found2:
2030     if((rp=p1->p_textp) != NULL)
2031     {
2032         if(rp->x_count == 0)
2033         {
2034             if(swap(rp->x_daddr, a, rp->x_size, B_READ))
2035                 goto swaper;
2036             rp->x_caddr = a;
2037             a += rp->x_size;
2038         }
2039         rp->x_count++;
2040     }
2041     rp = p1;
2042     if(swap(rp->p_addr, a, rp->p_size, B_READ))
2043         goto swaper;
2044     mfree(swapmap, (rp->p_size+7)/8, rp->p_addr);
2045     rp->p_addr = a;
2046     rp->p_flag |= SLOAD;
2047     rp->p_time = 0;
2048     goto slploop;
2049
2050 swaper:
2051     panic("swap error");
2052 }
2053 /* -----
2054 */
2055 */
2056 * Give up the processor till a wakeup occurs
2057 * on chan, at which time the process
2058 * enters the scheduling queue at priority pri.
2059 * The most important effect of pri is that when
2060 * pri < 0 a signal cannot disturb the sleep;
2061 * if the pri >= 0 signals will be processed.
2062 * Callers of this routine must be prepared for
2063 * premature return, and check that the reason for
2064 * sleeping has gone away.
2065 */
2066     slp(chan, pri)
2067 {
2068     register *rp, s;
2069     s = PS->integ;
2070     rp = u.u_proc;
2071     if(pri >= 0)
2072     {
2073         if(ssig())
2074             goto psig;
2075         slp();
2076         rp->p_wchan = chan;
2077         rp->p_stat = SWAIT;
2078         rp->p_pri = pri;
2079         slp0();
2080     }
2081     runin = 0;
2082     wakeup(&runin);
2083     }
2084     switch();
2085     if(ssig())
2086         goto psig;
2087     } else
2088     {
2089         slp6();
2090         rp->p_wchan = chan;
2091         rp->p_stat = SLEEP;
2092         rp->p_pri = pri;
2093         slp0();
2094     }
2095     PS->integ = s;
2096     return;
2097     /*
2098     * If priority was low (<=0) and
2099

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

2050 swaper:
2051     panic("swap error");
2052 }
2053 /* -----
2054 */
2055 */
2056 * Give up the processor till a wakeup occurs
2057 * on chan, at which time the process
2058 * enters the scheduling queue at priority pri.
2059 * The most important effect of pri is that when
2060 * pri < 0 a signal cannot disturb the sleep;
2061 * if the pri >= 0 signals will be processed.
2062 * Callers of this routine must be prepared for
2063 * premature return, and check that the reason for
2064 * sleeping has gone away.
2065 */
2066     slp(chan, pri)
2067 {
2068     register *rp, s;
2069     s = PS->integ;
2070     rp = u.u_proc;
2071     if(pri >= 0)
2072     {
2073         if(ssig())
2074             goto psig;
2075         slp6();
2076         rp->p_wchan = chan;
2077         rp->p_stat = SWAIT;
2078         rp->p_pri = pri;
2079         slp0();
2080     }
2081     runin = 0;
2082     wakeup(&runin);
2083     }
2084     switch();
2085     if(ssig())
2086         goto psig;
2087     } else
2088     {
2089         slp6();
2090         rp->p_wchan = chan;
2091         rp->p_stat = SLEEP;
2092         rp->p_pri = pri;
2093         slp0();
2094     }
2095     PS->integ = s;
2096     return;
2097     /*
2098     * If priority was low (<=0) and
2099

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

2100 * there has been a signal,
2101 * execute non-local goto to
2102 * the qsav location.
2103 * (see trap1/trap.c)
2104 */
2105 psig:
2106 aretu(u.u_qsav);
2107 }
2108 /*----- */
2109
2110 /*
2111 * Wake up all processes sleeping on chan.
2112 */
2113 wakeup(chan)
2114 {
2115     register struct proc *p;
2116     register c, i;
2117
2118     c = chan;
2119     p = &proc[0];
2120     i = NPROC;
2121     do {
2122         if(p->p_wchan == c) {
2123             setrun(p);
2124         }
2125         p++;
2126     } while(--i);
2127 }
2128 /*----- */
2129
2130 /*
2131 * Set the process running;
2132 * arrange for it to be swapped in if necessary.
2133 */
2134 setrun(p)
2135 {
2136     register struct proc *rp;
2137
2138     rp = p;
2139     rp->p_wchan = 0;
2140     rp->p_stat = SRUN;
2141     if(rp->p_pri < curpri)
2142         runrun++;
2143     if(runout != 0 && (rp->p_flag&SLOAD) == 0) {
2144         runout = 0;
2145         wakeup(&runout);
2146     }
2147 }
2148 /*----- */
2149

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

2150 /*
2151 * Set user priority.
2152 * The rescheduling flag (runrun)
2153 * is set if the priority is higher
2154 * than the currently running process.
2155 */
2156 setpri(up)
2157 {
2158     register *pp, p;
2159
2160     pp = up;
2161     p = (pp->p_cpu & 0377)/16;
2162     p += PUSER + pp->p_nice;
2163     if(p > 127)
2164         p = 127;
2165     if(p > curpri)
2166         runrun++;
2167     pp->p_pri = p;
2168 }
2169 /*----- */
2170
2171 /*
2172 * This routine is called to reschedule the CPU.
2173 * if the calling process is not in RUN state,
2174 * arrangements for it to restart must have
2175 * been made elsewhere, usually by calling via sleep.
2176 */
2177
2178 swtch()
2179 {
2180     static struct proc *p;
2181     register i, n;
2182     register struct proc *rp;
2183
2184     if(p == NULL)
2185         p = &proc[0];
2186     /*
2187     * Remember stack of caller
2188     */
2189     savu(u.u_rsav);
2190     /*
2191     * Switch to scheduler's stack
2192     */
2193     retu(proc[0].p_addr);
2194
2195 loop:
2196     runrun = 0;
2197     rp = p;
2198     p = NULL;
2199     n = 128;

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

2200 /*
2201 * Search for highest-priority runnable process
2202 */
2203 i = NPROC;
2204 do {
2205     rp++;
2206     if(rp >= &proc[NPROC])
2207         rp = &proc[0];
2208     if(rp->p_stat==SRUN && (rp->p_flag&SLDAD)!=0) {
2209         if(rp->p_pri < n)
2210             p = rp;
2211         n = rp->p_pri;
2212     }
2213 } while(--i);
2214 /*
2215 * If no process is runnable, idle.
2216 */
2217 if(p == NULL)
2218     return(rp->p_addr);
2219 /*
2220 * Switch to stack of the new process and set up
2221 * his segmentation registers.
2222 */
2223 rp = p;
2224 curpri = n;
2225 /* Switch to stack of the new process and set up
2226 * his segmentation registers.
2227 */
2228 return(rp->p_addr);
2229 sureg();
2230 /*
2231 * If the new process paused because it was
2232 * swapped out, set the stack level to the last call
2233 * to savu(u_ssav). This means that the return
2234 * which is executed immediately after the call to aretu
2235 * actually returns from the last routine which did
2236 * the savu.
2237 *
2238 * You are not expected to understand this.
2239 */
2240 if(rp->p_flag&SSWAP) {
2241     rp->p_flag = &-SSWAP;
2242     aretu(u_ssav);
2243 }
2244 /* The value returned here has many subtle implications.
2245 * See the newproc comments.
2246 */
2247 return(1);
2248 }
2249 /* ----- */

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

2250 /*
2251 * Change the size of the data+stack regions of the process.
2252 * If the size is shrinking, it's easy-- just release the
2253 * extra core. If it's growing, and there is core, just
2254 * allocate it and copy the image, taking care to reset
2255 * registers to account for the fact that the system's
2256 * stack has moved.
2257 * If there is no core, arrange for the process to be
2258 * swapped out after adjusting the size requirement--
2259 * when it comes in, enough core will be allocated.
2260 * Because of the save and SSWAP flags, control will
2261 * resume after the swap in swtch, which executes the return
2262 * from this stack level.
2263 *
2264 * After the expansion, the caller will take care of copying
2265 * the user's stack towards or away from the data area.
2266 */
2267 expand(newsize)
2268     int i, n;
2269     register *p, a1, a2;
2270 {
2271     p = u_proc;
2272     n = p->p_size;
2273     a1 = p->p_addr;
2274     p->p_size = newsize;
2275     if(n >= newsize)
2276         return;
2277     mtree(coremap, n-newsize, a1+newsize);
2278     return;
2279 }
2280 savu(u_ssav);
2281 a2 = malloc(coremap, newsize);
2282 if(a2 == NULL) {
2283     *swapped out, set the stack level to the last call
2284     * to savu(u_ssav). This means that the return
2285     * which is executed immediately after the call to aretu
2286     * actually returns from the last routine which did
2287     * the savu.
2288     * You are not expected to understand this.
2289 */
2290 p->p_addr = a2;
2291 for(i=0; i<n; i++)
2292     copyseg(a1+i, a2+i);
2293 mtree(coremap, n, a1);
2294 return(p->p_addr);
2295 sureg();
2296 }
2297 /* ----- */
2298 }
2299 /* ----- */

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976



```

2300 #
2301 /*
2302 */
2303
2304 #include "../param.h"
2305 #include "../seg.h"
2306 #include "../buf.h"
2307 #include "../conf.h"
2308
2309 /*
2310 * Address and structure of the
2311 * KL-11 console device registers.
2312 */
2313 struct
2314 {
2315     int    rsr;
2316     int    rbr;
2317     int    xsr;
2318     int    xbr;
2319 };
2320 /* ----- */
2321
2322 /*
2323 * In case console is off,
2324 * panicstr contains argument to last
2325 * call to panic.
2326 */
2327
2328 char      *panicstr;
2329
2330 /*
2331 * Scaled down version of C library printf.
2332 * Only %s %l %d (==%l) %o are recognized.
2333 * Used to print diagnostic information
2334 * directly on console tty.
2335 * Since it is not interrupt driven,
2336 * all system activities are pretty much
2337 * suspended.
2338 * Printf should not be used for chit-chat.
2339 */
2340 printf(fmt,x1,x2,x3,x4,x5,x6,x7,x8,x9,xa,xb,xc)
2341 char fmt[];
2342 {
2343     register char *s;
2344     register *adx, c;
2345
2346     adx = &x1;
2347 loop:
2348     while((c = *fmt++) != '%') {
2349         if(c == '\0')

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

2350         return
2351         putchar(c);
2352     }
2353     c = *fmt++;
2354     if(c == 'd' || c == 'l' || c == 'o')
2355         printn(*adx, c=='o'? 8: 10);
2356     if(c == 's') {
2357         s = *adx;
2358         while(c = *s++)
2359             putchar(c);
2360     }
2361     adx++;
2362     goto loop;
2363 }
2364 /* ----- */
2365
2366 /*
2367 * Print an unsigned integer in base b.
2368 */
2369 printn(n, b)
2370 {
2371     register a;
2372
2373     if(a = ldiv(n, b))
2374         printn(a, b);
2375     putchar(ldrem(n, b) + '0');
2376 }
2377 /* ----- */
2378
2379 /*
2380 * Print a character on console.
2381 * Attempts to save and restore device
2382 * status.
2383 * If the switches are 0, all
2384 * printing is inhibited.
2385 */
2386 putchar(c)
2387 {
2388     register rc, s;
2389
2390     rc = c;
2391     if(SW->integ == 0)
2392         return;
2393     while((KL->xsr&0200) ==0)
2394         ;
2395     if(rc == 0)
2396         return;
2397     s = KL->xsr;
2398     KL->xsr = 0;
2399     KL->xbr = rc;

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

2400 if(rc == '\n') {
2401     putchar('\r');
2402     putchar(0177);
2403     putchar(0177);
2404 }
2405     putchar(0);
2406     KL->xsr = s;
2407 }
2408 /* ----- */
2409
2410 /*
2411  * panic is called on unresolvable
2412  * fatal errors.
2413  * It syncs, prints "panic: msg" and
2414  * then loops.
2415  */
2416 panic(s)
2417     char *s;
2418 {
2419     panicstr = s;
2420     update();
2421     printf("panic: %s\n", s);
2422     for(;;)
2423         idle();
2424 }
2425 /* ----- */
2426
2427 /*
2428  * prdev prints a warning message of the
2429  * form "msg on dev x/y".
2430  * x and y are the major and minor parts of
2431  * the device argument.
2432  */
2433 prdev(str, dev)
2434 {
2435     printf("%s on dev %l%\n", str, dev.d_major, dev.d_minor);
2436 }
2437 /* ----- */
2438
2439
2440 /*
2441  * devert prints a diagnostic from
2442  * a device driver.
2443  * It prints the device, block number,
2444  * and an octal word (usually some error
2445  * status register) passed as argument.
2446  */
2447 devert(bp, o1, o2)
2448     int *bp;
2449 {
2450     register *rbp;
2451     rbp = bp;
2452     prdev("err", rbp->b_dev);
2453     printf("\n%l er%o\n", rbp->b_bkmo, o1, o2);
2454 }
2455 /* ----- */

```

```

2450     register *rbp;
2451     rbp = bp;
2452     prdev("err", rbp->b_dev);
2453     printf("\n%l er%o\n", rbp->b_bkmo, o1, o2);
2454 }
2455 /* ----- */
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499

```

```

2500 #
2501 /*
2502 */
2503
2504 /*
2505 * Structure of the coremap and swapmap
2506 * arrays. Consists of non-zero count
2507 * and base address of that many
2508 * contiguous units.
2509 * (The coremap unit is 64 bytes,
2510 * the swapmap unit is 512 bytes)
2511 * The addresses are increasing and
2512 * the list is terminated with the
2513 * first zero count.
2514 */
2515 struct map
2516 {
2517     char *m_size;
2518     char *m_addr;
2519 };
2520 /* ----- */
2521
2522 /*
2523 * Allocate size units from the given
2524 * map. Return the base of the allocated
2525 * space.
2526 * Algorithm is first fit.
2527 */
2528 malloc(mp, size)
2529 struct map *mp;
2530 {
2531     register int a;
2532     register struct map *bp;
2533
2534     for (bp = mp; bp->m_size; bp++) {
2535         if (bp->m_size >= size) {
2536             a = bp->m_addr;
2537             bp->m_addr += size;
2538             if ((bp->m_size -= size) == 0)
2539                 do {
2540                     bp++;
2541                     (bp-1)->m_addr = bp->m_addr;
2542                 } while ((bp-1)->m_size = bp->m_size);
2543             return(a);
2544         }
2545     }
2546     return(0);
2547 }
2548 /*----- */
2549

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

2550 /*
2551 * Free the previously allocated space aa
2552 * of size units into the specified map.
2553 * Sort aa into map and combine on
2554 * one or both ends if possible.
2555 */
2556 mfree(mp, size, aa)
2557 struct map *mp;
2558 {
2559     register struct map *bp;
2560     register int t;
2561     register int a;
2562
2563     a = aa;
2564     for (bp = mp; bp->m_addr <= a && bp->m_size != 0; bp++) {
2565         if (bp > mp && (bp-1)->m_addr + (bp-1)->m_size == a) {
2566             (bp-1)->m_size += size;
2567             if (a + size == bp->m_addr) {
2568                 (bp-1)->m_size += bp->m_size;
2569                 while (bp->m_size) {
2570                     bp++;
2571                     (bp-1)->m_addr = bp->m_addr;
2572                     (bp-1)->m_size = bp->m_size;
2573                 }
2574             }
2575         } else {
2576             if (a + size == bp->m_addr && bp->m_size) {
2577                 bp->m_addr -= size;
2578                 bp->m_size += size;
2579             } else if (size) do {
2580                 t = bp->m_addr;
2581                 bp->m_addr = a;
2582                 a = t;
2583                 t = bp->m_size;
2584                 bp->m_size = size;
2585                 bp++;
2586             } while (size = t);
2587         }
2588     }
2589     /*----- */
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976



# 2

**Traps, Interrupts  
and System Calls  
Process Management**



```

2600 /*
2601  * Location of the users' stored
2602  * registers relative to R0.
2603  * Usage is u.u_ar0[XX].
2604  */
2605 #define R0 (0)
2606 #define R1 (-2)
2607 #define R2 (-9)
2608 #define R3 (-8)
2609 #define R4 (-7)
2610 #define R5 (-6)
2611 #define R6 (-3)
2612 #define R7 (1)
2613 #define RPS (2)
2614
2615 #define TBIT 020 /* PS trace bit */
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

2650 #
2651 #include "../param.h"
2652 #include "../system.h"
2653 #include "../user.h"
2654 #include "../proc.h"
2655 #include "../reg.h"
2656 #include "../seg.h"
2657
2658 #define EBIT 1 /* user error bit in PS: C-bit */
2659 #define UMODE 0170000 /* user-mode bits in PS word */
2660 #define SETD 0170011 /* SETD instruction */
2661 #define SYS 0104400 /* sys (trap) instruction */
2662 #define USER 020 /* user-mode flag added to dev */
2663
2664 /*
2665  * structure of the system entry table (sysent.c)
2666  */
2667 struct sysent {
2668     int count; /* argument count */
2669     int (*call)(); /* name of handler */
2670 } sysent[64];
2671 /* ----- */
2672
2673 /*
2674  * Offsets of the user's registers relative to
2675  * the saved r0. See reg.h
2676  */
2677 char regloc[9]
2678 {
2679     R0, R1, R2, R3, R4, R5, R6, R7, RPS
2680 };
2681 /* ----- */
2682
2683 /*
2684  * Called from l40.s or l45.s when a processor trap occurs.
2685  * The arguments are the words saved on the system stack
2686  * by the hardware and software during the trap processing.
2687  * Their order is dictated by the hardware and the details
2688  * of C's calling sequence. They are peculiar in that
2689  * this call is not 'by value' and changed user registers
2690  * get copied back on return.
2691  * dev is the kind of trap that occurred.
2692  */
2693 trap(dev, sp, r1, nps, r0, pc, ps)
2694 {
2695     register i, a;
2696     register struct sysent *callp;
2697
2698     savfp();
2699     if ((ps&UMODE) == UMODE)

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

2700 dev = | USER;
2701 n.u.ar0 = kr0;
2702 switch(dev) {
2703
2704 /*
2705 * Trap not expected.
2706 * Usually a kernel mode bus error.
2707 * The numbers printed are used to
2708 * find the hardware PS/PC as follows.
2709 * (all numbers in octal 18 bits)
2710 address_of_saved_ps =
2711 (ka6*0100) + aps - 0140000;
2712 address_of_saved_pc =
2713 address_of_saved_ps - 2;
2714 */
2715 default:
2716 printf("ka6 = %\n", *ka6);
2717 printf("aps = %\n", &ps);
2718 printf("trap type %\n", dev);
2719 panic("trap");
2720 case 0+USER: /* bus error */
2721 f = SIGBUS;
2722 break;
2723 /*
2724 * If illegal instructions are not
2725 * being caught and the offending instruction
2726 * is a SETD, the trap is ignored.
2727 * This is because C produces a SETD at
2728 * the beginning of every program which
2729 * will trap on CPUs without 11/45 FPU.
2730 */
2731 case 1+USER: /* illegal instruction */
2732 if(fuword(pc-2)==SETD && n.u.signal[SIGINS]==0)
2733 goto out;
2734 f = SIGINS;
2735 break;
2736 case 2+USER: /* bpt or trace */
2737 f = SIGTRC;
2738 break;
2739 case 3+USER: /* lot */
2740 f = SIGIOT;
2741 break;
2742 case 5+USER: /* emt */
2743 f = SIGEMT;
2744 break;
2745 dev = | USER;
2746
2747 */

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

2750 case 6+USER: /* sys call */
2751 n.u.error = 0;
2752 ps = &~EBIT;
2753 callp = &sysent[fuword(pc-2)&077];
2754 if (callp == &sysent) /* indirect */
2755 a = fuword(pc);
2756 pc += 2;
2757 f = fuword(a);
2758 if ((f & ~077) != SYS)
2759 f = fuword(a);
2760 /* illegal */
2761 callp = &sysent[f&077];
2762 for(i=0; i<callp->count; i++)
2763 n.u_arg[i] = fuword(a += 2);
2764 } else {
2765 for(i=0; i<callp->count; i++)
2766 n.u_arg[i] = fuword(pc);
2767 pc += 2;
2768 }
2769 n.u.dirp = n.u_arg[0];
2770 trapl(callp->call);
2771 if(n.u.intflg)
2772 n.u.error = RINTR;
2773 if(n.u.error > 100) {
2774 if(n.u_error)
2775 ps = |EBIT;
2776 r0 = n.u_error;
2777 }
2778 goto out;
2779 }
2780 f = SIGSYS;
2781 break;
2782 /*
2783 * Since the floating exception is an
2784 * imprecise trap, a user generated
2785 * trap may actually come from kernel
2786 * mode. In this case, a signal is sent
2787 * to the current process to be picked
2788 * up later.
2789 */
2790 case 8: /* floating exception */
2791 psignal(n.u.proc, SIGFPE);
2792 return;
2793 case 8+USER:
2794 f = SIGFPE;
2795 break;
2796
2797 */

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976



```

2800 /*
2801  * If the user SP is below the stack segment,
2802  * grow the stack automatically.
2803  * This relies on the ability of the hardware
2804  * to restart a half executed instruction.
2805  * On the 11/40 this is not the case and
2806  * the routine backup/l40.s may fail.
2807  * The classic example is on the instruction
2808  *      cmp      -(sp),-(sp)
2809  */
2810 case 9+USER: /* segmentation exception */
2811     a = sp;
2812     if(backup(u.u_ar0) == 0)
2813         if(grow(a))
2814             goto out;
2815     i = SIGSEGV;
2816     break;
2817 }
2818 psignal(u.u_procp, i);
2819
2820 out:
2821 if(issig())
2822     psig();
2823 setpri(u.u_procp);
2824 }
2825 /* ----- */
2826
2827 /*
2828  * Call the system-entry routine f (out of the
2829  * sysent table). This is a subroutine for trap, and
2830  * not in-line, because if a signal occurs
2831  * during processing, an (abnormal) return is simulated from
2832  * the last caller to savu(qsav); if this took place
2833  * inside of trap, it wouldn't have a chance to clean up.
2834  *
2835  * If this occurs, the return takes place without
2836  * clearing u_intflg; if it's still set, trap
2837  * marks an error which means that a system
2838  * call (like read on a typewrite) got interrupted
2839  * by a signal.
2840  */
2841 trap1(f)
2842 int (*f)();
2843 {
2844
2845     u.u_intflg = 1;
2846     savu(u.u_qsav);
2847     (*f)();
2848     u.u_intflg = 0;
2849 }

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

2850 /* ----- */
2851
2852 /*
2853  * nonexistent system call-- set fatal error code.
2854  */
2855 nosys()
2856 {
2857     u.u_error = 100;
2858 }
2859 /*----- */
2860
2861 /*
2862  * Ignored system call
2863  */
2864 nullsys()
2865 {
2866 }
2867 /* ----- */
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

2900 #
2901 /*
2902 */
2903
2904 /*
2905 * This table is the switch used to transfer
2906 * to the appropriate routine for processing a system call.
2907 * Each row contains the number of arguments expected
2908 * and a pointer to the routine.
2909 */
2910 int sysent[]
2911 {

```

```

2912 0, knllys, /* 0 = indir */
2913 0, kexit, /* 1 = exit */
2914 0, kfork, /* 2 = fork */
2915 2, kread, /* 3 = read */
2916 2, kwrite, /* 4 = write */
2917 2, kopen, /* 5 = open */
2918 0, kclose, /* 6 = close */
2919 0, kwait, /* 7 = wait */
2920 2, kcreat, /* 8 = creat */
2921 2, klink, /* 9 = link */
2922 1, kunlink, /* 10 = unlink */
2923 2, kexec, /* 11 = exec */
2924 1, kchdir, /* 12 = chdir */
2925 0, kptime, /* 13 = time */
2926 3, kmknod, /* 14 = mknod */
2927 2, kchmod, /* 15 = chmod */
2928 2, kchown, /* 16 = chown */
2929 1, ksbreak, /* 17 = break */
2930 2, kstat, /* 18 = stat */
2931 2, kseek, /* 19 = seek */
2932 0, kgetpid, /* 20 = getpid */
2933 3, kmount, /* 21 = mount */
2934 1, kumount, /* 22 = umount */
2935 0, ksetuid, /* 23 = setuid */
2936 0, kgetuid, /* 24 = getuid */
2937 0, kstime, /* 25 = stime */
2938 3, kptrace, /* 26 = ptrace */
2939 0, knosys, /* 27 = x */
2940 1, kstat, /* 28 = tstat */
2941 0, knosys, /* 29 = x */
2942 1, knllys, /* 30 = smdate */
2943 1, kstat, /* 31 = stty */
2944 1, kgtty, /* 32 = gty */
2945 0, knosys, /* 33 = x */
2946 0, knice, /* 34 = nice */
2947 0, ksleep, /* 35 = sleep */
2948 0, ksync, /* 36 = sync */
2949 1, kkill, /* 37 = kill */

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

3000 #
3001 #include "../param.h"
3002 #include "../system.h"
3003 #include "../user.h"
3004 #include "../proc.h"
3005 #include "../buf.h"
3006 #include "../reg.h"
3007 #include "../inode.h"
3008
3009 /*
3010  * exec system call.
3011  * Because of the fact that an I/O buffer is used
3012  * to store the caller's arguments during exec,
3013  * and more buffers are needed to read in the text file,
3014  * deadly embraces waiting for free buffers are possible.
3015  * Therefore the number of processes simultaneously
3016  * running in exec has to be limited to NEXEC.
3017  */
3018 #define EXPRI -1
3019
3020 exec()
3021 {
3022     int ap, na, nc, *bp;
3023     int ts, ds, sep;
3024     register c, *ip;
3025     register char *cp;
3026     extern uchar;
3027
3028     /*
3029      * pick up file names
3030      * and check various modes
3031      * for execute permission
3032      */
3033
3034     ip = namei(&uchar, 0);
3035     if(ip == NULL)
3036         return;
3037     while(execnt >= NEXEC)
3038         sleep(&execnt, EXPRI);
3039     execnt++;
3040     bp = getblk(NODEV);
3041     if(access(ip, IEXEC) || (ip->i_mode&IFMT)!=0)
3042         goto bad;
3043
3044     /*
3045      * pack up arguments into
3046      * allocated disk buffer
3047      */
3048
3049     cp = bp->b_addr;

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

3050     na = 0;
3051     nc = 0;
3052     while(ap = fuword(u.u_arg[1])) {
3053         na++;
3054         if(ap == -1)
3055             goto bad;
3056         u.u_arg[1] += 2;
3057         for(;;) {
3058             c = fubyte(ap++);
3059             if(c == -1)
3060                 goto bad;
3061             *cp++ = c;
3062             nc++;
3063             if(nc > 510) {
3064                 u.u_error = E2BIG;
3065                 goto bad;
3066             }
3067             if(c == 0)
3068                 break;
3069         }
3070     }
3071     if((nc&1) != 0) {
3072         *cp++ = 0;
3073         nc++;
3074     }
3075
3076     /* read in first 8 bytes
3077      * of file for segment
3078      * sizes:
3079      * w0 = 407/410/411 (410 -> RO text) (411 -> sep ID)
3080      * w1 = text size
3081      * w2 = data size
3082      * w3 = bss size
3083      */
3084
3085     u.u_base = &u.u_arg[0];
3086     u.u_count = 8;
3087     u.u_offset[1] = 0;
3088     u.u_offset[0] = 0;
3089     u.u_segflg = 1;
3090     readi(ip);
3091     u.u_segflg = 0;
3092     if(u.u_error)
3093         goto bad;
3094     sep = 0;
3095     if(u.u_arg[0] == 0407) {
3096         u.u_arg[2] += u.u_arg[1];
3097         u.u_arg[1] = 0;
3098     } else
3099     if(u.u_arg[0] == 0411)

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

3100      sep++; else
3101      if(u_n_arg[0] != 0410) {
3102          u_n_error = ENOEXEC;
3103      }
3104      if(u_n_arg[1] != 0 && !p->flagTEXT) == 0 && !p->count == 1) {
3105          u_n_error = ETEXTBSY;
3106      }
3107      goto bad;
3108  }
3109  /*
3110  * find text and data sizes
3111  * try them out for possible
3112  * exceed of max sizes
3113  */
3114  ts = (u_n_arg[1]+63)>>6 & 0177;
3115  ds = ((u_n_arg[2]+u_n_arg[3]+63)>>6) & 0177;
3116  if(estabur(ts, ds, SIZE, sep))
3117      goto bad;
3118  /*
3119  * allocate and clear core
3120  * at this point, committed
3121  * to the new image
3122  */
3123  u_n_prof[3] = 0;
3124  xfree();
3125  expand(SIZE);
3126  kalloc(p);
3127  c = SIZE+ds+SIZE;
3128  expand(c);
3129  while(--c >= 0)
3130      clearseg(u_n_procp->p_addr+c);
3131  /* read in data segment */
3132  estabur(0, ds, 0, 0);
3133  u_n_base = 0;
3134  u_n_offset[1] = 020+u_n_arg[1];
3135  u_n_count = u_n_arg[2];
3136  readi(p);
3137  /*
3138  * initialize stack segment
3139  */
3140  u_n_tsiz = ts;
3141  u_n_dsiz = ds;
3142  }
3143  /*
3144  * find text and data sizes
3145  * try them out for possible
3146  * exceed of max sizes
3147  */
3148  ts = (u_n_arg[1]+63)>>6 & 0177;
3149  ds = ((u_n_arg[2]+u_n_arg[3]+63)>>6) & 0177;
3150  if(estabur(ts, ds, SIZE, sep))
3151      goto bad;
3152  /*
3153  * find text and data sizes
3154  * try them out for possible
3155  * exceed of max sizes
3156  */
3157  ts = (u_n_arg[1]+63)>>6 & 0177;
3158  ds = ((u_n_arg[2]+u_n_arg[3]+63)>>6) & 0177;
3159  if(estabur(ts, ds, SIZE, sep))
3160      goto bad;
3161  /*
3162  * set SUID/SGID protections, if no tracing
3163  */
3164  if ((u_n_procp->p_flag&STRC) == 0) {
3165      if(!p->mode&ISUID)
3166          if(u_n_uid != 0) {
3167              u_n_uid = p->uid;
3168              u_n_uid = p->uid;
3169          }
3170      if(!p->mode&ISGID)
3171          if(u_n_gid != 0)
3172              u_n_gid = p->uid;
3173      /* clear sigs, regs, and return */
3174      c = p;
3175      for(p = &u_n_signal[0]; p < &u_n_signal[NSIG]; p++)
3176          if((p->sig & 1) == 0)
3177              *p = 0;
3178      for(cp = &regloc[0]; cp < &regloc[6];)
3179          *cp++ = 0;
3180      u_n_ar0[R7] = 0;
3181      u_n_ar0[*cp++] = 0;
3182      for(ip = &u_n_fsav[0]; ip < &u_n_fsav[25];)
3183          *ip++ = 0;
3184      ip = c;
3185      bad:
3186      iput(p);
3187      prelsz(bp);
3188      if(execnt >= NEXRC)
3189          wakeup(&execnt);
3190      }
3191  }
3192  }
3193  }
3194  }
3195  }
3196  }
3197  }
3198  }
3199  }

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lyons, 1976

```

3150  u_n_size = SIZE;
3151  u_n_sep = sep;
3152  estabur(u_n_tsiz, u_n_dsiz, u_n_ssize, u_n_sep);
3153  cp = bp->b_addr;
3154  ap = -nc - na*2 - 4;
3155  u_n_ar0[R6] = ap;
3156  suword(ap, na);
3157  c = -nc;
3158  while(na--) {
3159      suword(ap+2, c);
3160  }
3161  do
3162      subyte(c++, *cp);
3163  while(*cp++);
3164  suword(ap+2, -1);
3165  /*
3166  * set SUID/SGID protections, if no tracing
3167  */
3168  if ((u_n_procp->p_flag&STRC) == 0) {
3169      if(!p->mode&ISUID)
3170          if(u_n_uid != 0) {
3171              u_n_uid = p->uid;
3172              u_n_uid = p->uid;
3173          }
3174      if(!p->mode&ISGID)
3175          if(u_n_gid != 0)
3176              u_n_gid = p->uid;
3177      /* clear sigs, regs, and return */
3178      c = p;
3179      for(p = &u_n_signal[0]; p < &u_n_signal[NSIG]; p++)
3180          if((p->sig & 1) == 0)
3181              *p = 0;
3182      for(cp = &regloc[0]; cp < &regloc[6];)
3183          *cp++ = 0;
3184      u_n_ar0[R7] = 0;
3185      u_n_ar0[*cp++] = 0;
3186      for(ip = &u_n_fsav[0]; ip < &u_n_fsav[25];)
3187          *ip++ = 0;
3188      ip = c;
3189      bad:
3190      iput(p);
3191      prelsz(bp);
3192      if(execnt >= NEXRC)
3193          wakeup(&execnt);
3194      }
3195  }
3196  }
3197  }
3198  }
3199  }

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lyons, 1976

```

3200 /* ----- */
3201
3202 /* exit system call:
3203  * pass back caller's r0
3204  */
3205 rexit()
3206 {
3207
3208     u.u_arg[0] = u.u_ar0[R0] << 8;
3209     exit();
3210 }
3211 /* ----- */
3212
3213 /* Release resources.
3214  * Save u. area for parent to look at.
3215  * Enter zombie state.
3216  * Wake up parent and init processes,
3217  * and dispose of children.
3218  */
3219 exit()
3220 {
3221     register int *q, a;
3222     register struct proc *p;
3223
3224     u.u_procp->p_flag = & ~STRC;
3225     for(q = &u.u_signal[0]; q < &u.u_signal[NSIG];)
3226         *q++ = 1;
3227     for(q = &u.u_ofile[0]; q < &u.u_ofile[NOFILE]; q++)
3228         if(a = *q) {
3229             *q = NULL;
3230             closef(a);
3231         }
3232     iput(u.u_cdir);
3233     xfree();
3234     a = malloc(swapmap, 1);
3235     if(a == NULL)
3236         panic("out of swap");
3237     p = getblk(swapdev, a);
3238     bcopy(&u, p->b_addr, 256);
3239     bwrite(p);
3240     q = u.u_procp;
3241     mfree(coremap, q->p_size, q->p_addr);
3242     q->p_addr = a;
3243     q->p_stat = SZOMB;
3244
3245     loop:
3246     for(p = &proc[0]; p < &proc[NPROC]; p++)
3247         if(q->p_ppid == p->p_pid) {
3248             wakeup(&proc[1]);
3249             wakeup(p);

```

```

3250         for(p = &proc[0]; p < &proc[NPROC]; p++)
3251             if(q->p_pid == p->p_ppid) {
3252                 p->p_ppid = 1;
3253                 if (p->p_stat == SSTOP)
3254                     setrun(p);
3255             }
3256             switch();
3257             /* no return */
3258         }
3259     q->p_ppid = 1;
3260     goto loop;
3261 }
3262 /* ----- */
3263
3264 /* Wait system call.
3265  * Search for a terminated (zombie) child,
3266  * finally lay it to rest, and collect its status.
3267  * Look also for stopped (traced) children,
3268  * and pass back status from them.
3269  */
3270 wait()
3271 {
3272     register f, *bp;
3273     register struct proc *p;
3274
3275     f = 0;
3276     loop:
3277     for(p = &proc[0]; p < &proc[NPROC]; p++)
3278         if(p->p_ppid == u.u_procp->p_pid) {
3279             f++;
3280             if(p->p_stat == SZOMB) {
3281                 u.u_ar0[R0] = p->p_pid;
3282                 bp = bread(swapdev, f=p->p_addr);
3283                 mfree(swapmap, 1, f);
3284                 p->p_stat = NULL;
3285                 p->p_pid = 0;
3286                 p->p_ppid = 0;
3287                 p->p_sig = 0;
3288                 p->p_ttyp = 0;
3289                 p->p_flag = 0;
3290                 p = bp->b_addr;
3291                 u.u_cstime[0] += p->u_cstime[0];
3292                 dpadd(u.u_cstime, p->u_cstime[1]);
3293                 dpadd(u.u_cstime, p->u_stime);
3294                 u.u_cstime[0] += p->u_cutime[0];
3295                 dpadd(u.u_cutime, p->u_cutime[1]);
3296                 dpadd(u.u_cutime, p->u_otime);
3297                 u.u_ar0[R1] = p->u_arg[0];
3298                 brelse(bp);
3299                 return;

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

3300 }
3301 { if(p->p_stat == SSTOP) {
3302   if((p->p_flag&SWTRD) == 0) {
3303     p->p_flag = | SWTRD;
3304     u_n_ar0[R0] = p->p_pid;
3305     u_n_ar0[R1] = (p->p_sig<8) |
3306     0177;
3307     return;
3308   }
3309   p->p_flag = & ~(STRC|SWTRD);
3310   setrn(p);
3311   }
3312   }
3313   if(f) {
3314     sleep(u_n_procp, WAIT);
3315     goto loop;
3316   }
3317   u_n_error = ECHILD;
3318 }
3319 /* ----- */
3320
3321 /* fork system call. */
3322 fork()
3323 {
3324   register struct proc *p1, *p2;
3325   p1 = u_n_procp;
3326   for(p2 = kproc[0]; p2 > kproc[NPROC]; p2++)
3327     if(p2->p_stat == NULL)
3328       goto found;
3329   u_n_error = EAGAIN;
3330   goto out;
3331 }
3332 found:
3333   {
3334     if(newproc()) {
3335       u_n_ar0[R0] = p1->p_pid;
3336       u_n_cstime[0] = 0;
3337       u_n_cstime[1] = 0;
3338       u_n_stime = 0;
3339       u_n_cutime[0] = 0;
3340       u_n_cutime[1] = 0;
3341       u_n_utime = 0;
3342       return;
3343     }
3344     u_n_ar0[R0] = p2->p_pid;
3345   }
3346 out:
3347   u_n_ar0[R7] = + 2;
3348 }
3349 /* ----- */

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

3350 /* break system call.
3351 * -- bad planning: "break" is a dirty word in C.
3352 */
3353 sbreak()
3354 {
3355   register a, n, d;
3356   int i;
3357   /* set n to new data size
3358   * set d to new-old
3359   * set n to new total size
3360   */
3361   * set n to new total size
3362   /*
3363   n = ((u_n_arg[0]+63)>6) & 0177;
3364   if(!u_n_sep)
3365     n = - nseg(u_n_tsize) * 128;
3366   if(n > 0)
3367     n = 0;
3368   d = n - u_n_dsize;
3369   n = n + USIZE+u_n_ssize;
3370   if(estabur(u_n_tsize, u_n_dsize+d, u_n_ssize, u_n_sep))
3371     return;
3372   u_n_dsize =+ d;
3373   if(d > 0)
3374     goto bigger;
3375   a = u_n_procp->p_addr + n - u_n_ssize;
3376   i = n;
3377   n = u_n_ssize;
3378   while(n--) {
3379     copyseg(a-d, a);
3380     a++;
3381   }
3382   expand(i);
3383   return;
3384 }
3385 bigger:
3386   expand(n);
3387   a = u_n_procp->p_addr + n;
3388   n = u_n_ssize;
3389   while(n--) {
3390     a--;
3391     copyseg(a-d, a);
3392   }
3393   while(d--)
3394     clearseg(--a);
3395 }
3396 /* ----- */
3397
3398
3399

```

```

3400 #
3401 /*
3402  * Everything in this file is
3403  * a routine implementing a system call.
3404  */
3405
3406 #include "../param.h"
3407 #include "../user.h"
3408 #include "../reg.h"
3409 #include "../inode.h"
3410 #include "../system.h"
3411 #include "../proc.h"
3412
3413 getswit()
3414 {
3415     u.u_ar0[R0] = SW->integ;
3416 }
3417 /* ----- */
3418
3419 gtime()
3420 {
3421     u.u_ar0[R0] = time[0];
3422     u.u_ar0[R1] = time[1];
3423 }
3424 /* ----- */
3425
3426 stime()
3427 {
3428     if(suser()) {
3429         time[0] = u.u_ar0[R0];
3430         time[1] = u.u_ar0[R1];
3431         wakeup(tout);
3432     }
3433 }
3434 /* ----- */
3435
3436 setuid()
3437 {
3438     register uid;
3439     uid = u.u_ar0[R0].lobyte;
3440     if(u.u_ruid == uid.lobyte || suser()) {
3441         u.u_uid = uid;
3442         u.u_procp->p_uid = uid;
3443         u.u_ruid = uid;
3444     }
3445 }

```

```

3450 /* ----- */
3451
3452 getuid()
3453 {
3454     u.u_ar0[R0].lobyte = u.u_ruid;
3455     u.u_ar0[R0].hibyte = u.u_uid;
3456 }
3457 /* ----- */
3458
3459 setgid()
3460 {
3461     register gid;
3462     gid = u.u_ar0[R0].lobyte;
3463     if(u.u_rgid == gid.lobyte || suser()) {
3464         u.u_gid = gid;
3465         u.u_rgid = gid;
3466     }
3467 }
3468 /* ----- */
3469
3470 getgid()
3471 {
3472     u.u_ar0[R0].lobyte = u.u_rgid;
3473     u.u_ar0[R0].hibyte = u.u_gid;
3474 }
3475 /* ----- */
3476
3477 getpid()
3478 {
3479     u.u_ar0[R0] = u.u_procp->p_pid;
3480 }
3481 /* ----- */
3482
3483 sync()
3484 {
3485     update();
3486 }
3487 /* ----- */
3488
3489 nice()
3490 {
3491     register n;
3492     n = u.u_ar0[R0];
3493     if(n > 20)
3494         n = 20;

```

```

3500 if(n < 0 && isuser())
3501     n = 0;
3502     u.n_procp->p_nice = n;
3503 }
3504 /* ----- */
3505 /*
3506 * Unlink system call.
3507 * panic: unlink -- "cannot happen"
3508 */
3509 /*
3510 unlink()
3511 {
3512     register *ip, *pp;
3513     extern uchar;
3514     pp = namei(kuchar, 2);
3515     if(pp == NULL)
3516         return;
3517     prele(ip);
3518     ip = iget(pp->i_dev, u.n_dent.u_ino);
3519     if(ip == NULL)
3520         panic("unlink -- iaget");
3521     if(ip->i_mode&IFMT)==IFDIR && isuser())
3522         goto out;
3523     u.n_offset[1] = - DIRSIZ+2;
3524     u.n_base = &u.n_dent;
3525     u.n_count = DIRSIZ+2;
3526     u.n_dent.u_ino = 0;
3527     write(ip);
3528     ip->i_nlink--;
3529     ip->i_flag |= IUPD;
3530     ip->i_flag = | IUPD;
3531 out:
3532     out:
3533     iput(ip);
3534     iput(ip);
3535 }
3536 /* ----- */
3537 chdir()
3538 {
3539     register *ip;
3540     extern uchar;
3541     ip = namei(kuchar, 0);
3542     if(ip == NULL)
3543         return;
3544     if(ip->i_mode&IFMT) != IFDIR)
3545         return;
3546     if(ip->i_nerror = ENOTDIR;
3547     bad:
3548     iput(ip);
3549

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

3550     return;
3551 }
3552 if(access(ip, IXRC))
3553     goto bad;
3554 iput(u.n_cdir);
3555     u.n_cdir = ip;
3556     prele(ip);
3557 }
3558 /* ----- */
3559 chmod()
3560 {
3561     register *ip;
3562     if(ip == owner()) == NULL)
3563         return;
3564     ip->i_mode = &~0777;
3565     if(u.n_uid)
3566         return;
3567     if(u.n_arg[1] & ISVTX;
3568     ip->i_mode = | u.n_arg[1]&0777;
3569     ip->i_flag = | IUPD;
3570     iput(ip);
3571 }
3572 /* ----- */
3573 chown()
3574 {
3575     register *ip;
3576     if(isuser() || ip == owner()) == NULL)
3577         return;
3578     ip->i_uid = u.n_arg[1].lobyte;
3579     ip->i_gid = u.n_arg[1].hbyte;
3580     ip->i_flag = | IUPD;
3581     iput(ip);
3582 }
3583 /* ----- */
3584 Change modified date of file:
3585 * time to r0-r1; sys smdate; file
3586 * This call has been withdrawn because it messes up
3587 * incremental dumps (pseudo-old files aren't dumped).
3588 * It works though and you can uncomment it if you like.
3589 smdate()
3590 {
3591     register struct inode *ip;
3592     int tbuf[2];
3593     register int *tp;
3594     iput(ip);
3595

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976



```

3600
3601 if ((ip = owner()) == NULL)
3602     return;
3603 ip->i_flag |= IUPD;
3604 tp = &tbu[2];
3605 *--tp = u.u_ar0[R1];
3606 *--tp = u.u_ar0[R0];
3607 iupdat(ip, tp);
3608 ip->i_flag |= ~IUPD;
3609 iput(ip);
3610 }
3611 */
3612 /* ----- */
3613
3614 ssig()
3615 {
3616     register a;
3617
3618     a = u.u_arg[0];
3619     if(a<=0 || a>=NSIG || a==SIGKIL) {
3620         u.u_error = EINVAL;
3621         return;
3622     }
3623     u.u_ar0[R0] = u.u_signal[a];
3624     u.u_signal[a] = u.u_arg[1];
3625     if(u.u_procp->p_sig == a)
3626         u.u_procp->p_sig = 0;
3627 }
3628 /* ----- */
3629
3630 kill()
3631 {
3632     register struct proc *p, *q;
3633     register a;
3634     int f;
3635
3636     f = 0;
3637     a = u.u_ar0[R0];
3638     q = u.u_procp;
3639     for(p = &proc[0]; p < &proc[NPROC]; p++) {
3640         if(p == q)
3641             continue;
3642         if(a != 0 && p->p_pid != a)
3643             continue;
3644         if(a==0&&(p->p_ttyp!=q->p_ttyp|p<=&proc[1]))
3645             continue;
3646         if(u.u_uid != 0 && u.u_uid != p->p_uid)
3647             continue;
3648         f++;
3649         psignal(p, u.u_arg[0]);

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

3650     }
3651     if(f == 0)
3652         u.u_error = ESRCH;
3653 }
3654 /* ----- */
3655
3656 times()
3657 {
3658     register *p;
3659
3660     for(p = &u.u_otime; p < &u.u_utime+6;) {
3661         suword(u.u_arg[0], *p++);
3662         u.u_arg[0] += 2;
3663     }
3664 }
3665 /* ----- */
3666
3667 profil()
3668 {
3669     u.u_prof[0] = u.u_arg[0] & ~1; /* base of sample buf */
3670     u.u_prof[1] = u.u_arg[1]; /* size of same */
3671     u.u_prof[2] = u.u_arg[2]; /* pc offset */
3672     u.u_prof[3] = (u.u_arg[3]>>1) & 077777; /* pc scale */
3673 }
3674 /* ----- */
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3699

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

3700 #
3701 #include "../param.h"
3702 #include "../system.h"
3703 #include "../user.h"
3704 #include "../proc.h"
3705 #define UMODE 0170000
3706 #define SCHMAG 10
3707 #define
3708
3709 /*
3710 * clock is called straight from
3711 * the real time clock interrupt.
3712 *
3713 * functions:
3714 * reptime clock
3715 * copy *swtches to display
3716 * implement callouts
3717 * maintain user/system times
3718 * maintain date
3719 * profile
3720 * tout wakeup (sys sleep)
3721 * lightning bolt wakeup (every 4 sec)
3722 * alarm clock signals
3723 * jab the scheduler
3724 */
3725 clock(dev, sp, r1, nps, r0, pc, ps)
3726 {
3727     register struct callout *p1, *p2;
3728     register struct proc *pp;
3729
3730 /*
3731 * restart clock
3732 */
3733 * lks = 0115;
3734
3735 /*
3736 */
3737 * display register
3738 */
3739 display();
3740
3741 /*
3742 */
3743 * callouts
3744 * if done, just return
3745 * else update first non-zero time
3746 */
3747 if(callout[0].c_func == 0)
3748     goto out;
3749

```

Reproduced under license from the Western Electric Company, NY

Copyright, J. Lions, 1976

```

3750 p2 = callout[0];
3751 while(p2->c_time<=0 && p2->c_func!=0)
3752     p2++;
3753 p2->c_time--;
3754
3755 /*
3756 * if ps is high, just return
3757 */
3758
3759 if((ps&0340) != 0)
3760     goto out;
3761
3762 /*
3763 * callout
3764 */
3765 sp15();
3766
3767 if(callout[0].c_time <= 0) {
3768     p1 = callout[0];
3769     while(p1->c_func != 0 && p1->c_time <= 0) {
3770         (*p1->c_func)(p1->c_arg);
3771         p1++;
3772     }
3773     p2 = callout[0];
3774     while(p2->c_func = p1->c_func) {
3775         p2->c_time = p1->c_time;
3776         p2->c_arg = p1->c_arg;
3777         p1++;
3778         p2++;
3779     }
3780 }
3781 /*
3782 */
3783 * lightning bolt time-out
3784 * and time of day
3785
3786 out:
3787 if((ps&UMODE) == UMODE) {
3788     u.n_utime++;
3789     if(u.n_prof[3])
3790         incupc(ps, u.n_prof);
3791 } else
3792     u.n_stime++;
3793
3794 pp = u.n_proc;
3795 if(++pp->cpu == 0)
3796     pp->cpu--;
3797 if(++bolt >= HZ)
3798     if((ps&0340) != 0)
3799         return;

```

Reproduced under license from the Western Electric Company, NY

Copyright, J. Lions, 1976

```

3800         lbolt =- HZ;
3801         if(++time[1] == 0)
3802             ++time[0];
3803         spl1();
3804         if(time[1]==tout[1] && time[0]==tout[0])
3805             wakeup(tout);
3806         if((time[1]&03) == 0) {
3807             runrun++;
3808             wakeup(&lbolt);
3809         }
3810         for(pp = &proc[0]; pp < &proc[NPROC]; pp++)
3811             if (pp->p_stat) {
3812                 if(pp->p_time != 127)
3813                     pp->p_time++;
3814                 if((pp->p_cpu & 0377) > SCHMAG)
3815                     pp->p_cpu -= SCHMAG; else
3816                     pp->p_cpu = 0;
3817                 if(pp->p_pri > PUSER)
3818                     setpri(pp);
3819             }
3820         if(runin!=0) {
3821             runin = 0;
3822             wakeup(&runin);
3823         }
3824         if((ps&UMODE) == UMODE) {
3825             u.u_ar0 = &r0;
3826             if(!issig())
3827                 psig();
3828             setpri(u.u_proc);
3829         }
3830     }
3831 }
3832 /* ----- */
3833
3834 /*
3835  * timeout is called to arrange that
3836  * fun(arg) is called in tim/HZ seconds.
3837  * An entry is sorted into the callout
3838  * structure. The time in each structure
3839  * entry is the number of HZ's more
3840  * than the previous entry.
3841  * In this way, decrementing the
3842  * first entry has the effect of
3843  * updating all entries.
3844  */
3845 timeout(fun, arg, tim)
3846 {
3847     register struct callout *p1, *p2;
3848     register t;
3849     int s;

```

```

3850
3851     t = tim;
3852     s = PS->integ;
3853     p1 = &callout[0];
3854     spl7();
3855     while(p1->c_func != 0 && p1->c_time <= t) {
3856         t =- p1->c_time;
3857         p1++;
3858     }
3859     p1->c_time =- t;
3860     p2 = p1;
3861     while(p2->c_func != 0)
3862         p2++;
3863     while(p2 >= p1) {
3864         (p2+1)->c_time = p2->c_time;
3865         (p2+1)->c_func = p2->c_func;
3866         (p2+1)->c_arg = p2->c_arg;
3867         p2--;
3868     }
3869     p1->c_time = t;
3870     p1->c_func = fun;
3871     p1->c_arg = arg;
3872     PS->integ = s;
3873 }
3874 /* ----- */
3875
3876
3877
3878
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3899

```

```

3900 #
3901 /*
3902 */
3903
3904 #include "../param.h"
3905 #include "../system.h"
3906 #include "../user.h"
3907 #include "../proc.h"
3908 #include "../inode.h"
3909 #include "../reg.h"
3910
3911 /*
3912 * priority for tracing
3913 */
3914 #define IPCPRI (-1)
3915
3916 /*
3917 * structure to access an array of integers.
3918 */
3919 struct
3920 {
3921     int     inta[];
3922 };
3923 /* ----- */
3924
3925 /*
3926 * Tracing variables.
3927 * Used to pass trace command from
3928 * parent to child being traced.
3929 * This data base cannot be
3930 * shared and is locked
3931 * per user.
3932 */
3933 struct
3934 {
3935     int     ip_lock;
3936     int     ip_req;
3937     int     ip_addr;
3938     int     ip_data;
3939 } ipc;
3940 /* ----- */
3941
3942 /*
3943 * Send the specified signal to
3944 * all processes with 'tp' as its
3945 * controlling teletype.
3946 * Called by tty.c for quits and
3947 * interrupts.
3948 */
3949 signal(tp, sig)

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

3950 {
3951     register struct proc *p;
3952
3953     for (p = kproc[0]; p > kproc[NPROC]; p++)
3954         if (p->p_ttyp == tp)
3955             psignal(p, sig);
3956 }
3957 /* ----- */
3958
3959 /*
3960 * Send the specified signal to
3961 * the specified process.
3962 */
3963 psignal(p, sig)
3964 int *p;
3965 {
3966     register *rp;
3967
3968     if (sig >= NSIG)
3969         return;
3970     rp = p;
3971     if (rp->p_sig != SIGKILL)
3972         rp->p_sig = sig;
3973     if (rp->p_stat > PUSER)
3974         rp->p_stat = PUSER;
3975     if (rp->p_stat == SWAIT)
3976         setrun(rp);
3977 }
3978 /* ----- */
3979
3980 /*
3981 * Returns true if the current
3982 * process has a signal to process.
3983 * This is asked at least once
3984 * each time a process enters the
3985 * system.
3986 * A signal does not do anything
3987 * directly to a process; it sets
3988 * a flag that asks the process to
3989 * do something to itself.
3990 */
3991 isig()
3992 {
3993     register n;
3994     register struct proc *p;
3995
3996     p = u_proc;
3997     {
3998         if (n = p->p_sig)
3999             if (p->p_flag&STRC)
4000                 stop();

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

4000             if ((n = p->p_sig) == 0)
4001                 return(0);
4002         }
4003         if((u.u_signal[n]&1) == 0)
4004             return(n);
4005     }
4006     return(0);
4007 }
4008 /* ----- */
4009
4010 /*
4011  * Enter the tracing STOP state.
4012  * In this state, the parent is
4013  * informed and the process is able to
4014  * receive commands from the parent.
4015  */
4016 stop()
4017 {
4018     register struct proc *pp, *cp;
4019
4020 loop:
4021     cp = u.u_procp;
4022     if(cp->p_pid != 1)
4023         for (pp = &proc[0]; pp < &proc[NPROC]; pp++)
4024             if (pp->p_pid == cp->p_ppid) {
4025                 wakeup(pp);
4026                 cp->p_stat = SSTOP;
4027                 swtch();
4028                 if ((cp->p_flag&STRC)==0 || procxmt())
4029                     return;
4030                 goto loop;
4031             }
4032     exit();
4033 }
4034 /* ----- */
4035
4036 /*
4037  * Perform the action specified by
4038  * the current signal.
4039  * The usual sequence is:
4040  * if(issig())
4041  *     psig();
4042  */
4043 psig()
4044 {
4045     register n, p;
4046     register *rp;
4047
4048     rp = u.u_procp;
4049     n = rp->p_sig;

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

4050     rp->p_sig = 0;
4051     if((p=u.u_signal[n]) != 0) {
4052         u.u_error = 0;
4053         if(n != SIGINS && n != SIGTRC)
4054             u.u_signal[n] = 0;
4055         n = u.u_ar0[R6] - 4;
4056         grow(n);
4057         suword(n+2, u.u_ar0[RPS]);
4058         suword(n, u.u_ar0[R7]);
4059         u.u_ar0[R6] = n;
4060         u.u_ar0[RPS] = & ~TBIT;
4061         u.u_ar0[R7] = p;
4062         return;
4063     }
4064     switch(n) {
4065
4066     case SIGQUIT:
4067     case SIGINS:
4068     case SIGTRC:
4069     case SIGIOT:
4070     case SIGEMT:
4071     case SIGFPT:
4072     case SIGBUS:
4073     case SIGSEGV:
4074     case SIGSYS:
4075         u.u_arg[0] = n;
4076         if(core())
4077             n =+ 0200;
4078     }
4079     u.u_arg[0] = (u.u_ar0[R0]<<8) | n;
4080     exit();
4081 }
4082 /* ----- */
4083
4084 /*
4085  * Create a core image on the file "core"
4086  * If you are looking for protection glitches,
4087  * there are probably a wealth of them here
4088  * when this occurs to a suid command.
4089  *
4090  * It writes USIZE block of the
4091  * user.h area followed by the entire
4092  * data+stack segments.
4093  */
4094 core()
4095 {
4096     register s, *ip;
4097     extern schar;
4098
4099     u.u_error = 0;

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

4100 n.u_dirp = "core";
4101 ip = namei(&sch, 1);
4102 if(ip == NULL) {
4103     return(0);
4104 }
4105 ip = maknode(0666);
4106 if(ip == NULL)
4107     return(0);
4108 }
4109 if(!access(ip, IWRTX) &&
4110 (ip-<imode&IWM) == 0 &&
4111 n.u_nuid == n.u_xuid) {
4112     trunc(ip);
4113     n.u_offset[0] = 0;
4114     n.u_offset[1] = 0;
4115     n.u_base = &u;
4116     n.u_count = USIZE*64;
4117     n.u_segflg = 1;
4118     writel(ip);
4119     s = n.u_procp->p_size - USIZE;
4120     estabur(0, s, 0, 0);
4121     n.u_base = 0;
4122     n.u_count = s*64;
4123     n.u_segflg = 0;
4124     writel(ip);
4125 }
4126 input(ip);
4127 return(n.u_error==0);
4128 }
4129 /* ----- */
4130
4131 /*
4132 * grow the stack to include the SP
4133 * true return in successful.
4134 */
4135
4136 grow(sp)
4137 char *sp;
4138 {
4139     register a, si, i;
4140     if(sp >= -n.u_size*64)
4141         return(0);
4142     return(0);
4143     si = ldv(-sp, 64) - n.u_size + SINCR;
4144     if(si <= 0)
4145         return(0);
4146     if(estabur(n.u_tsize, n.u_dsize, n.u_ssize+si, n.u_sep))
4147         return(0);
4148     expand(n.u_procp->p_size+si);
4149     a = n.u_procp->p_addr + n.u_procp->p_size;

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

4150 for(i=n.u_size; i; i--) {
4151     a--;
4152     copyseg(a-si, a);
4153 }
4154 for(i=si; i; i--)
4155     clearseg(--a);
4156 n.u_size += si;
4157 return(i);
4158 }
4159 /* ----- */
4160
4161 /*
4162 * sys-trace system call.
4163 */
4164 ptrace()
4165 {
4166     register struct proc *p;
4167     if (n.u_arg[2] <= 0) {
4168         n.u_procp->p_flag |= STRC;
4169         return;
4170     }
4171     for (p=procp; p < &procp[NPROC]; p++)
4172         if (p->p_stat==SSTOP
4173             && p->p_pid==n.u_arg[0]
4174             && p->p_pid==n.u_arg[1])
4175             goto found;
4176     n.u_error = ESRCH;
4177     return;
4178 }
4179 /* ----- */
4180
4181 found:
4182     while (ipc_lock)
4183         sleep(&ipc, IPCPRI);
4184     ipc_lock = p->p_pid;
4185     ipc_data = n.u_arg[0];
4186     ipc_addr - n.u_arg[1] & ~01;
4187     ipc_req = n.u_arg[2];
4188     p->p_flag &~ SWTMD;
4189     setrn(p);
4190     while (ipc_req > 0)
4191         sleep(&ipc, IPCPRI);
4192     u_n_ar0[R0] = ipc_data;
4193     if (ipc_req < 0)
4194         n.u_error = EIO;
4195     ipc_lock = 0;
4196     wakeup(&ipc);
4197 }
4198 /* ----- */
4199 */

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

4200 * Code that the child process
4201 * executes to implement the command
4202 * of the parent process in tracing.
4203 */
4204 procxmt()
4205 {
4206     register int i;
4207     register int *p;
4208
4209     if (ipc.ip_lock != u.u_procp->p_pid)
4210         return(0);
4211     i = ipc.ip_req;
4212     ipc.ip_req = 0;
4213     wakeup(&ipc);
4214     switch (i) {
4215
4216     /* read user I */
4217     case 1:
4218         if (fuibyte(ipc.ip_addr) == -1)
4219             goto error;
4220         ipc.ip_data = fuiword(ipc.ip_addr);
4221         break;
4222
4223     /* read user D */
4224     case 2:
4225         if (fubyte(ipc.ip_addr) == -1)
4226             goto error;
4227         ipc.ip_data = fuword(ipc.ip_addr);
4228         break;
4229
4230     /* read u */
4231     case 3:
4232         i = ipc.ip_addr;
4233         if (i < 0 || i >= (USIZE<<6))
4234             goto error;
4235         ipc.ip_data = u.inta[i>>1];
4236         break;
4237
4238     /* write user I (for now, always an error) */
4239     case 4:
4240         if (suiword(ipc.ip_addr, 0) < 0)
4241             goto error;
4242         suiword(ipc.ip_addr, ipc.ip_data);
4243         break;
4244
4245     /* write user D */
4246     case 5:
4247         if (suword(ipc.ip_addr, 0) < 0)
4248             goto error;
4249         suword(ipc.ip_addr, ipc.ip_data);

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

4250         break;
4251
4252     /* write u */
4253     case 6:
4254         p = &u.inta[ipc.ip_addr>>1];
4255         if (p >= u.u_fsav && p < &u.u_fsav[25])
4256             goto ok;
4257         for (i=0; i<9; i++)
4258             if (p == &u.u_ar0[regloc[i]])
4259                 goto ok;
4260         goto error;
4261     ok:
4262         if (p == &u.u_ar0[RPS]) {
4263             /* assure user space */
4264             ipc.ip_data = | 0170000;
4265             /* priority 0 */
4266             ipc.ip_data = & ~0340;
4267         }
4268         *p = ipc.ip_data;
4269         break;
4270
4271     /* set signal and continue */
4272     case 7:
4273         u.u_procp->p_sig = ipc.ip_data;
4274         return(1);
4275
4276     /* force exit */
4277     case 8:
4278         exit();
4279
4280     default:
4281     error:
4282         ipc.ip_req = -1;
4283     }
4284     return(0);
4285 }
4286 /* ----- */
4287
4288
4289
4290
4291
4292
4293
4294
4295
4296
4297
4298
4299

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976





# 3

**Program Swapping  
Basic Input/Output  
Block Devices**



```

4300 /*
4301  * Text structure.
4302  * One allocated per pure
4303  * procedure on swap device.
4304  * Manipulated by text.c
4305  */
4306 struct text
4307 {
4308     int      x_daddr;      /* disk address of segment */
4309     int      x_caddr;      /* core address, if loaded */
4310     int      x_size; /* size (*64) */
4311     int      *x_iptr;      /* inode of prototype */
4312     char     x_count;      /* reference count */
4313     char     x_ccount;     /* number of loaded references */
4314 } text[NTEXT];
4315 /* ----- */
4316
4317
4318
4319
4320
4321
4322
4323
4324
4325
4326
4327
4328
4329
4330
4331
4332
4333
4334
4335
4336
4337
4338
4339
4340
4341
4342
4343
4344
4345
4346
4347
4348
4349

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

4350 #
4351 #include "../param.h"
4352 #include "../system.h"
4353 #include "../user.h"
4354 #include "../proc.h"
4355 #include "../text.h"
4356 #include "../inode.h"
4357
4358 /* Swap out process p.
4359  * The ff flag causes its core to be freed--
4360  * it may be off when called to create an image for a
4361  * child process in newproc.
4362  * Os is the old size of the data area of the process,
4363  * and is supplied during core expansion swaps.
4364  *
4365  * panic: out of swap space
4366  * panic: swap error -- IO error
4367  */
4368 xswap(p, ff, os)
4369 int *p;
4370 { register *rp, a;
4371
4372     rp = p;
4373     if(os == 0)
4374         os = rp->p_size;
4375     a = malloc(swapmap, (rp->p_size+7)/8);
4376     if(a == NULL)
4377         panic("out of swap space");
4378     xccdec(rp->p_textp);
4379     rp->p_flag |= SLOCK;
4380     if(swap(a, rp->p_addr, os, 0))
4381         panic("swap error");
4382     if(ff)
4383         mfree(coremap, os, rp->p_addr);
4384     rp->p_addr = a;
4385     rp->p_flag = & ~(SLOAD|SLOCK);
4386     rp->p_time = 0;
4387     if(runout) {
4388         runout = 0;
4389         wakeup(&runout);
4390     }
4391 }
4392 /* ----- */
4393
4394 /*
4395  * relinquish use of the shared text segment
4396  * of a process.
4397  */
4398 xfree()
4399 { register *xp, *ip;

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

4450 }
4451 if((xp=rp) == NULL) panic("out of text");
4452 xp-<x_count = 1;
4453 xp-<x_count = 0;
4454 xp-<x_ptr = ip;
4455 ts = ((u.n_arg[1]+63)>>6) & 0177;
4456 xp-<x_size = ts;
4457 if((xp->x_daddr = malloc(swapmap, (ts+7)/8)) == NULL)
4458 panic("out of swap space");
4459 expand(USIZE+ts);
4460 estabur(0, ts, 0, 0);
4461 u.n_count = u.n_arg[1];
4462 u.n_offset[1] = 020;
4463 u.n_base = 0;
4464 readl(ip);
4465 rp = u.n_procp;
4466 rp->p_flag = | SLOCK;
4467 swap(xp->x_daddr, rp->p_daddr+USIZE, ts, 0);
4468 rp->p_flag = &~SLOCK;
4469 rp->p_textp = xp;
4470 rp = ip;
4471 rp->i_flag = | ITEXT;
4472 rp-<i_count++;
4473 expand(USIZE);
4474 out:
4475 if(xp->x_count == 0) {
4476 savu(u.n_rsav);
4477 savu(u.n_ssav);
4478 xswap(u.n_procp, 1, 0);
4479 u.n_procp->p_flag = | SSWAP;
4480 swch();
4481 /* no return */
4482 }
4483 xp-<x_count++;
4484 }
4485 /* ----- */
4486 register struct text *xp;
4487 register *rp, *ts;
4488 /* Decrement the in-core usage count of a shared text
4489 segment. When it drops to zero, free the core space.
4490 */
4491 xcdec(xp);
4492 int *xp;
4493 }
4494 register *rp;
4495 if((rp=xp)!=NULL && rp->x_count!=0)
4496 if(--rp->x_count == 0)
4497 mfree(coremap, rp->x_size, rp->x_caddr);
4498 }
4499

```

```

4400
4401 if((xp=u.n_procp->p_textp) != NULL) {
4402 u.n_procp->p_textp = NULL;
4403 xcdec(xp);
4404 if(--xp->x_count == 0) {
4405 ip = xp->x_ptr;
4406 if((ip->i_mod&ISVIX) == 0) {
4407 xp->x_ptr = NULL;
4408 mfree(swapmap, (xp->x_size+7)/8,
4409 xp->x_daddr);
4410 ip->i_flag = &~ITEXT;
4411 iput(ip);
4412 }
4413 }
4414 }
4415 }
4416 /* ----- */
4417 /* Attach to a shared text segment.
4418 * If there is no shared text, just return.
4419 * If there is, hook up to it:
4420 * If it is not currently being used, it has to be read
4421 * in from the inode (ip) and established in the swap space.
4422 * If it is being used, but not currently in core,
4423 * a swap has to be done to get it back.
4424 * The full coroutine glory has to be invoked--
4425 * see sfp.c-- because if the calling process
4426 * is misplaced in core the text image might not fit.
4427 * Quite possibly the code after "out:" could check to
4428 * see if the text does fit and simply swap it in.
4429 *
4430 * panic: out of swap space
4431 */
4432 * /
4433 xalloc(ip)
4434 int *ip;
4435 }
4436 register struct text *xp;
4437 register *rp, *ts;
4438 if(u.n_arg[1] == 0) return;
4439 rp = NULL;
4440 for(xp = &text[0]; xp < &text[TEXT]; xp++)
4441 if(xp->x_ptr == NULL)
4442 if(ip=>i_ptr == 0) {
4443 if(ip == NULL)
4444 rp = xp;
4445 } else {
4446 if(xp->x_ptr == ip) {
4447 xp->x_ptr = NULL;
4448 } else {
4449 goto out;
4450

```

```

4500 /*
4501 * Each buffer in the pool is usually doubly linked into two
4502 * lists: for the device with which it is currently associat-
4503 * ed (always) and also for a list of blocks available for
4504 * allocation for other use (usually).
4505 * The latter list is kept in last-used order, and the two
4506 * lists are doubly linked to make it easy to remove
4507 * a buffer from one list when it was found by
4508 * looking through the other.
4509 * A buffer is on the available list, and is liable
4510 * to be reassigned to another disk block, if and only
4511 * if it is not marked BUSY. When a buffer is busy, the
4512 * available-list pointers can be used for other purposes.
4513 * Most drivers use the forward ptr as a link in their I/O
4514 * active queue.
4515 * A buffer header contains all the information required
4516 * to perform I/O.
4517 * Most of the routines which manipulate these things
4518 * are in bio.c.
4519 */
4520 struct buf
4521 {
4522     int     b_flags;          /* see defines below */
4523     struct buf *b_forw;      /* headed by devtab of b_dev */
4524     struct buf *b_back;     /* " */
4525     struct buf *av_forw;    /* position on free list, */
4526     struct buf *av_back;    /* if not BUSY*/
4527     int     b_dev;          /* major+minor device name */
4528     int     b_wcount;       /* transfer count (usu. words) */
4529     char    *b_addr;        /* low order core address */
4530     char    *b_xmem;        /* high order core address */
4531     char    *b_blkno;       /* block # on device */
4532     char    b_error;        /* returned after I/O */
4533     char    *b_resid;       /* words not transferred after
4534                               error */
4535 } buf[NBUF];
4536 /* ----- */
4537
4538 /*
4539 * Each block device has a devtab, which contains private
4540 * state stuff and 2 list heads: the b_forw/b_back list,
4541 * which is doubly linked and has all the buffers currently
4542 * associated with the major device;
4543 * and the d_actf/d_actl list, which is private to the
4544 * device but in fact is always used for the head and tail
4545 * of the I/O queue for the device.
4546 * Various routines in bio.c look at b_forw/b_back
4547 * (notice they are the same as in the buf structure)
4548 * but the rest is private to each device driver.
4549 */

```

```

4550
4551 struct devtab
4552 {
4553     char    d_active;       /* busy flag */
4554     char    d_errcnt;       /* error count (for recovery)*/
4555     struct buf *b_forw;    /* first buffer for this dev */
4556     struct buf *b_back;    /* last buffer for this dev */
4557     struct buf *d_actf;    /* head of I/O queue */
4558     struct buf *d_actl;    /* tail of I/O queue */
4559 };
4560 /* ----- */
4561
4562 /*
4563 * This is the head of the queue of available
4564 * buffers-- all unused except for the 2 list heads.
4565 */
4566
4567 struct     buf bfreelist;
4568
4569 /*
4570 * These flags are kept in b_flags.
4571 */
4572 #define B_WRITE 0          /* non-read pseudo-flag */
4573 #define B_READ 01         /* read when I/O occurs */
4574 #define B_DONE 02         /* transaction finished */
4575 #define B_ERROR 04        /* transaction aborted */
4576 #define B_BUSY 010        /* not on av_forw/back list */
4577 #define B_PHYS 020        /* Physical IO potentially
4578                               using the Unibus map */
4579 #define B_MAP 040         /* This block has the UNIBUS
4580                               map allocated */
4581 #define B_WANTED 0100     /* issue wakeup when
4582                               BUSY goes off */
4583 #define B_RELOC 0200      /* no longer used */
4584 #define B_ASYNC 0400      /* don't wait wait for I/O
4585                               completion */
4586 #define B_DELWRI 01000    /* don't write till block
4587                               leaves available list */
4588
4589
4590
4591
4592
4593
4594
4595
4596
4597
4598
4599

```

```

4600 /* Used to dissect integer device code
4601 * into major (driver designation) and
4602 * minor (driver parameter) parts.
4603 */
4604 struct
4605     char d_minor;
4606     char d_major;
4607 };
4608 /* -----
4609 /* Declaration of block device
4610 * switch. Each entry (row) is
4611 * the only link between the
4612 * main unix code and the driver.
4613 * The initialization of the
4614 * device switches is in the
4615 * file conf.c.
4616 */
4617 struct ddevsw
4618     {
4619     int (*d_open)();
4620     int (*d_close)();
4621     int (*d_strategy)();
4622     } ddevsw[];
4623 /* -----
4624 /* Nblkdev is the number of entries
4625 * (rows) in the block switch. It is
4626 * set in binit/bio.c by making
4627 * a pass over the switch.
4628 * Used in bounds checking on major
4629 * device numbers.
4630 */
4631 int nblkdev;
4632 /* Character device switch.
4633 */
4634 struct cdevsw
4635     {
4636     int (*d_open)();
4637     int (*d_close)();
4638     int (*d_read)();
4639     int (*d_write)();
4640     int (*d_sgtty)();
4641     } cdevsw[];
4642 /* -----
4643 /* Number of character switch entries.
4644 */
4645 * Set by cinit/tty.c
4646 */
4647 int nchrdev;
4648
4649
4650

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

4650 /*
4651 * this file is created, along with the file "low.s",
4652 * by the program "mkconf.c", to reflect the actual
4653 * configuration of peripheral devices on a system.
4654 */
4655 int (*bdevsw[])()
4656 {
4657     knldev, knldev, krsstrategy, krtab, /*rk */
4658     knodev, knodev, 0, /*rp */
4659     knodev, knodev, 0, /*rt */
4660     knodev, knodev, 0, /*tm */
4661     knodev, knodev, 0, /*tc */
4662     knodev, knodev, 0, /*hs */
4663     knodev, knodev, 0, /*hp */
4664     knodev, knodev, 0, /*ht */
4665     0
4666 };
4667
4668
4669 int (*cdevsw[])()
4670 {
4671     klopen, kklclose, kklread, kklwrite, kklsgtty,
4672     /* console */
4673     kpcopen, kpclose, kpcread, kpcwrite, knodev,
4674     /* pc */
4675     klopen, klpclose, klpwrite, knodev,
4676     /* lp */
4677     knodev, knodev, knodev, knodev, /*dc */
4678     knodev, knodev, knodev, knodev, /*dh */
4679     knodev, knodev, knodev, knodev, /*dp */
4680     knodev, knodev, knodev, knodev, /*d】 */
4681     knodev, knodev, knodev, knodev, /*dn */
4682     knldev, knldev, kmmread, kmmwrite, knodev,
4683     /* mem */
4684     knldev, knldev, kkrread, kkrwrite, knodev,
4685     /* rk */
4686     knodev, knodev, knodev, knodev, /*rt */
4687     knodev, knodev, knodev, knodev, /*rp */
4688     knodev, knodev, knodev, knodev, /*tm */
4689     knodev, knodev, knodev, knodev, /*hs */
4690     knodev, knodev, knodev, knodev, /*hp */
4691     knodev, knodev, knodev, knodev, /*ht */
4692     0
4693 };
4694
4695 int rootdev {(0<<8)|0};
4696 int swapdev {(0<<8)|0};
4697 int swplo 4000; /* cannot be zero */
4698 int nswap 872;
4699

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

4700 #
4701 /*
4702 */
4703
4704 #include "../param.h"
4705 #include "../user.h"
4706 #include "../buf.h"
4707 #include "../conf.h"
4708 #include "../system.h"
4709 #include "../proc.h"
4710 #include "../seg.h"
4711
4712 /*
4713 * This is the set of buffres proper, whose heads
4714 * were declared in buf.h. There can exist buffer
4715 * headers not pointing here that are used purely
4716 * as arguments to the I/O routines to describe
4717 * I/O to be done-- e.g. swbuf, just below, for
4718 * swapping.
4719 */
4720 char      buffers[NBUF] [514];
4721 struct    buf      swbuf;
4722
4723 /*
4724 * Declarations of the tables for the magtape devices;
4725 * see bdwrite.
4726 */
4727 int      tmtab;
4728 int      httab;
4729
4730 /*
4731 * The following several routines allocate and free
4732 * buffers with various side effects. In general the
4733 * arguments to an allocate routine are a device and
4734 * a block number, and the value is a pointer to
4735 * the buffer header; the buffer is marked "busy"
4736 * so that no one else can touch it. If the block was
4737 * already in core, no I/O need be done; if it is
4738 * already busy, the process waits until it becomes free.
4739 * The following routines allocate a buffer:
4740 * getblk
4741 * bread
4742 * breada
4743 * Eventually the buffer must be released, possibly with the
4744 * side effect of writing it out, by using one of
4745 * bwrite
4746 * bdwrite
4747 * bawrite
4748 * brelse
4749 */

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

4750
4751 /* Read in (if necessary) the block and
4752 * return a buffer pointer.
4753 */
4754 bread(dev, blkno)
4755 {
4756     register struct buf *rbp;
4757
4758     rbp = getblk(dev, blkno);
4759     if (rbp->b_flags&B_DONE)
4760         return(rbp);
4761     rbp->b_flags =| B_READ;
4762     rbp->b_wcount = -256;
4763     (*bdevsw[dev.d_major].d_strategy)(rbp);
4764     iowait(rbp);
4765     return(rbp);
4766 }
4767 /* ----- */
4768
4769 /*
4770 * Read in the block, like bread, but also start I/O on the
4771 * read-ahead block (which is not allocated to the caller)
4772 */
4773 breada(aDEV, blkno, rablkno)
4774 {
4775     register struct buf *rbp, *rabp;
4776     register int dev;
4777
4778     dev = aDEV;
4779     rbp = 0;
4780     if (!incore(dev, blkno)) {
4781         rbp = getblk(dev, blkno);
4782         if ((rbp->b_flags&B_DONE) == 0) {
4783             rbp->b_flags =| B_READ;
4784             rbp->b_wcount = -256;
4785             (*bdevsw[aDEV.d_major].d_strategy)(rbp);
4786         }
4787     }
4788     if (rablkno && !incore(dev, rablkno)) {
4789         rabp = getblk(dev, rablkno);
4790         if (rabp->b_flags & B_DONE)
4791             brelse(rabp);
4792     } else {
4793         rabp->b_flags =| B_READ|B_ASYNC;
4794         rabp->b_wcount = -256;
4795         (*bdevsw[aDEV.d_major].d_strategy)(rabp);
4796     }
4797 }
4798 if (rbp==0)
4799     return(bread(dev, blkno));

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

4800 lowait(rbp);
4801 return(rbp);
4802 }
4803 /* ----- */
4804
4805 /*
4806 * Write the buffer, waiting for completion.
4807 * The release the buffer.
4808 */
4809 bwrite(bp)
4810 struct buf *bp;
4811 {
4812     register struct buf *rbp;
4813     register flag;
4814
4815     rbp = bp;
4816     flag = rbp->b_flags;
4817     rbp->b_flags = ~ (B_READ | B_DONE | B_ERROR | B_DELMRI);
4818     rbp->b_wcount = -256;
4819     (*bdevsw[rbp->b_dev.d_major].d_strategy)(rbp);
4820     if ((flag & B_ASYNC) == 0) {
4821         lowait(rbp);
4822         bwrite(rbp);
4823     } else if ((flag & B_DELMRI) == 0)
4824         geterror(rbp);
4825 }
4826 /* ----- */
4827
4828 /*
4829 * Release the buffer, marking it so that if it is grabbed
4830 * for another purpose it will be written out before being
4831 * given up (e.g. when writing a partial block where it is
4832 * assumed that another write for the same block will soon
4833 * follow). This can't be done for magtape, since writes
4834 * must be done in the same order as requested.
4835 */
4836 bwrite(bp)
4837 struct buf *bp;
4838 {
4839     register struct buf *rbp;
4840     register struct devtab *dp;
4841     rbp = bp;
4842     dp = bdevsw[rbp->b_dev.d_major].d_tab;
4843     if (dp == &mtab || dp == &htab)
4844         bwrite(rbp);
4845     else {
4846         rbp->b_flags = | B_DELMRI | B_DONE;
4847     }
4848 }

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

4850 }
4851 /* ----- */
4852
4853 /* Release the buffer, start I/O on it, but don't wait
4854 * for completion */
4855
4856 bwrite(bp)
4857 struct buf *bp;
4858 {
4859     register struct buf *rbp;
4860     rbp = bp;
4861     rbp->b_flags = | B_ASYNC;
4862     bwrite(rbp);
4863 }
4864 /* ----- */
4865
4866 /* release the buffer, with no I/O implied.
4867 */
4868
4869 bwrite(bp)
4870 struct buf *bp;
4871 {
4872     register struct buf *rbp, **backp;
4873     register int sps;
4874     rbp = bp;
4875     if (rbp->b_flags & B_WANTED)
4876         wakeup(rbp);
4877     if (btreelist.b_flags & B_WANTED)
4878         btreelist.b_flags = ~ B_WANTED;
4879     wakeup(&btreelist);
4880     if (rbp->b_flags < B_ASYNC)
4881         splay();
4882     rbp->b_flags = ~ (B_WANTED | B_BUSY | B_ASYNC);
4883     (*backp)->av_forw = rbp;
4884     rbp->av_back = *backp;
4885     rbp->av_forw = &btreelist;
4886     *backp = rbp;
4887     PS->integ = sps;
4888 }
4889 /* ----- */
4890
4891 PS->integ = sps;
4892 }
4893
4894 /* See if the block is associated with some buffer
4895 * (mainly to avoid getting hung up on a wait in breada)
4896 */
4897
4898 incore(adev, blkno)
4899

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976



```

4900 {
4901     register int dev;
4902     register struct buf *bp;
4903     register struct devtab *dp;
4904
4905     dev = adev;
4906     dp = bdevsw[adev.d_major].d_tab;
4907     for (bp=dp->b_forw; bp != dp; bp = bp->b_forw)
4908         if (bp->b_blkno==blkno && bp->b_dev==dev)
4909             return(bp);
4910     return(0);
4911 }
4912 /* ----- */
4913
4914 /* Assign a buffer for the given block.  If the appropriate
4915 * block is already associated, return it; otherwise search
4916 * for the oldest non-busy buffer and reassign it.
4917 * When a 512-byte area is wanted for some random reason
4918 * (e.g. during exec, for the user arglist) getblk can be
4919 * called with device NODEV to avoid unwanted associativity.
4920 */
4921 getblk(dev, blkno)
4922 {
4923     register struct buf *bp;
4924     register struct devtab *dp;
4925     extern lbolt;
4926
4927     if(dev.d_major >= nblkdev)
4928         panic("blkdev");
4929
4930     loop:
4931     if (dev < 0)
4932         dp = &bfreelist;
4933     else {
4934         dp = bdevsw[dev.d_major].d_tab;
4935         if(dp == NULL)
4936             panic("devtab");
4937         for (bp=dp->b_forw; bp != dp; bp = bp->b_forw) {
4938             if (bp->b_blkno!=blkno || bp->b_dev!=dev)
4939                 continue;
4940             spl6();
4941             if (bp->b_flags&B_BUSY) {
4942                 bp->b_flags |= B_WANTED;
4943                 sleep(bp, PRIBIO);
4944                 spl0();
4945                 goto loop;
4946             }
4947             spl0();
4948             notavail(bp);
4949             return(bp);

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

4950     }
4951 }
4952 spl6();
4953 if (bfreelist.av_forw == &bfreelist) {
4954     bfreelist.b_flags |= B_WANTED;
4955     sleep(&bfreelist, PRIBIO);
4956     spl0();
4957     goto loop;
4958 }
4959 spl0();
4960 notavail(bp = bfreelist.av_forw);
4961 if (bp->b_flags & B_DELWRI) {
4962     bp->b_flags |= B_ASYNC;
4963     bwrite(bp);
4964     goto loop;
4965 }
4966 bp->b_flags = B_BUSY | B_RELOC;
4967 bp->b_back->b_forw = bp->b_forw;
4968 bp->b_forw->b_back = bp->b_back;
4969 bp->b_forw = dp->b_forw;
4970 bp->b_back = dp;
4971 dp->b_forw->b_back = bp;
4972 dp->b_forw = bp;
4973 bp->b_dev = dev;
4974 bp->b_blkno = blkno;
4975 return(bp);
4976 }
4977 /* ----- */
4978
4979 /* Wait for I/O completion on the buffer; return errors
4980 * to the user.
4981 */
4982 iowait(bp)
4983 struct buf *bp;
4984 {
4985     register struct buf *rbp;
4986
4987     rbp = bp;
4988     spl6();
4989     while ((rbp->b_flags&B_DONE)==0)
4990         sleep(rbp, PRIBIO);
4991     spl0();
4992     geterror(rbp);
4993 }
4994 /* ----- */
4995
4996 /* Unlink a buffer from the available list and mark it busy.
4997 * (internal interface)
4998 */
4999 notavil(bp)

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

5000 struct buf *bp;
5001 }
5002 register struct buf *rbp;
5003 register int sps;
5004
5005 rbp = bp;
5006 sps = PS->integ;
5007 spl6();
5008 rbp->av_back->av_forw = rbp->av_forw;
5009 rbp->av_forw->av_back = rbp->av_back;
5010 rbp->b_flags = B_BUSY;
5011 PS->integ = sps;
5012 }
5013 /* ----- */
5014
5015 /* Mark I/O complete on a buffer, release it if i/o is
5016 * asynchronous, and wake up anyone waiting for it.
5017 */
5018 idone(bp)
5019 struct buf *bp;
5020 }
5021 register struct buf *rbp;
5022
5023 rbp = bp;
5024 if(rbp->b_flags*B_MAP)
5025 mapfree(rbp);
5026 rbp->b_flags = B_DONE;
5027 if (rbp->b_flags&B_ASYNC)
5028 brelse(rbp);
5029 else {
5030 rbp->b_flags = B_WANTED;
5031 wakeup(rbp);
5032 }
5033 }
5034 /* ----- */
5035
5036 /* Zero the core associated with a buffer.
5037 */
5038 clrbuf(bp)
5039 int *bp;
5040 {
5041 register *p;
5042 register c;
5043 p = bp->b_addr;
5044 c = 256;
5045 do
5046 *p++ = 0;
5047 while(--c);
5048 }
5049 }

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

5050 /* ----- */
5051
5052 /* Initialize the buffer I/O system by freeing
5053 * all buffers and setting all device buffer lists to empty.
5054 */
5055 binit()
5056 {
5057 register struct buf *bp;
5058 register struct devtab *dp;
5059 register int i;
5060 struct devsw *bdp;
5061
5062 btreelist.b_forw = btreelist.b_back =
5063 btreelist.av_forw = btreelist.av_back = btreelist;
5064 for (i=0; i<NBUF; i++) {
5065 bp = bbuf[i];
5066 bp->b_dev = -1;
5067 bp->b_addr = bbufers[i];
5068 bp->b_back = &btreelist;
5069 bp->b_forw = btreelist.b_forw;
5070 btreelist.b_forw->b_back = bp;
5071 btreelist.b_forw = bp;
5072 bp->b_flags = B_BUSY;
5073 brelse(bp);
5074 }
5075 i = 0;
5076 for (bdp = bdevsw; bdp->d_open; bdp++) {
5077 dp = bdp->d_tab;
5078 if(dp) {
5079 dp->b_forw = dp;
5080 dp->b_back = dp;
5081 }
5082 i++;
5083 }
5084 bblkdev = i;
5085 }
5086 /* ----- */
5087
5088 /* Device start routine for disks
5089 * and other devices that have the register
5090 * layout of the older DBC controllers (RF, RK, RP, TM)
5091 */
5092 #define IENABLR 0100
5093 #define WCOM 02
5094 #define RCOM 04
5095 #define GO 01
5096 devstart(bp, devloc, devblk, hbcom)
5097 struct buf *bp;
5098 int *devloc;
5099 {
5100

```

```

5100 register int *dp;
5101 register struct buf *rbp;
5102 register int com;
5103
5104 dp = devloc;
5105 rbp = bp;
5106 *dp = devblk; /* block address */
5107 *--dp = rbp->b_addr; /* buffer address */
5108 *--dp = rbp->b_wcount; /* word count */
5109 com = (hbcom<<8) | IENABLE | GO |
5110 ((rbp->b_xmem & 03) << 4);
5111 if (rbp->b_flags&B_READ) /* command + x-mem */
5112     com = | RCOM;
5113 else
5114     com = | WCOM;
5115 *--dp = com;
5116 }
5117 /* ----- */
5118
5119 /* startup routine for RH controllers. */
5120 #define RHWCOM 060
5121 #define RHRCOM 070
5122
5123 rhstart(bp, devloc, devblk, abae)
5124 struct buf *bp;
5125 int *devloc, *abae;
5126 {
5127     register int *dp;
5128     register struct buf *rbp;
5129     register int com;
5130
5131     dp = devloc;
5132     rbp = bp;
5133     if(cputype == 70)
5134         *abae = rbp->b_xmem;
5135     *dp = devblk; /* block address */
5136     *--dp = rbp->b_addr; /* buffer address */
5137     *--dp = rbp->b_wcount; /* word count */
5138     com = IENABLE | GO |
5139         ((rbp->b_xmem & 03) << 8);
5140     if (rbp->b_flags&B_READ) /* command + x-mem */
5141         com = | RHRCOM; else
5142         com = | RHWCOM;
5143     *--dp = com;
5144 }
5145 /* ----- */
5146
5147 /*
5148 * 11/70 routine to allocate the
5149 * UNIBUS map and initialize for

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

5150 * a unibus device.
5151 * The code here and in
5152 * rhstart assumes that an rh on an 11/70
5153 * is an rh70 and contains 22 bit addressing.
5154 */
5155 int maplock;
5156 mapalloc(abp)
5157 struct buf *abp;
5158 {
5159     register i, a;
5160     register struct buf *bp;
5161
5162     if(cputype != 70)
5163         return;
5164     spl6();
5165     while(maplock&B_BUSY) {
5166         maplock = | B_WANTED;
5167         sleep(&maplock, PSWP);
5168     }
5169     maplock = | B_BUSY;
5170     spl0();
5171     bp = abp;
5172     bp->b_flags = | B_MAP;
5173     a = bp->b_xmem;
5174     for(i=16; i<32; i+=2)
5175         UBMAP->r[i+1] = a;
5176     for(a++; i<48; i+=2)
5177         UBMAP->r[i+1] = a;
5178     bp->b_xmem = 1;
5179 }
5180 /* ----- */
5181
5182 mapfree(bp)
5183 struct buf *bp;
5184 {
5185
5186     bp->b_flags = & ~B_MAP;
5187     if(maplock&B_WANTED)
5188         wakeup(&maplock);
5189     maplock = 0;
5190 }
5191 /* ----- */
5192
5193 /*
5194 * swap I/O
5195 */
5196 swap(blkno, coreaddr, count, rdflg)
5197 {
5198     register int *fp;
5199

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

5250 fp = kswbuf.b_flags;
5251 spl6();
5252 while (*fp & B_BUSY) {
5253     *fp = | B_WANTED;
5254     sleep(fp, PSM);
5255     * The device number
5256     * Read/write flag
5257     * Essentially all the work is computing physical addresses
5258     * and validating them.
5259     * /
5260 physio(strat, abp, dev, rw)
5261 struct buf *abp;
5262 int (*strat)();
5263     {
5264     register struct buf *bp;
5265     register char *base;
5266     register int nb;
5267     int ts;
5268     wakeup(fp);
5269     spl0();
5270     *fp = k~(B_BUSY|B_WANTED);
5271     return(*fp & ERROR);
5272     }
5273     * /
5274     * Check odd base, odd count, and address wraparound
5275     * /
5276     * make sure all write-behind blocks
5277     * on dev (or NODEV for all)
5278     * are flushed out.
5279     * (from umount and update)
5280     * /
5281     * Check overlap with text. (ts and nb now
5282     * in 64-byte clicks)
5283     * /
5284     * Check that transfer is either entirely in the
5285     * data or in the stack: that is, either
5286     * the end is in the data or the start is in the stack
5287     * (remember wraparound was already checked).
5288     * /
5289     * ((base+u.n_count)<<6)&01777) >= ts+u.n_size
5290     * nb > 1024-u.n_size)
5291     goto bad;
5292     spl6();
5293     while (bp->b_flags & B_BUSY) {
5294     * /
5295     * sleep(bp, PRIBIO);
5296     * /
5297     * bp->b_flags = | B_WANTED;
5298     * /
5299     * bp->b_flags = B_BUSY | B_PHYS | rw;

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

5300     * /
5301     * fp = | B_WANTED;
5302     * sleep(fp, PSM);
5303     * /
5304     * bp->b_flags = B_BUSY | B_PHYS | rdtlg;
5305     * /
5306     * (count<<5); /* 32 w/block */
5307     * /
5308     * sbuf.b_dev = swapdev;
5309     * /
5310     * sbuf.b_wcount = - (count<<6); /* 64 b/block */
5311     * /
5312     * sbuf.b_addr = coreaddr<<6;
5313     * /
5314     * sbuf.b_xmem = (coreaddr>>10) & 077;
5315     * /
5316     * sbuf.b_strategy) (&swbuf);
5317     * /
5318     * while ((*fp & B_BUSY) == 0)
5319     * /
5320     * sleep(fp, PSM);
5321     * /
5322     * make sure all write-behind blocks
5323     * on dev (or NODEV for all)
5324     * are flushed out.
5325     * (from umount and update)
5326     * /
5327     * bp->b_flags = B_BUSY | B_PHYS | rdtlg;
5328     * /
5329     * ((base+u.n_count)<<6)&01777) >= ts+u.n_size
5330     * nb > 1024-u.n_size)
5331     goto bad;
5332     spl6();
5333     while (bp->b_flags & B_BUSY) {
5334     * /
5335     * sleep(bp, PRIBIO);
5336     * /
5337     * bp->b_flags = | B_WANTED;
5338     * /
5339     * bp->b_flags = B_BUSY | B_PHYS | rw;

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

5300 bp->b_dev = dev;
5301 /*
5302  * Compute physical address by simulating
5303  * the segmentation hardware.
5304  */
5305 bp->b_addr = base&077;
5306 base = (u.u_sep? UDSA: UISA)->r[nb>>7] + (nb&0177);
5307 bp->b_addr += base<<6;
5308 bp->b_xmem = (base>>10) & 077;
5309 bp->b_blkno = lshift(u.u_offset, -9);
5310 bp->b_wcount = -((u.u_count>>1) & 077777);
5311 bp->b_error = 0;
5312 u.u_procp->p_flag |= SLOCK;
5313 (*strat)(bp);
5314 spl6();
5315 while ((bp->b_flags&B_DONE) == 0)
5316     sleep(bp, PRIBIO);
5317 u.u_procp->p_flag &= ~SLOCK;
5318 if (bp->b_flags&B_WANTED)
5319     wakeup(bp);
5320 spl0();
5321 bp->b_flags &= ~(B_BUSY|B_WANTED);
5322 u.u_count = (-bp->b_resid)<<1;
5323 geterror(bp);
5324 return;
5325 bad:
5326 u.u_error = EFAULT;
5327 }
5328 /* ----- */
5329
5330 /*
5331  * Pick up the device's error number and pass it to the
5332  * user; if there is an error but the number is 0 set a
5333  * generalised code. Actually the latter is always true
5334  * because devices don't yet return specific errors.
5335  */
5336 geterror(abp)
5337 struct buf *abp;
5338 {
5339     register struct buf *bp;
5340
5341     bp = abp;
5342     if (bp->b_flags&B_ERROR)
5343         if ((u.u_error = bp->b_error)==0)
5344             u.u_error = EIO;
5345 }
5346 /* ----- */
5347
5348
5349

```

```

5350 #
5351 /*
5352 */
5353
5354 /*
5355  * RK disk driver
5356  */
5357
5358 #include "../param.h"
5359 #include "../buf.h"
5360 #include "../conf.h"
5361 #include "../user.h"
5362
5363 #define RKADDR 0177400
5364 #define NRK 4
5365 #define NRKBLK 4872
5366
5367 #define RESET 0
5368 #define GO 01
5369 #define DRESET 014
5370 #define IENABLE 0100
5371 #define DRY 0200
5372 #define ARDY 0100
5373 #define WLO 020000
5374 #define CTLRDY 0200
5375
5376 struct {
5377     int rkds;
5378     int rker;
5379     int rkcs;
5380     int rkwc;
5381     int rkba;
5382     int rkda;
5383 };
5384 /* ----- */
5385
5386 struct devtab rktab;
5387 struct buf rrkbuf;
5388
5389 rkstrategy(abp)
5390 struct buf *abp;
5391 {
5392     register struct buf *bp;
5393     register *qc, *ql;
5394     int d;
5395
5396     bp = abp;
5397     if (bp->b_flags&B_PHYS)
5398         mapalloc(bp);
5399     d = bp->b_dev.d_minor-7;

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

5400 if(d <= 0)
5401     d = 1;
5402     if(bp->b_blkmo >= NRKBLK*d) {
5403         bp->b_flags = | B_ERROR;
5404         fdone(bp);
5405         return;
5406     }
5407     bp->av_forw = 0;
5408     sp15();
5409     if (rktab.d_actf==0)
5410         rktab.d_actf = bp;
5411     else
5412         rktab.d_actl->av_forw = bp;
5413         rktab.d_actl = bp;
5414         if (rktab.d_actlve==0)
5415             rktab.d_actlve==0)
5416             sp10();
5417         }
5418         /* ----- */
5419         rkaddr(bp)
5420         struct buf *bp;
5421         struct buf *bp;
5422     }
5423         register struct buf *p;
5424         register int b;
5425         int d, m;
5426         p = bp;
5427         b = p->b_blkmo;
5428         m = p->b_dminor - 7;
5429         p = p->b_dminor - 7;
5430         if(m <= 0)
5431             d = p->b_dminor;
5432         else {
5433             d = lrem(b, m);
5434             b = ldiv(b, m);
5435         }
5436         return(d<<13 | (b/12)<<4 | b%12);
5437     }
5438     /* ----- */
5439     rkstart()
5440     {
5441         register struct buf *bp;
5442         register struct buf *bp;
5443         if ((bp = rktab.d_actf) == 0)
5444             return;
5445         rktab.d_actlve++;
5446         devstart(bp, &RKADDR->rksda, rkaddr(bp), 0);
5447     }
5448     /* ----- */
5449     }

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

5450     rkintr()
5451     {
5452         register struct buf *bp;
5453         if (rktab.d_actlve == 0)
5454             return;
5455         bp = rktab.d_actf;
5456         rktab.d_actlve = 0;
5457         if (RKADDR->rks < 0) {
5458             /* error bit */
5459             deverror(bp, RKADDR->rker, RKADDR->rks);
5460             RKADDR->rks = RESET|GO;
5461             while((RKADDR->rks&CRTLIDY) == 0) {
5462                 if (++rktab.d_errcnt <= 10)
5463                     rktab.d_errcnt = 0;
5464                 rktab.d_actf = bp->av_forw;
5465                 fdone(bp);
5466                 rkstart();
5467                 bp->b_flags = | B_ERROR;
5468             }
5469             rktab.d_errcnt = 0;
5470             rkread(dev)
5471             {
5472                 physio(rkstrategy, &rktbuf, dev, B_READ);
5473             }
5474             /* ----- */
5475             rkwrite(dev)
5476             {
5477                 physio(rkstrategy, &rktbuf, dev, B_WRITE);
5478             }
5479             /* ----- */
5480             rkstart()
5481             {
5482                 physio(rkstrategy, &rktbuf, dev, B_WRITE);
5483             }
5484         }
5485     }
5486     /* ----- */
5487     }
5488     }
5489     }
5490     }
5491     }
5492     }
5493     }
5494     }
5495     }
5496     }
5497     }
5498     }
5499     }

```

# 4

**Files and Directories**  
**File Systems**  
**Pipes**





```

5500 /*
5501  * One file structure is allocated
5502  * for each open/creat/pipe call.
5503  * Main use is to hold the read/write
5504  * pointer associated with each open
5505  * file.
5506  */
5507 struct      file
5508 {
5509     char      f_flag;
5510     char      f_count;          /* reference count */
5511     int       f_inode;         /* pointer to inode structure */
5512     char      *f_offset[2];    /* read/write character pointer */
5513 } file[NFILE];
5514 /* ----- */
5515
5516 /* flags */
5517 #define      FREAD      01
5518 #define      FWRITE     02
5519 #define      FPIPE      04
5520
5521
5522
5523
5524
5525
5526
5527
5528
5529
5530
5531
5532
5533
5534
5535
5536
5537
5538
5539
5540
5541
5542
5543
5544
5545
5546
5547
5548
5549

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

5550 /*
5551  * Definition of the unix super block.
5552  * The root super block is allocated and
5553  * read in iinit/alloc.c. Subsequently
5554  * a super block is allocated and read
5555  * with each mount (smount/sys3.c) and
5556  * released with umount (sumount/sys3.c).
5557  * A disk block is ripped of for storage.
5558  * See alloc.c for general alloc/free
5559  * routines for free list and I list.
5560  */
5561 struct filsys
5562 {
5563     int      s_ysize;          /* size in blocks of I list */
5564     int      s_fsize;         /* size in blocks of entire volume */
5565     int      s_nfree;         /* number of in core free blocks
5566                               (between 0 and 100) */
5567     int      s_free[100];     /* in core free blocks */
5568     int      s_ninode;        /* number of in core I nodes (0-100) */
5569     int      s_inode[100];    /* in core free I nodes */
5570     char      s_flock;        /* lock during free list manipulation */
5571     char      s_ilock;        /* lock during I list manipulation */
5572     char      s_fmod;         /* super block modified flag */
5573     char      s_ronly;        /* mounted read-only flag */
5574     int      s_time[2];       /* current date of last update */
5575     int      pad[50];
5576 };
5577 /* ----- */
5578
5579
5580
5581
5582
5583
5584
5585
5586
5587
5588
5589
5590
5591
5592
5593
5594
5595
5596
5597
5598
5599

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

5600 /*
5601 * Inode structure as it appears on
5602 * the disk. Not used by the system,
5603 * but by things like check, df, dump.
5604 */
5605 struct inode
5606 {
5607     int     i_mode;
5608     char    i_nlink;
5609     char    i_uid;
5610     char    i_gid;
5611     char    i_size0;
5612     char    *i_size1;
5613     int     i_addr[8];
5614     int     i_atime[2];
5615     int     i_mtime[2];
5616 };
5617 /* ----- */

```

```

5650 /* The I node is the focus of all
5651 * file activity in unix. There is a unique
5652 * inode allocated for each active file,
5653 * each current directory, each mounted-on
5654 * file, text file, and the root. An inode is 'named'
5655 * by its dev/inumber pair. (iget/get.c)
5656 * Data, from mode on, is read in
5657 * from permanent inode on volume.
5658 */
5659 struct inode
5660 {
5661     char    i_flag;
5662     char    i_count; /* reference count */
5663     int     i_dev; /* device where inode resides */
5664     int     i_number; /* i number, 1-to-1 with device
5665     address */
5666     int     i_mode;
5667     char    i_nlink; /* directory entries */
5668     char    i_uid; /* owner */
5669     char    i_gid; /* group of owner */
5670     char    i_size0; /* most significant of size */
5671     char    i_size1; /* least sig */
5672     int     i_addr[8]; /* device addresses constituting file */
5673     int     i_last; /* last logical block read (for
5674     read-ahead) */
5675     } inode[NINODE];
5676 ----- */

```

```

5677 /* flags */
5678 #define ILOCK 01 /* inode is locked */
5679 #define IUPD 02 /* inode has been modified */
5680 #define IACC 04 /* inode access time to be updated */
5681 #define IMOUNT 010 /* inode is mounted on */
5682 #define IWANT 020 /* some process waiting on lock */
5683 #define ITEXT 040 /* inode is pure text prototype */
5684 /* modes */
5685 #define IALLLOC 0100000 /* file is used */
5686 #define IRMT 060000 /* type of file */
5687 #define IDIR 040000 /* directory */
5688 #define ICHR 020000 /* character special */
5689 #define IBLK 060000 /* block special, 0 is regular */
5690 #define IARG 010000 /* large addressing algorithm */
5691 #define ISUID 04000 /* set user id on execution */
5692 #define ISGID 02000 /* set group id on execution */
5693 #define ISVTX 01000 /* save swapped text even after use */
5694 #define IREAD 0400 /* read, write, execute permissions */
5695 #define IWRITE 0200
5696 #define IEXEC 0100
5697 #define IEXEC 0100
5698 #define IEXEC 0100
5699

```

```

5700 #
5701 #include "../param.h"
5702 #include "../system.h"
5703 #include "../user.h"
5704 #include "../reg.h"
5705 #include "../file.h"
5706 #include "../inode.h"
5707
5708 /*
5709  * read system call
5710  */
5711 read()
5712 {
5713     rdwr(FREAD);
5714 }
5715 /* ----- */
5716
5717 /*
5718  * write system call
5719  */
5720 write()
5721 {
5722     rdwr(FWRITE);
5723 }
5724 /* ----- */
5725
5726 /*
5727  * common code for read and write calls:
5728  * check permissions, set base, count, and offset,
5729  * and switch out to readi, writei, or pipe code.
5730  */
5731 rdwr(mode)
5732 {
5733     register *fp, m;
5734
5735     m = mode;
5736     fp = getf(u.u_ar0[R0]);
5737     if(fp == NULL)
5738         return;
5739     if((fp->f_flag&m) == 0) {
5740         u.u_error = EBADF;
5741         return;
5742     }
5743     u.u_base = u.u_arg[0];
5744     u.u_count = u.u_arg[1];
5745     u.u_segflg = 0;
5746     if(fp->f_flag&FPIPE) {
5747         if(m==FREAD)
5748             readp(fp); else
5749             writep(fp);

```

```

5750     } else {
5751         u.u_offset[1] = fp->f_offset[1];
5752         u.u_offset[0] = fp->f_offset[0];
5753         if(m==FREAD)
5754             readi(fp->f_inode); else
5755             writei(fp->f_inode);
5756         dpadd(fp->f_offset, u.u_arg[1]-u.u_count);
5757     }
5758     u.u_ar0[R0] = u.u_arg[1]-u.u_count;
5759 }
5760 /* ----- */
5761
5762 /*
5763  * open system call
5764  */
5765 open()
5766 {
5767     register *ip;
5768     extern uchar;
5769
5770     ip = namei(&uchar, 0);
5771     if(ip == NULL)
5772         return;
5773     u.u_arg[1]++;
5774     open1(ip, u.u_arg[1], 0);
5775 }
5776 /* ----- */
5777
5778 /*
5779  * creat system call
5780  */
5781 creat()
5782 {
5783     register *ip;
5784     extern uchar;
5785
5786     ip = namei(&uchar, 1);
5787     if(ip == NULL) {
5788         if(u.u_error)
5789             return;
5790         ip = maknode(u.u_arg[1]&07777&(~ISVTX));
5791         if(ip==NULL)
5792             return;
5793         open1(ip, FWRITE, 2);
5794     }
5795     open1(ip, FWRITE, 1);
5796 }
5797 /* ----- */
5798
5799 /*

```

```

5800 * common code for open and creat.
5801 * Check permissions, allocate an open file structure,
5802 * and call the device open routine if any.
5803 */
5804 open(ip, mode, trf)
5805 int *ip;
5806 {
5807     register struct file *fp;
5808     register *rip, m;
5809     int i;
5810     rip = ip;
5811     m = mode;
5812     if(trf != 2)
5813     {
5814         if(m&FREAD)
5815             access(rip, IRREAD);
5816         if(m&FWRITE)
5817             access(rip, IWWRITE);
5818         if(rip->mode&IFMT) == IFDIR
5819             u.u_error = EISDIR;
5820     }
5821     if(u.u_error)
5822         goto out;
5823     if(trf)
5824         truncate(rip);
5825     if(rip)
5826         prele(rip);
5827     if((rip == failloc()) == NULL)
5828         goto out;
5829     fp->f_lag = mk(FREAD|FWRITE);
5830     fp->f_inode = rip;
5831     i = u.u_ar0[R0];
5832     open(rip, m&FWRITE);
5833     if(u.u_error == 0)
5834         return;
5835     u.u_offle[i] = NULL;
5836     fp->f_count--;
5837 out:
5838     if(rip)
5839         iput(rip);
5840 }
5841 /* ----- */
5842 /*
5843 */
5844 * close system call
5845 */
5846 close()
5847 {
5848     register *fp;
5849 }

```

```

5850 fp = getf(u.u_ar0[R0]);
5851 if(fp == NULL)
5852     return;
5853 u.u_offle[u.u_ar0[R0]] = NULL;
5854 closef(fp);
5855 }
5856 /* ----- */
5857 /*
5858 */
5859 * seek system call
5860 */
5861 seek()
5862 {
5863     int n[2];
5864     register *fp, t;
5865     fp = getf(u.u_ar0[R0]);
5866     if(fp == NULL)
5867         return;
5868     if(fp->f_lag&FPIPE)
5869         u.u_error = EPIPE;
5870     return;
5871     if(u.u_error)
5872         return;
5873     t = u.u_arg[1];
5874     if(t > 2)
5875         n[1] = u.u_arg[0] << 9;
5876         n[0] = u.u_arg[0] << 7;
5877     if(t == 3)
5878         n[0] = & 0777;
5879     } else {
5880         n[1] = u.u_arg[0];
5881         n[0] = 0;
5882         if(t == 0 && n[1] > 0)
5883             n[0] = -1;
5884     }
5885     switch(t) {
5886     case 1:
5887         n[0] += fp->f_offset[0];
5888     case 4:
5889         n[0] += fp->f_offset[1];
5890         dpadd(n, fp->f_offset[1]);
5891     case 0:
5892         n[0] += fp->f_inode->i_size0377;
5893         dpadd(n, fp->f_inode->i_size1);
5894     case 3:
5895         n[0] += fp->f_inode->i_size1;
5896     }
5897 }

```

```

5900 }
5901 fp->f_offset[1] = n[1];
5902 fp->f_offset[0] = n[0];
5903 }
5904 /* ----- */
5905
5906
5907 /* link system call
5908 */
5909 link()
5910 {
5911     register *ip, *xp;
5912     extern uchar;
5913
5914     ip = namei(&uchar, 0);
5915     if(ip == NULL)
5916         return;
5917     if(ip->i_nlink >= 127) {
5918         u.u_error = EMLINK;
5919         goto out;
5920     }
5921     if((ip->i_mode&IFMT)==IFDIR && !suser())
5922         goto out;
5923     /*
5924      * ulock to avoid possible hanging in namei
5925      */
5926     ip->i_flag =& ~ILOCK;
5927     u.u_dirp = u.u_arg[1];
5928     xp = namei(&uchar, 1);
5929     if(xp != NULL) {
5930         u.u_error = EEXIST;
5931         iput(xp);
5932     }
5933     if(u.u_error)
5934         goto out;
5935     if(u.u_pdir->i_dev != ip->i_dev) {
5936         iput(u.u_pdir);
5937         u.u_error = EXDEV;
5938         goto out;
5939     }
5940     wdir(ip);
5941     ip->i_nlink++;
5942     ip->i_flag |= IUPD;
5943
5944 out:
5945     iput(ip);
5946 }
5947 /* ----- */
5948
5949 /*

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

5950 * mknod system call
5951 */
5952 mknod()
5953 {
5954     register *ip;
5955     extern uchar;
5956
5957     if(suser()) {
5958         ip = namei(&uchar, 1);
5959         if(ip != NULL) {
5960             u.u_error = EEXIST;
5961             goto out;
5962         }
5963     }
5964     if(u.u_error)
5965         return;
5966     ip = maknode(u.u_arg[1]);
5967     if (ip==NULL)
5968         return;
5969     ip->i_addr[0] = u.u_arg[2];
5970
5971 out:
5972     iput(ip);
5973 }
5974 /* ----- */
5975
5976 /* sleep system call
5977 * not to be confused with the sleep internal routine.
5978 */
5979 sslep()
5980 {
5981     char *d[2];
5982
5983     spl7();
5984     d[0] = time[0];
5985     d[1] = time[1];
5986     dpadd(d, u.u_ar0[R0]);
5987
5988     while(dpcmp(d[0], d[1], time[0], time[1]) > 0) {
5989         if(dpcmp(tout[0], tout[1], time[0], time[1]) <= 0 ||
5990            dpcmp(tout[0], tout[1], d[0], d[1]) > 0) {
5991             tout[0] = d[0];
5992             tout[1] = d[1];
5993         }
5994         sleep(tout, PSLEP);
5995     }
5996     spl0();
5997 }
5998 /* ----- */
5999

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

6050  update(ip, time);
6051  bp = bread(ip->dev, ldiv(ip->number+31, 16));
6052  cp = bp->b_addr + 32*irem(ip->number+31, 16) + 24;
6053  ip = &(ip->dev);
6054  for(i=0; i<14; i++) {
6055      sword(ub, *ip++);
6056      ub = + 2;
6057  }
6058  for(i=0; i<4; i++) {
6059      sword(ub, *cp++);
6060      ub = + 2;
6061  }
6062  prelse(bp);
6063  }
6064  /* ----- */
6065  /*
6066  *
6067  * the dup system call.
6068  */
6069  dup()
6070  {
6071      register i, *fp;
6072      fp = getf(u_n_ar0[R0]);
6073      if(fp == NULL)
6074          return;
6075      if ((i = ufailoc()) > 0)
6076          return;
6077      u_n_ofile[i] = fp;
6078      fp->f_count++;
6079  }
6080  /* ----- */
6081  /*
6082  *
6083  * the mount system call.
6084  */
6085  /*
6086  smount()
6087  {
6088      int d;
6089      register *ip;
6090      register struct mount *mp, *smp;
6091      extern uchar;
6092      d = getmdev();
6093      if(u_n_error)
6094          return;
6095      u_n_dirp = u_n_arg[1];
6096      ip = named(&uchar, 0);
6097      if(ip == NULL)
6098          return;
6099  }

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

6000  #
6001  #include "../param.h"
6002  #include "../system.h"
6003  #include "../reg.h"
6004  #include "../buf.h"
6005  #include "../files.h"
6006  #include "../user.h"
6007  #include "../inode.h"
6008  #include "../file.h"
6009  #include "../conf.h"
6010
6011  /*
6012  * the fstat system call.
6013  */
6014  fstat()
6015  {
6016      register *fp;
6017      fp = getf(u_n_ar0[R0]);
6018      if(fp == NULL)
6019          return;
6020      statl(ip->f_inode, u_n_arg[0]);
6021  }
6022  /* ----- */
6023  /*
6024  *
6025  * the stat system call.
6026  */
6027  /*
6028  stat()
6029  {
6030      register ip;
6031      extern uchar;
6032      ip = named(&uchar, 0);
6033      if(ip == NULL)
6034          return;
6035      statl(ip, u_n_arg[1]);
6036      iput(ip);
6037  }
6038  /* ----- */
6039  /*
6040  *
6041  * The basic routine for fstat and stat:
6042  * get the inode and pass appropriate parts back.
6043  */
6044  /*
6045  statl(ip, ub)
6046  int ip;
6047  {
6048      register i, *bp, *cp;
6049  }

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

6100  if(ip->i_count!=1 || (ip->i_mode&(IFBLK&IFCHR))!=0)
6101      goto out;
6102  smp = NULL;
6103  for(mp = &mount[0]; mp < &mount[NMOUNT]; mp++) {
6104      if(mp->m_bufp != NULL) {
6105          if(d == mp->m_dev)
6106              goto out;
6107      } else
6108          if(smp == NULL)
6109              smp = mp;
6110  }
6111  if(smp == NULL)
6112      goto out;
6113  (*bdevsw[d.d_major].d_open)(d, !u.u_arg[2]);
6114  if(u.u_error)
6115      goto out;
6116  mp = bread(d, 1);
6117  if(u.u_error) {
6118      brelse(mp);
6119      goto out1;
6120  }
6121  smp->m_inodp = ip;
6122  smp->m_dev = d;
6123  smp->m_bufp = getblk(NODEV);
6124  bcopy(mp->b_addr, smp->m_bufp->b_addr, 256);
6125  smp = smp->m_bufp->b_addr;
6126  smp->s_ilock = 0;
6127  smp->s_flock = 0;
6128  smp->s_ronly = u.u_arg[2] & 1;
6129  brelse(mp);
6130  ip->i_flag |= IMOUNT;
6131  prele(ip);
6132  return;
6133
6134 out:
6135  u.u_error = EBUSY;
6136 out1:
6137  iput(ip);
6138 }
6139 /* ----- */
6140 /*
6141 * the umount system call.
6142 */
6143 */
6144 sumount()
6145 {
6146     int d;
6147     register struct inode *ip;
6148     register struct mount *mp;
6149

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

6150     update();
6151     d = getmdev();
6152     if(u.u_error)
6153         return;
6154     for(mp = &mount[0]; mp < &mount[NMOUNT]; mp++)
6155         if(mp->m_bufp!=NULL && d==mp->m_dev)
6156             goto found;
6157     u.u_error = EINVAL;
6158     return;
6159
6160 found:
6161     for(ip = &inode[0]; ip < &inode[NINODE]; ip++)
6162         if(ip->i_number!=0 && d==ip->i_dev) {
6163             u.u_error = EBUSY;
6164             return;
6165         }
6166     (*bdevsw[d.d_major].d_close)(d, 0);
6167     ip = mp->m_inodp;
6168     ip->i_flag |= ~IMOUNT;
6169     iput(ip);
6170     ip = mp->m_bufp;
6171     mp->m_bufp = NULL;
6172     brelse(ip);
6173 }
6174 /* ----- */
6175 /*
6176 * Common code for mount and umount.
6177 * Check that the user's argument is a reasonable thing
6178 * on which to mount, and return the device number if so.
6179 */
6180 */
6181 getmdev()
6182 {
6183     register d, *ip;
6184     extern uchar;
6185
6186     ip = namei(&uchar, 0);
6187     if(ip == NULL)
6188         return;
6189     if((ip->i_mode&IFMT) != IFBLK)
6190         u.u_error = ENOTBLK;
6191     d = ip->i_addr[0];
6192     if(ip->i_addr[0].d_major >= nblkdev)
6193         u.u_error = ENXIO;
6194     iput(ip);
6195     return(d);
6196 }
6197 /* ----- */
6198
6199

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

6200 #
6201 /*
6202 */
6203 #include "../param.h"
6204 #include "../inode.h"
6205 #include "../user.h"
6206 #include "../buf.h"
6207 #include "../conf.h"
6208 #include "../system.h"
6209 #include
6210
6211 /*
6212 * Read the file corresponding to
6213 * the inode pointed at by the argument.
6214 * the actual read arguments are found
6215 * in the variables:
6216 * u_base
6217 * u_offset
6218 * u_count
6219 * u_segflg
6220 */
6221 read(aip)
6222 struct inode *aip;
6223 {
6224     int *bp;
6225     int lbn, bn, on;
6226     register dn, n;
6227     register struct inode *ip;
6228
6229     ip = aip;
6230     if(u_count == 0)
6231         return;
6232     ip->i_flag |= IACC;
6233     if(ip-<i_mode&IWRITE) == ICHR)
6234         (*cdvsw[ip->i_addr[0].d_major].d_read)(ip->i_addr[0]);
6235     return;
6236 }
6237
6238 do {
6239     lbn = bn = ip->i_offset[1] & 0777;
6240     on = u.u_offset[1] & 0777;
6241     n = min(512-on, u.u_count);
6242     if(ip-<i_mode&IWRITE) != IBLK)
6243         dn = dpcmp(ip->i_size&0377, ip-<i_size);
6244     if(dn >= 0)
6245         return;
6246     n = min(n, dn);
6247     if(bn == 0)
6248         return;
6249     if(bn == 0)
6250         return;
6251     dn = ip->i_dev;
6252     } else {
6253         dn = ip->i_addr[0];
6254         rblock = bn+1;
6255         if(ip->i_last+1 == lbn)
6256             bp = breada(dn, bn, rblock);
6257         else
6258             bp = bread(dn, bn);
6259         ip->i_last = lbn;
6260         lmove(bp, on, n, B_READ);
6261         brelse(bp);
6262     } while(u.u_error==0 && u.u_count!=0);
6263 } /* ----- */
6264
6265 /*
6266 * Write the file corresponding to
6267 * the inode pointed at by the argument.
6268 * the actual read arguments are found
6269 * in the variables:
6270 * u_base
6271 * u_offset
6272 * u_count
6273 * u_segflg
6274 * write to kernel/user
6275 */
6276 write(aip)
6277 struct inode *aip;
6278 {
6279     int *bp;
6280     int lbn, bn, on;
6281     register dn, n;
6282     register struct inode *ip;
6283     ip = aip;
6284     ip-<i_flag |= IACC|IUPD;
6285     if(ip-<i_mode&IWRITE) == ICHR)
6286         (*cdvsw[ip->i_addr[0].d_major].d_write)(ip->i_addr[0]);
6287     return;
6288     if(u_count == 0)
6289         return;
6290     if(u.u_offset == 0)
6291         return;
6292     bn = ip->i_offset[1] & 0777;
6293     do {
6294         dn = ip->i_offset[0] & 0777;
6295         on = u.u_offset[1] & 0777;
6296         n = min(512-on, u.u_count);
6297         if(ip-<i_mode&IWRITE) != IBLK)
6298             return;
6299         if(bn == 0)
6300             return;
6301         if(bn == 0)
6302             return;
6303         dn = ip->i_dev;
6304         rblock = bn+1;
6305         if(ip->i_last+1 == lbn)
6306             bp = breada(dn, bn, rblock);
6307         else
6308             bp = bread(dn, bn);
6309         ip->i_last = lbn;
6310         lmove(bp, on, n, B_READ);
6311         brelse(bp);
6312     } while(u.u_error==0 && u.u_count!=0);
6313 } /* ----- */
6314
6315 /*
6316 * the actual read arguments are found
6317 * in the variables:
6318 * u_base
6319 * u_offset
6320 * u_count
6321 * u_segflg
6322 * read to kernel/user
6323 */
6324 read(aip)
6325 struct inode *aip;
6326 {
6327     int *bp;
6328     int lbn, bn, on;
6329     register dn, n;
6330     register struct inode *ip;
6331     ip = aip;
6332     if(u_count == 0)
6333         return;
6334     ip->i_flag |= IACC;
6335     if(ip-<i_mode&IWRITE) == ICHR)
6336         (*cdvsw[ip->i_addr[0].d_major].d_read)(ip->i_addr[0]);
6337     return;
6338 }
6339
6340 do {
6341     lbn = bn = ip->i_offset[1] & 0777;
6342     on = u.u_offset[1] & 0777;
6343     n = min(512-on, u.u_count);
6344     if(ip-<i_mode&IWRITE) != IBLK)
6345         dn = dpcmp(ip->i_size&0377, ip-<i_size);
6346     if(dn >= 0)
6347         return;
6348     n = min(n, dn);
6349     if(bn == 0)
6350         return;
6351     if(bn == 0)
6352         return;
6353     dn = ip->i_dev;
6354     } else {
6355         dn = ip->i_addr[0];
6356         rblock = bn+1;
6357         if(ip->i_last+1 == lbn)
6358             bp = breada(dn, bn, rblock);
6359         else
6360             bp = bread(dn, bn);
6361         ip->i_last = lbn;
6362         lmove(bp, on, n, B_READ);
6363         brelse(bp);
6364     } while(u.u_error==0 && u.u_count!=0);
6365 } /* ----- */
6366
6367 /*
6368 * Write the file corresponding to
6369 * the inode pointed at by the argument.
6370 * the actual read arguments are found
6371 * in the variables:
6372 * u_base
6373 * u_offset
6374 * u_count
6375 * u_segflg
6376 * write to kernel/user
6377 */
6378 write(aip)
6379 struct inode *aip;
6380 {
6381     int *bp;
6382     int lbn, bn, on;
6383     register dn, n;
6384     register struct inode *ip;
6385     ip = aip;
6386     ip-<i_flag |= IACC|IUPD;
6387     if(ip-<i_mode&IWRITE) == ICHR)
6388         (*cdvsw[ip->i_addr[0].d_major].d_write)(ip->i_addr[0]);
6389     return;
6390     if(u_count == 0)
6391         return;
6392     if(u.u_offset == 0)
6393         return;
6394     bn = ip->i_offset[1] & 0777;
6395     do {
6396         dn = ip->i_offset[0] & 0777;
6397         on = u.u_offset[1] & 0777;
6398         n = min(512-on, u.u_count);
6399         if(ip-<i_mode&IWRITE) != IBLK)
6400             return;
6401         if(bn == 0)
6402             return;
6403         if(bn == 0)
6404             return;
6405         dn = ip->i_dev;
6406         rblock = bn+1;
6407         if(ip->i_last+1 == lbn)
6408             bp = breada(dn, bn, rblock);
6409         else
6410             bp = bread(dn, bn);
6411         ip->i_last = lbn;
6412         lmove(bp, on, n, B_READ);
6413         brelse(bp);
6414     } while(u.u_error==0 && u.u_count!=0);
6415 } /* ----- */

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976



```

6300         dn = ip->i_dev;
6301     } else
6302         dn = ip->i_addr[0];
6303     if(n == 512)
6304         bp = getblk(dn, bn); else
6305         bp = bread(dn, bn);
6306     iomove(bp, on, n, B_WRITE);
6307     if(u.u_error != 0)
6308         brelse(bp); else
6309     if ((u.u_offset[1]&0777)==0)
6310         bawrite(bp); else
6311         bdwrite(bp);
6312     if(dpcmp(ip->i_size0&0377, ip->i_size1,
6313         u.u_offset[0], u.u_offset[1]) < 0 &&
6314         (ip->i_mode&(IFBLK&IFCHR)) == 0) {
6315         ip->i_size0 = u.u_offset[0];
6316         ip->i_size1 = u.u_offset[1];
6317     }
6318     ip->i_flag |= IUPD;
6319 } while(u.u_error==0 && u.u_count!=0);
6320 }
6321 /* ----- */
6322
6323 /* Return the logical maximum
6324 * of the 2 arguments.
6325 */
6326 max(a, b)
6327 char *a, *b;
6328 {
6329
6330     if(a > b)
6331         return(a);
6332     return(b);
6333 }
6334 /* ----- */
6335
6336 /* Return the logical minimum
6337 * of the 2 arguments.
6338 */
6339 min(a, b)
6340 char *a, *b;
6341 {
6342
6343     if(a < b)
6344         return(a);
6345     return(b);
6346 }
6347 /* ----- */
6348
6349

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

6350 /* Move 'an' bytes at byte location
6351 * &bp->b_addr[o] to/from (flag) the
6352 * user/kernel (u.segflg) area starting at u.base.
6353 * Update all the arguments by the number
6354 * of bytes moved.
6355 *
6356 * There are 2 algorithms,
6357 * if source address, dest address and count
6358 * are all even in a user copy,
6359 * then the machine language copyin/copyout
6360 * is called.
6361 * If not, its done byte-by-byte with
6362 * cpass and passc.
6363 */
6364 iomove(bp, o, an, flag)
6365 struct buf *bp;
6366 {
6367     register char *cp;
6368     register int n, t;
6369
6370     n = an;
6371     cp = bp->b_addr + o;
6372     if(u.u_segflg==0 && ((n | cp | u.u_base)&01)==0) {
6373         if (flag==B_WRITE)
6374             cp = copyin(u.u_base, cp, n);
6375         else
6376             cp = copyout(cp, u.u_base, n);
6377         if (cp) {
6378             u.u_error = EFAULT;
6379             return;
6380         }
6381         u.u_base += n;
6382         dpadd(u.u_offset, n);
6383         u.u_count -= n;
6384         return;
6385     }
6386     if (flag==B_WRITE) {
6387         while(n--) {
6388             if ((t = cpass()) < 0)
6389                 return;
6390             *cp++ = t;
6391         }
6392     } else
6393         while (n--)
6394             if (passc(*cp++) < 0)
6395                 return;
6396 }
6397 /* ----- */
6398
6399

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

6400 #
6401 #include "../param.h"
6402 #include "../conf.h"
6403 #include "../inode.h"
6404 #include "../user.h"
6405 #include "../buf.h"
6406 #include "../system.h"
6407
6408 /* Bmap defines the structure of file system storage
6409  * by returning the physical block number on a device given
6410  * the inode and the logical block number in a file.
6411  * When convenient, it also leaves the physical
6412  * block number of the next block of the file in rblock
6413  * for use in read-ahead.
6414  */
6415 bmap(ip, bn)
6416 struct inode *ip;
6417 int bn;
6418 {
6419     register *bp, *bap, nb;
6420     int *nbp, d, i;
6421     d = ip->i_dev;
6422     if(bn < ~077777) {
6423         u_error = EFBIG;
6424         return(0);
6425     }
6426     if(ip->i_mode&ILARG) == 0) {
6427         /* small file algorithm */
6428         if((bn & ~7) == 0) {
6429             /* convert small to large */
6430             if((bp = alloc(d)) == NULL)
6431                 return(NULL);
6432             *bap++ = ip->i_addr[i++];
6433             for(i=0; i<8; i++) {
6434                 bap = bp->b_addr;
6435                 return(NULL);
6436             }
6437             ip->i_addr[0] = bp->b_blkno;
6438             bwrite(bp);
6439             ip->i_mode |= ILARG;
6440             goto large;
6441         }
6442         nb = ip->i_addr[bn];
6443         if(nb == 0 && (bp = alloc(d)) != NULL) {
6444             bwrite(bp);
6445         }
6446     }
6447 }
6448
6449

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

6450 nb = bp->b_blkno;
6451 ip->i_addr[bn] = nb;
6452 ip->i_flag |= IUPD;
6453 }
6454 rblock = 0;
6455 if (bn>7)
6456     rblock = ip->i_addr[bn+1];
6457     return(nb);
6458 }
6459 /* large file algorithm */
6460 large:
6461     i = bn>8;
6462     if(bn & 0174000)
6463         i = 7;
6464     if((nb=ip->i_addr[i]) == 0) {
6465         ip->i_flag |= IUPD;
6466         if (bp = alloc(d)) == NULL)
6467             return(NULL);
6468         ip->i_addr[i] = bp->b_blkno;
6469         return(NULL);
6470     } else
6471         bp = bread(d, nb);
6472     bap = bp->b_addr;
6473     /* "huge" fetch of double indirect block */
6474     if(i == 7) {
6475         i = (bn>8) & 0377 - 7;
6476         if((nb=bap[i]) == 0) {
6477             if((nbp = alloc(d)) == NULL) {
6478                 bwrite(bp);
6479                 return(NULL);
6480             }
6481             bap[i] = nbp->b_blkno;
6482             bwrite(bp);
6483             /* else {
6484                 bwrite(bp);
6485                 bwrite(bp);
6486                 nbp = bread(d, nb);
6487             }
6488             bp = nbp;
6489             bap = bp->b_addr;
6490         }
6491         /* normal indirect fetch */
6492         i = bn & 0377;
6493         if((nb=bap[i]) == 0 && (nbp = alloc(d)) != NULL) {
6494             nb = nbp->b_blkno;
6495             bap[i] = nb;
6496         }
6497     }
6498 }
6499

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

6500         bdwrite(nbp);
6501         bdwrite(bp);
6502     } else
6503         brelse(bp);
6504     rablock = 0;
6505     if(i < 255)
6506         rablock = bap[i+1];
6507     return(nb);
6508 }
6509 /* ----- */
6510
6511 /* Pass back c to the user at his location u_base;
6512  * update u_base, u_count, and u_offset. Return -1
6513  * on the last character of the user's read.
6514  * u_base is in the user address space unless u_segflg
6515  * is set.
6516  */
6517 passc(c)
6518 char c;
6519 {
6520
6521     if(u.u_segflg)
6522         *u.u_base = c; else
6523         if(subyte(u.u_base, c) < 0) {
6524             u.u_error = EFAULT;
6525             return(-1);
6526         }
6527     u.u_count--;
6528     if(++u.u_offset[1] == 0)
6529         u.u_offset[0]++;
6530     u.u_base++;
6531     return(u.u_count == 0? -1: 0);
6532 }
6533 /* ----- */
6534
6535 /*
6536  * Pick up and return the next character from the user's
6537  * write call at location u_base;
6538  * update u_base, u_count, and u_offset. Return -1
6539  * when u_count is exhausted. u_base is in the user's
6540  * address space unless u_segflg is set.
6541  */
6542 cpass()
6543 {
6544     register c;
6545
6546     if(u.u_count == 0)
6547         return(-1);
6548     if(u.u_segflg)
6549         c = *u.u_base; else

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

6550         if((c=fubyte(u.u_base)) < 0) {
6551             u.u_error = EFAULT;
6552             return(-1);
6553         }
6554     u.u_count--;
6555     if(++u.u_offset[1] == 0)
6556         u.u_offset[0]++;
6557     u.u_base++;
6558     return(c&0377);
6559 }
6560 /* ----- */
6561
6562 /*
6563  * Routine which sets a user error; placed in
6564  * illegal entries in the bdevsw and cdevsw tables.
6565  */
6566 nodev()
6567 {
6568     u.u_error = ENODEV;
6569 }
6570 /* ----- */
6571
6572 /*
6573  * Null routine; placed in insignificant entries
6574  * in the bdevsw and cdevsw tables.
6575  */
6576 nulldev()
6577 {
6578 }
6579 /* ----- */
6580
6581 /*
6582  * copy count words from from to to.
6583  */
6584 bcopy(from, to, count)
6585 int *from, *to;
6586 {
6587     register *a, *b, c;
6588
6589     a = from;
6590     b = to;
6591     c = count;
6592     do
6593         *b++ = *a++;
6594     while(--c);
6595 }
6596 /* ----- */
6597
6598
6599

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

6600 #
6601 /*
6602 */
6603 #include "../param.h"
6604 #include "../user.h"
6605 #include "../fills.h"
6606 #include "../file.h"
6607 #include "../conf.h"
6608 #include "../inode.h"
6609 #include "../reg.h"
6610 #include "reg.h"
6611 /*
6612 */
6613 * Convert a user supplied
6614 * file descriptor into a pointer
6615 * to a file structure.
6616 * Only task is to check range
6617 * of the descriptor.
6618 */
6619 getf(f)
6620 {
6621     register *fp, rf;
6622     rf = f;
6623     if (rf < 0 || rf == NOFILE)
6624         goto bad;
6625     fp = u.n_ofile[rf];
6626     if (fp != NULL)
6627         return (fp);
6628     bad:
6629     u.n_error = EBADF;
6630     return (NULL);
6631 }
6632 /*
6633 */
6634 /*
6635 */
6636 * Internal form of close.
6637 * Decrement reference count on
6638 * file structure and call close
6639 * on last close.
6640 * Also make sure the pipe protocol
6641 * does not constipate.
6642 */
6643 closef(fp)
6644 int *ip;
6645 {
6646     register *rip, *ip;
6647     rip = fp;
6648     if (rip->f_flags&FPIPE) {
6649

```

Reproduced under license from the Western Electric Company, NY

Copyright, J. Lions, 1976

```

6650 ip = rip->f_inode;
6651 ip->i_mode = &~(IRBAD|IWRITE);
6652 wakeup(ip+1);
6653 wakeup(ip+2);
6654 }
6655 if (rip->f_count <= 1)
6656     closef(rip->f_inode, rip->f_flags&FWRITE);
6657     rip->f_count--;
6658 }
6659 /*
6660 */
6661 /*
6662 * Decrement reference count on an
6663 * inode due to the removal of a
6664 * referencing file structure.
6665 * On the last close, switchout
6666 * to the close entry point of special
6667 * device handler.
6668 * Note that the handler gets called
6669 * on every open and only on the last
6670 * close.
6671 */
6672 closef(ip, rw)
6673 int *ip;
6674 {
6675     register *rip;
6676     register dev, maj;
6677     rip = ip;
6678     dev = rip->i_addr[0];
6679     maj = rip->i_addr[0].d_major;
6680     if (rip->i_count <= 1)
6681         switch(rip->i_mode&IFMT) {
6682
6683     case IFCHR:
6684         (*cdevsw[maj].d_close)(dev, rw);
6685         break;
6686     case IFBLK:
6687         (*bdevsw[maj].d_close)(dev, rw);
6688     }
6689     iput(rip);
6690 }
6691 /*
6692 */
6693 /*
6694 */
6695 /*
6696 * openf called to allow handler
6697 * of special files to initialize and
6698 * validate before actual IO.
6699 * Called on all sorts of opens

```

Reproduced under license from the Western Electric Company, NY

Copyright, J. Lions, 1976

```

6700 * and also on mount.
6701 */
6702 openi(ip, rw)
6703 int *ip;
6704 {
6705     register *rip;
6706     register dev, maj;
6707
6708     rip = ip;
6709     dev = rip->i_addr[0];
6710     maj = rip->i_addr[0].d_major;
6711     switch(rip->i_mode&IFMT) {
6712
6713     case IFCHR:
6714         if(maj >= nchrdev)
6715             goto bad;
6716         (*cdevsw[maj].d_open)(dev, rw);
6717         break;
6718
6719     case IFBLK:
6720         if(maj >= nblkdev)
6721             goto bad;
6722         (*bdevsw[maj].d_open)(dev, rw);
6723     }
6724     return;
6725
6726 bad:
6727     u.u_error = ENXIO;
6728 }
6729 /* ----- */
6730
6731 /*
6732 * Check mode permission on inode pointer.
6733 * Mode is READ, WRITE, or EXEC.
6734 * In the case of WRITE, the
6735 * read-only status of the file
6736 * system is checked.
6737 * Also in WRITE, prototype text
6738 * segments cannot be written.
6739 * The mode is shifted to select
6740 * the owner/group/other fields.
6741 * The super user is granted all
6742 * permissions except for EXEC where
6743 * at least one of the EXEC bits must
6744 * be on.
6745 */
6746 access(aip, mode)
6747 int *aip;
6748 {
6749     register *ip, m;

```

```

6750
6751     ip = aip;
6752     m = mode;
6753     if(m == IWRITE) {
6754         if(getfs(ip->i_dev)->s_ronly != 0) {
6755             u.u_error = EROFS;
6756             return(1);
6757         }
6758         if(ip->i_flag & ITEXT) {
6759             u.u_error = ETXTBSY;
6760             return(1);
6761         }
6762     }
6763     if(u.u_uid == 0) {
6764         if(m == IEXEC && (ip->i_mode &
6765             (IEXEC | (IEXEC>>3) | (IEXEC>>6))) == 0)
6766             goto bad;
6767         return(0);
6768     }
6769     if(u.u_uid != ip->i_uid) {
6770         m =>> 3;
6771         if(u.u_gid != ip->i_gid)
6772             m =>> 3;
6773     }
6774     if((ip->i_mode&m) != 0)
6775         return(0);
6776
6777 bad:
6778     u.u_error = EACCES;
6779     return(1);
6780 }
6781 /* ----- */
6782
6783 /*
6784 * Look up a pathname and test if
6785 * the resultant inode is owned by the
6786 * current user.
6787 * If not, try for super-user.
6788 * If permission is granted,
6789 * return inode pointer.
6790 */
6791 owner()
6792 {
6793     register struct inode *ip;
6794     extern uchar();
6795
6796     if ((ip = namei(uchar, 0)) == NULL)
6797         return(NULL);
6798     if(u.u_uid == ip->i_uid)
6799         return(ip);

```

```

6800 if (user())
6801     return(ip);
6802     iput(ip);
6803     return(NULL);
6804 }
6805 /* ----- */
6806
6807 /*
6808 * Test if the current user is the
6809 * super user.
6810 */
6811 user()
6812 {
6813     if (u_uid == 0)
6814         return(1);
6815     return(1);
6816     u_n_error = EPERM;
6817     return(0);
6818 }
6819 /* ----- */
6820
6821 /*
6822 * Allocate a user file descriptor.
6823 */
6824 ufailloc()
6825 {
6826     register i;
6827     for (i=0; i<NORFILE; i++)
6828         if (u_offile[i] == NULL)
6829             {
6830                 u_n_ar0[R0] = i;
6831                 return(i);
6832             }
6833     u_n_error = EMFILE;
6834     return(-1);
6835 }
6836 /* ----- */
6837
6838 /*
6839 * Allocate a user file descriptor
6840 * and a file structure.
6841 * Initialize the descriptor
6842 * to point at the file structure.
6843 *
6844 * no file -- if there are no available
6845 * file structures.
6846 */
6847 failloc()
6848 {
6849     register struct file *fp;

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

6850 register i;
6851     if ((i = ufailloc()) > 0)
6852         return(NULL);
6853     for (fp = &file[0]; fp < &file[NFILE]; fp++)
6854         if (fp->f_count==0)
6855             {
6856                 u_n_offile[i] = fp;
6857                 fp->f_count++;
6858                 fp->f_offset[0] = 0;
6859                 fp->f_offset[1] = 0;
6860                 return(fp);
6861             }
6862     printf("no file\n");
6863     u_n_error = ENFILE;
6864     return(NULL);
6865 }
6866 /* ----- */

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

6900 #
6901 /*
6902 */
6903
6904 #include "../param.h"
6905 #include "../system.h"
6906 #include "../filesystem.h"
6907 #include "../conf.h"
6908 #include "../buf.h"
6909 #include "../inode.h"
6910 #include "../user.h"
6911
6912 /*
6913  * iinit is called once (from main)
6914  * very early in initialization.
6915  * It reads the root's super block
6916  * and initializes the current date
6917  * from the last modified date.
6918  *
6919  * panic: iinit -- cannot read the super
6920  * block. Usually because of an IO error.
6921  */
6922 iinit()
6923 {
6924     register *cp, *bp;
6925
6926     (*bdevsw[rootdev.d_major].d_open)(rootdev, 1);
6927     bp = bread(rootdev, 1);
6928     cp = getblk(NODEV);
6929     if(u.u_error)
6930         panic("iinit");
6931     bcopy(bp->b_addr, cp->b_addr, 256);
6932     brelse(bp);
6933     mount[0].m_bufp = cp;
6934     mount[0].m_dev = rootdev;
6935     cp = cp->b_addr;
6936     cp->s_flock = 0;
6937     cp->s_iloc = 0;
6938     cp->s_ronly = 0;
6939     time[0] = cp->s_time[0];
6940     time[1] = cp->s_time[1];
6941 }
6942 /* ----- */
6943 /* ----- */
6944
6945 /*
6946  * alloc will obtain the next available
6947  * free disk block from the free list of
6948  * the specified device.
6949  * The super block has up to 100 remembered

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

6950  * free blocks; the last of these is read to
6951  * obtain 100 more . . .
6952  *
6953  * no space on dev x/y -- when
6954  * the free list is exhausted.
6955  */
6956 alloc(dev)
6957 {
6958     int bno;
6959     register *bp, *ip, *fp;
6960
6961     fp = getfs(dev);
6962     while(fp->s_flock)
6963         sleep(&fp->s_flock, PINOD);
6964     do {
6965         if(fp->s_nfree <= 0)
6966             goto nospace;
6967         bno = fp->s_free[--fp->s_nfree];
6968         if(bno == 0)
6969             goto nospace;
6970     } while (badblock(fp, bno, dev));
6971     if(fp->s_nfree <= 0) {
6972         fp->s_flock++;
6973         bp = bread(dev, bno);
6974         ip = bp->b_addr;
6975         fp->s_nfree = *ip++;
6976         bcopy(ip, fp->s_free, 100);
6977         brelse(bp);
6978         fp->s_flock = 0;
6979         wakeup(&fp->s_flock);
6980     }
6981     bp = getblk(dev, bno);
6982     clrbuf(bp);
6983     fp->s_fmod = 1;
6984     return(bp);
6985
6986 nospace:
6987     fp->s_nfree = 0;
6988     prdev("no space", dev);
6989     u.u_error = ENOSPC;
6990     return(NULL);
6991 }
6992 /*----- */
6993 /*----- */
6994
6995 /*
6996  * place the specified disk block
6997  * back on the free list of the
6998  * specified device.
6999  */

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

7000 free(dev, bno)
7001 {
7002     register *fp, *bp, *ip;
7003     fp = gets(dev);
7004     ip = gets(dev);
7005     fp->s_fmmod = 1;
7006     while(fp->s_flock)
7007         sleep(&fp->s_flock, PINOD);
7008     if (badblock(fp, bno, dev))
7009         return;
7010     if(fp->s_nfree <= 0) {
7011         fp->s_nfree = 1;
7012         fp->s_nfree[0] = 0;
7013     }
7014     if(fp->s_nfree >= 100) {
7015         fp->s_flock++;
7016         bp = getblk(dev, bno);
7017         ip = bp->b_addr;
7018         *ip++ = fp->s_nfree;
7019         bcopy(fp->s_nfree, ip, 100);
7020         fp->s_nfree = 0;
7021         bwrite(bp);
7022         fp->s_flock = 0;
7023         wakeup(&fp->s_flock);
7024     }
7025     fp->s_nfree[fp->s_nfree++] = bno;
7026     fp->s_fmmod = 1;
7027 }
7028 /* -----
7029 /* -----
7030
7031 /*
7032 * Check that a block number is in the
7033 * range between the I list and the size
7034 * of the device.
7035 * This is used mainly to check that a
7036 * garbage file system has not been mounted.
7037 *
7038 * bad block on dev x/y -- not in range
7039 */
7040 badblock(afp, abn, dev)
7041 {
7042     register struct filsys *fp;
7043     register char *bn;
7044     fp = afp;
7045     bn = abn;
7046     if (bn > fp->s_size+2 || bn <= fp->s_fsize)
7047         prdev("bad block", dev);
7048     return(1);
7049 }
    
```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

7050     }
7051     return(0);
7052 }
7053 /* -----
7054 /* -----
7055 */
7056 /*
7057 * Allocate an unused I node
7058 * on the specified device.
7059 * Used with file creation.
7060 * The algorithm keeps up to
7061 * 100 spare I node in the
7062 * super block. When this runs out,
7063 * a linear search through the
7064 * I list is instituted to pick
7065 * up 100 more.
7066 */
7067 alloc(dev)
7068 {
7069     register *fp, *bp, *ip;
7070     int i, j, k, lno;
7071     fp = gets(dev);
7072     while(fp->s_flock)
7073         sleep(&fp->s_flock, PINOD);
7074     loop:
7075         if(fp->s_ninode > 0) {
7076             lno = fp->s_ninode[fp->s_ninode];
7077             ip = lget(dev, lno);
7078             if (ip==NULL)
7079                 return(NULL);
7080             if(ip->l_mode == 0) {
7081                 for(bp = &ip->l_mode; bp < &ip->l_addr[8];)
7082                     *bp++ = 0;
7083             fp->s_fmmod = 1;
7084             return(ip);
7085         }
7086     /*
7087     * Inode was allocated after all.
7088     * Look some more.
7089     */
7090     iput(ip);
7091     goto loop;
7092 }
7093     fp->s_flock++;
7094     lno = 0;
7095     for(i=0; i<fp->s_size; i++) {
7096         bp = bread(dev, i+2);
7097         ip = bp->b_addr;
7098         for(j=0; j<256; j+=16) {
7099
    
```



```

7100         ino++;
7101         if(ip[j] != 0)
7102             continue;
7103         for(k=0; k<NINODE; k++)
7104             if(dev == inode[k].i_dev &&
7105                ino == inode[k].i_number)
7106                 goto cont;
7107         fp->s_inode[fp->s_ninode++] = ino;
7108         if(fp->s_ninode >= 100)
7109             break;
7110         cont:;
7111     }
7112     brelse(bp);
7113     if(fp->s_ninode >= 100)
7114         break;
7115 }
7116 fp->s_iloc = 0;
7117 wakeup(&fp->s_iloc);
7118 if (fp->s_ninode > 0)
7119     goto loop;
7120 prdev("Out of inodes", dev);
7121 u.u_error = ENOSPC;
7122 return(NULL);
7123 }
7124 /* ----- */
7125 /* ----- */
7126
7127 /*
7128  * Free the specified I node
7129  * on the specified device.
7130  * The algorithm stores up
7131  * to 100 I nodes in the super
7132  * block and throws away any more.
7133  */
7134 ifree(dev, ino)
7135 {
7136     register *fp;
7137
7138     fp = getfs(dev);
7139     if(fp->s_iloc)
7140         return;
7141     if(fp->s_ninode >= 100)
7142         return;
7143     fp->s_inode[fp->s_ninode++] = ino;
7144     fp->s_fmod = 1;
7145 }
7146 /* ----- */
7147 /* ----- */
7148
7149 /*

```

```

7150 * getfs maps a device number into
7151 * a pointer to the incore super
7152 * block.
7153 * The algorithm is a linear
7154 * search through the mount table.
7155 * A consistency check of the
7156 * in core free-block and i-node
7157 * counts.
7158 *
7159 * bad count on dev x/y -- the count
7160 * check failed. At this point, all
7161 * the counts are zeroed which will
7162 * almost certainly lead to "no space"
7163 * diagnostic
7164 * panic: no fs -- the device is not mounted.
7165 * this "cannot happen"
7166 */
7167 getfs(dev)
7168 {
7169     register struct mount *p;
7170     register char *n1, *n2;
7171
7172     for(p = &mount[0]; p < &mount[NMOUNT]; p++)
7173         if(p->m_bufp != NULL && p->m_dev == dev) {
7174             p = p->m_bufp->b_addr;
7175             n1 = p->s_nfree;
7176             n2 = p->s_ninode;
7177             if(n1 > 100 || n2 > 100) {
7178                 prdev("bad count", dev);
7179                 p->s_nfree = 0;
7180                 p->s_ninode = 0;
7181             }
7182             return(p);
7183         }
7184     panic("no fs");
7185 }
7186 /* ----- */
7187 /* ----- */
7188
7189 /*
7190 * update is the internal name of
7191 * 'sync'. It goes through the disk
7192 * queues to initiate sandbagged IO;
7193 * goes through the I nodes to write
7194 * modified nodes; and it goes through
7195 * the mount table to initiate modified
7196 * super blocks.
7197 */
7198
7199

```

```

7200 update()
7201 {
7202   register struct inode *ip;
7203   register struct mount *mp;
7204   register struct mount *mp;
7205   return;
7206   if(update)
7207     return;
7208   for(mp = &mount[NMOUNT]; mp > &mount[NMOUNT]; mp++)
7209     update(mp);
7210   if(mp->mnt_buftp != NULL)
7211     continue;
7212   if(ip->s_flocki=0 || ip->s_flocki=0)
7213     continue;
7214   if(ip->s_flocki=0 || ip->s_flocki=0)
7215     continue;
7216   bp = getblk(mp->mnt_dev, 1);
7217   ip->s_fmod = 0;
7218   ip->s_time[0] = time[0];
7219   ip->s_time[1] = time[1];
7220   bcopy(ip, bp->b_addr, 256);
7221   write(bp);
7222   for(ip = &inode[0]; ip < &inode[NINODE]; ip++)
7223     if(ip->flag & ILOCK)
7224       ip->flag = ILOCK;
7225   update(ip, time);
7226   prele(ip);
7227   }
7228   update = 0;
7229   bflush(NODEV);
7230 }
7231 /* -----
7232 /* -----
7233 */
7234 ip = NULL;
7235 for(p = &inode[0]; p < &inode[NINODE]; p++)
7236   if(dev==p->dev & ino==p->ino)
7237     if(p->flag & ILOCK)
7238       p->flag = ILOCK;
7239   sleep(p, PINOD);
7240   goto loop;
7241 }
7242 if(p->flag & ILOCK)
7243   for(ip = &mount[0];
7244     ip < &mount[NMOUNT]; ip++)
7245     if(ip->mnt_buftp == p)
7246       dev = ip->mnt_dev;
7247   ino = ROOTINO;
7248   goto loop;
7249 }

```

```

7250 #
7251 #include "param.h"
7252 #include "system.h"
7253 #include "user.h"
7254 #include "inode.h"
7255 #include "filesystem.h"
7256 #include "conf.h"
7257 #include "buf.h"
7258
7259 /*
7260 * Look up an inode by device, number.
7261 * If it is in core (in the inode structure),
7262 * honor the locking protocol.
7263 * If it is not in core, read it in from the
7264 * specified device.
7265 * If the inode is mounted on, perform
7266 * the indicated indirection.
7267 * In all cases, a pointer to a locked
7268 * inode structure is returned.
7269 *
7270 * printf warning: no inodes -- if the inode
7271 * structure is full
7272 * panic: no limit -- if the mounted file
7273 * system is not in the mount table.
7274 * "cannot happen"
7275 */
7276 lget(dev, ino)
7277 {
7278   register struct inode *p;
7279   register *ip2;
7280   int *ip1;
7281   register struct mount *ip;
7282   loop:
7283   ip = NULL;
7284   for(p = &inode[0]; p < &inode[NINODE]; p++)
7285     if(dev==p->dev & ino==p->ino)
7286       if(p->flag & ILOCK)
7287         p->flag = ILOCK;
7288     sleep(p, PINOD);
7289     goto loop;
7290 }
7291 if(p->flag & ILOCK)
7292   for(ip = &mount[0];
7293     ip < &mount[NMOUNT]; ip++)
7294     if(ip->mnt_buftp == p)
7295       dev = ip->mnt_dev;
7296   ino = ROOTINO;
7297   goto loop;
7298 }
7299 #

```

```

7300             panic("no imt");
7301         }
7302         p->i_count++;
7303         p->i_flag |= ILOCK;
7304         return(p);
7305     }
7306     if(ip==NULL && p->i_count==0)
7307         ip = p;
7308 }
7309 if((p=ip) == NULL) {
7310     printf("Inode table overflow\n");
7311     u.u_error = ENFILE;
7312     return(NULL);
7313 }
7314 p->i_dev = dev;
7315 p->i_number = ino;
7316 p->i_flag = ILOCK;
7317 p->i_count++;
7318 p->i_lastr = -1;
7319 ip = bread(dev, ldiv(ino+31,16));
7320 /*
7321  * Check I/O errors
7322  */
7323 if (ip->b_flags&B_ERROR) {
7324     brelse(ip);
7325     iput(p);
7326     return(NULL);
7327 }
7328 ip1 = ip->b_addr + 32*lrem(ino+31, 16);
7329 ip2 = &p->i_mode;
7330 while(ip2 < &p->i_addr[8])
7331     *ip2++ = *ip1++;
7332 brelse(ip);
7333 return(p);
7334 }
7335 /* ----- */
7336 /*
7337  * Decrement reference count of
7338  * an inode structure.
7339  * On the last reference,
7340  * write the inode out and if necessary,
7341  * truncate and deallocate the file.
7342  */
7343 iput(p)
7344 struct inode *p;
7345 {
7346     register *rp;
7347     rp = p;

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

7350     if(rp->i_count == 1) {
7351         rp->i_flag |= ILOCK;
7352         if(rp->i_nlink <= 0) {
7353             itrunc(rp);
7354             rp->i_mode = 0;
7355             ifree(rp->i_dev, rp->i_number);
7356         }
7357         iupdat(rp, time);
7358         prele(rp);
7359         rp->i_flag = 0;
7360         rp->i_number = 0;
7361     }
7362     rp->i_count--;
7363     prele(rp);
7364 }
7365 /* ----- */
7366 /*
7367  * Check accessed and update flags on
7368  * an inode structure.
7369  * If either is on, update the inode
7370  * with the corresponding dates
7371  * set to the argument tm.
7372  */
7373 iupdat(p, tm)
7374 int *p;
7375 int *tm;
7376 {
7377     register *ip1, *ip2, *rp;
7378     int *bp, i;
7379     rp = p;
7380     if((rp->i_flag&(IUPD|IACC)) != 0) {
7381         if(getfs(rp->i_dev)->s_ronly)
7382             return;
7383         i = rp->i_number+31;
7384         bp = bread(rp->i_dev, ldiv(i,16));
7385         ip1 = bp->b_addr + 32*lrem(i, 16);
7386         ip2 = &rp->i_mode;
7387         while(ip2 < &rp->i_addr[8])
7388             *ip1++ = *ip2++;
7389         if(rp->i_flag&IACC) {
7390             *ip1++ = time[0];
7391             *ip1++ = time[1];
7392         } else
7393             ip1 += 2;
7394         if(rp->i_flag&IUPD) {
7395             *ip1++ = *tm++;
7396             *ip1++ = *tm;
7397         }
7398     }
7399 }

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

7400 bwrite(bp);
7401 }
7402 /* ----- */
7403
7404 /*
7405 */
7406 * free all the disk blocks associated
7407 * with the specified inode structure.
7408 * The blocks of the file are removed
7409 * in reverse order. This FIFO
7410 * algorithm will tend to maintain
7411 * a contiguous free list much longer
7412 * than FIFO.
7413 */
7414 truncate(ip)
7415 int *ip;
7416 }
7417 register *rp, *bp, *cp;
7418 int *dp, *ep;
7419
7420 rp = ip;
7421 if((rp->i_mode&IFCHR&IRBLK) == 0)
7422 return;
7423 for(ip = rp->i_addr[7]; ip <= rp->i_addr[0]; ip--)
7424 if(*ip)
7425 if((rp->i_mode&IWRG) == 0)
7426 bp = bread(rp->i_dev, *ip);
7427 for(cp = bp->b_addr+512; cp <= bp->b_addr;
7428 cp--)
7429 if(*cp)
7430 if(ip == rp->i_addr[7])
7431 dp = bread(rp->i_dev, *cp);
7432 for(ep = dp->b_addr+512;
7433 ep <= dp->b_addr; ep--)
7434 if(*ep)
7435 free(rp->i_dev, *ep);
7436 bwrite(dp);
7437 }
7438 free(rp->i_dev, *cp);
7439 }
7440 bwrite(bp);
7441 }
7442 free(rp->i_dev, *ip);
7443 }
7444 }
7445 rp->i_mode = &~IWRG;
7446 rp->i_size = 0;
7447 rp->i_size1 = 0;
7448 rp->i_flag = IUPD;
7449 }

```

Reproduced under license from the Western Electric Company, NY  
Copyright, 1976

```

7450 /* ----- */
7451 /*
7452 */
7453 * Make a new file.
7454 */
7455 maknode(mode)
7456 {
7457 register *ip;
7458 ip = ialloc(n.u_pdir->i_dev);
7459 if (ip == NULL)
7460 return(NULL);
7461 return(NULL);
7462 ip->i_flag = IACC|IUPD;
7463 ip->i_mode = mode|IALLOC;
7464 ip->i_nlink = 1;
7465 ip->i_uid = n.u_uid;
7466 ip->i_gid = n.u_gid;
7467 wdir(ip);
7468 return(ip);
7469 }
7470 /* ----- */
7471 /*
7472 */
7473 * Write a directory entry with
7474 * parameters left as side effects
7475 * to a call to namei.
7476 */
7477 wdir(ip)
7478 int *ip;
7479 {
7480 register char *cp1, *cp2;
7481 n.u_dent.u_ino = ip->i_number;
7482 cp1 = n.u_dent.u_name[0];
7483 for(cp2 = n.u_dbuf[0]; cp2 < n.u_dbuf[DIRSIZ];)
7484 *cp1++ = *cp2++;
7485 n.u_count = DIRSIZ+2;
7486 n.u_segflg = 1;
7487 n.u_base = n.u_dent;
7488 wwrite(n.u_pdir);
7489 iput(n.u_pdir);
7490 }
7491 /* ----- */
7492
7493
7494
7495
7496
7497
7498
7499

```

Reproduced under license from the Western Electric Company, NY  
Copyright, 1976

```

7500 #
7501 #include "../param.h"
7502 #include "../inode.h"
7503 #include "../user.h"
7504 #include "../system.h"
7505 #include "../buf.h"
7506
7507 /*
7508  * Convert a pathname into a pointer to
7509  * an inode. Note that the inode is locked.
7510  *
7511  * func = function called to get next char of name
7512  * &uchar if name is in user space
7513  * &schar if name is in system space
7514  * flag = 0 if name is sought
7515  * 1 if name is to be created
7516  * 2 if name is to be deleted
7517  */
7518 namei(func, flag)
7519 int (*func)();
7520 {
7521     register struct inode *dp;
7522     register c;
7523     register char *cp;
7524     int eo, *bp;
7525
7526     /*
7527     * If name starts with '/' start from
7528     * root: otherwise start from current dir.
7529     */
7530
7531     dp = u.u_cdir;
7532     if((c=(*func)()) == '/');
7533         dp = rootdir;
7534     iget(dp->i_dev, dp->i_number);
7535     while(c == '/')
7536         c = (*func)();
7537     if(c == '\0' && flag != 0) {
7538         u.u_error = ENOENT;
7539         goto out;
7540     }
7541
7542 cloop:
7543     /*
7544     * Here dp contains pointer
7545     * to last component matched.
7546     */
7547
7548     if(u.u_error)
7549         goto out;

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

7550     if(c == '\0')
7551         return(dp);
7552
7553     /*
7554     * If there is another component,
7555     * dp must be a directory and
7556     * must have x permission.
7557     */
7558
7559     if((dp->i_mode&IFMT) != IFDIR) {
7560         u.u_error = ENOTDIR;
7561         goto out;
7562     }
7563     if(access(dp, IEXEC))
7564         goto out;
7565
7566     /* Gather up name into
7567     * users' dir buffer.
7568     */
7569
7570     cp = &u.u_dbuf[0];
7571     while(c!=='/' && c!='\0' && u.u_error==0) {
7572         if(cp < &u.u_dbuf[DIRSIZ])
7573             *cp++ = c;
7574         c = (*func)();
7575     }
7576     while(cp < &u.u_dbuf[DIRSIZ])
7577         *cp++ = '\0';
7578     while(c == '/')
7579         c = (*func)();
7580     if(u.u_error)
7581         goto out;
7582
7583     /* Set up to search a directory. */
7584
7585     u.u_offset[1] = 0;
7586     u.u_offset[0] = 0;
7587     u.u_segflg = 1;
7588     eo = 0;
7589     u.u_count = ldiv(dp->i_size1, DIRSIZ+2);
7590     bp = NULL;
7591
7592 eloop:
7593
7594     /*
7595     * If at the end of the directory,
7596     * the search failed. Report what
7597     * is appropriate as per flag.
7598     */
7599

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

7600 if(u_n_count == 0)
7601     if(bp != NULL)
7602         brelse(bp);
7603     if(flag==1 && c=='\0')
7604         {
7605             if(access(dp, IWRITE))
7606                 goto out;
7607             u_n_pdir = dp;
7608             if(eo)
7609                 u_n_offset[1] = eo-DIRISIZ-2; else
7610                     dp->i_flag = IUPD;
7611             return(NULL);
7612         }
7613     u_n_error = ENOENT;
7614     goto out;
7615 }
7616 /*
7617 * If offset is on a block boundary,
7618 * read the next directory block.
7619 * Release previous if it exists.
7620 */
7621 if((u_n_offset[1]&0777) == 0)
7622     {
7623         if(bp != NULL)
7624             brelse(bp);
7625         bp = bread(dp->i_dev,
7626             bmap(dp, idiv(u_n_offset[1], 512)));
7627     }
7628 /* Note first empty directory slot
7629 * in eo for possible creat.
7630 * String compare the directory entry
7631 * and the current component.
7632 * If they do not match, go back to loop.
7633 */
7634 /*
7635 bcopy(bp->b_addr+(u_n_offset[1]&0777), &u_n_dent,
7636     (DIRISIZ+2)/2);
7637 u_n_offset[1] += DIRISIZ+2;
7638 u_n_count--;
7639 if(u_n_dent.u_ino == 0)
7640     {
7641         if(eo == 0)
7642             goto loop;
7643     }
7644 for(cp = &u_n_dbuf[0]; cp < &u_n_dbuf[DIRISIZ]; cp++)
7645     if(*cp != cp[u_n_dent.u_name - u_n_dbuf])
7646         goto loop;
7647 }
7648
7649

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

7650 /* Here a component matched is a directory.
7651 * If there is more pathname, go back to
7652 * loop, otherwise return.
7653 */
7654 if(bp != NULL)
7655     brelse(bp);
7656     if(flag==2 && c=='\0')
7657         {
7658             if(access(dp, IWRITE))
7659                 goto out;
7660             return(dp);
7661         }
7662     bp = dp->i_dev;
7663     input(dp);
7664     dp =iget(bp, u_n_dent.u_ino);
7665     if(dp == NULL)
7666         return(NULL);
7667     goto loop;
7668 out:
7669     input(dp);
7670     return(NULL);
7671 }
7672 /*
7673 * Return the next character from the
7674 * kernel string pointed at by dirp.
7675 */
7676 * Return the next character from the
7677 * kernel string pointed at by dirp.
7678 */
7679 schar()
7680 {
7681     return(*u_n_dirp++ & 0377);
7682 }
7683 /*
7684 * Return the next character from the
7685 * user string pointed at by dirp.
7686 */
7687 * Return the next character from the
7688 * user string pointed at by dirp.
7689 */
7690 uchar()
7691 {
7692     register c;
7693     c = fubyte(u_n_dirp++);
7694     if(c == -1)
7695         u_n_error = EFAULT;
7696     return(c);
7697 }
7698 /*
7699

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

7700 #include "../param.h"
7701 #include "../system.h"
7702 #include "../user.h"
7703 #include "../inode.h"
7704 #include "../file.h"
7705 #include "../reg.h"
7706
7707 /* Max allowable buffering per pipe.
7708 * This is also the max size of the
7709 * file created to implement the pipe.
7710 * If this size is bigger than 4096,
7711 * pipes will be implemented in LARGE
7712 * files, which is probably not good.
7713 */
7714
7715 #define PIPSIZ 4096
7716
7717 /* The sys-pipe entry.
7718 * Allocate an inode on the root device.
7719 * Allocate 2 file structures.
7720 * Put it all together with flags.
7721 */
7722
7723 pipe()
7724 {
7725     register *ip, *rf, *wf;
7726     int r;
7727
7728     ip = ialloc(rootdev);
7729     if(ip == NULL)
7730         return;
7731     rf = falloc();
7732     if(rf == NULL) {
7733         iput(ip);
7734         return;
7735     }
7736     r = u.u_ar0[R0];
7737     wf = falloc();
7738     if(wf == NULL) {
7739         rf->f_count = 0;
7740         u.u_ofile[r] = NULL;
7741         iput(ip);
7742         return;
7743     }
7744     u.u_ar0[R1] = u.u_ar0[R0];
7745     u.u_ar0[R0] = r;
7746     wf->f_flag = FWRITE|FPIPE;
7747     wf->f_inode = ip;
7748     rf->f_flag = FREAD|FPIPE;
7749     rf->f_inode = ip;

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

7750     ip->i_count = 2;
7751     ip->i_flag = IACC|IUPD;
7752     ip->i_mode = IALLOC;
7753 }
7754 /* ----- */
7755
7756 /* Read call directed to a pipe.
7757 */
7758 readp(fp)
7759 int *fp;
7760 {
7761     register *rp, *ip;
7762
7763     rp = fp;
7764     ip = rp->f_inode;
7765 loop:
7766     /* Very conservative locking.
7767     */
7768     plock(ip);
7769     /* If the head (read) has caught up with
7770     * the tail (write), reset both to 0.
7771     */
7772     if(rp->f_offset[1] == ip->i_size1) {
7773         if(rp->f_offset[1] != 0) {
7774             rp->f_offset[1] = 0;
7775             ip->i_size1 = 0;
7776             if(ip->i_mode & IWRITE) {
7777                 ip->i_mode = & ~IWRITE;
7778                 wakeup(ip+1);
7779             }
7780         }
7781
7782         /* If there are not both reader and
7783         * writer active, return without
7784         * satisfying read.
7785         */
7786         prele(ip);
7787         if(ip->i_count < 2)
7788             return;
7789         ip->i_mode = | IREAD;
7790         sleep(ip+2, PPIPE);
7791         goto loop;
7792     }
7793     /* Read and return
7794     */
7795     u.u_offset[0] = 0;
7796     u.u_offset[1] = rp->f_offset[1];
7797     readi(ip);
7798     rp->f_offset[1] = u.u_offset[1];
7799     prele(ip);

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

7800 }
7801 /* ----- */
7802 /* Write call directed to a pipe.
7803 */
7804 writep(fp)
7805 {
7806     register *rp, *ip, c;
7807     rp = fp;
7808     ip = rp->f_inode;
7809     c = n_u_count;
7810     loop:
7811     /* If all done, return.
7812     */
7813     plock(ip);
7814     if(c == 0)
7815     {
7816         prele(ip);
7817         n_u_count = 0;
7818         return;
7819     }
7820     /* If there are not both read and
7821     * write sides of the pipe active,
7822     * return error and signal too.
7823     */
7824     if(ip->count < 2)
7825     {
7826         prele(ip);
7827         n_u_error = EPIPE;
7828         psignal(n_u_proc, SIGPIPE);
7829         return;
7830     }
7831     /* If the pipe is full,
7832     * wait for reads to delete
7833     * and truncate it.
7834     */
7835     if(ip->size == PIPESZ)
7836     {
7837         ip->mode |= IWRITE;
7838         prele(ip);
7839         sleep(ip+1, PIPB);
7840         goto loop;
7841     }
7842     /* Write what is possible and
7843     * loop back.
7844     */
7845     n_u_offset[0] = 0;
7846     n_u_offset[1] = ip->size;
7847     n_u_count = min(c, PIPESZ-n_u_offset[1]);
7848     write(ip);
7849     prele(ip);

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

7850     if(ip->mode&IRRD) {
7851         ip->mode = &~IRRD;
7852         wakeup(ip+2);
7853     }
7854     goto loop;
7855 }
7856 /* ----- */
7857 /* Lock a pipe.
7858 * If its already locked,
7859 * set the WANT bit and sleep.
7860 */
7861 plock(ip)
7862 {
7863     int *ip;
7864     register *rp;
7865     rp = ip;
7866     while(rp->flag&ILOCK) {
7867         rp->flag |= IWANT;
7868         sleep(rp, PIPB);
7869     }
7870     rp->flag = |IWANT;
7871     rp->flag = |ILOCK;
7872 }
7873 /* ----- */
7874 /* Unlock a pipe.
7875 * If WANT bit is on,
7876 * wakeup.
7877 * This routine is also used
7878 * to unlock inodes in general.
7879 */
7880 prele(ip)
7881 {
7882     int *ip;
7883     register *rp;
7884     rp = ip;
7885     if(rp->flag & ~ILOCK;
7886     {
7887         if(rp->flag&IWANT) {
7888             rp->flag = &~IWANT;
7889             wakeup(rp);
7890         }
7891     }
7892 }
7893 /* ----- */
7894 }

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976



# 5

**Character Oriented  
Special Files**



```

7900 /*
7901 * A clist structure is the head
7902 * of a linked list queue of characters.
7903 * The characters are stored in 4-word
7904 * blocks containing a link and 6 characters.
7905 * The routines getc and putc (m45.s or m40.s)
7906 * manipulate these structures.
7907 */
7908 struct clist
7909 {
7910     int     c_cc;          /* character count */
7911     int     c_cf;          /* pointer to first block */
7912     int     c_cl;          /* pointer to last block */
7913 };
7914 /* ----- */
7915
7916 /*
7917 * A tty structure is needed for
7918 * each UNIX character device that
7919 * is used for normal terminal IO.
7920 * The routines in tty.c handle the
7921 * common code associated with
7922 * these structures.
7923 * The definition and device dependent
7924 * code is in each driver (kl.c dc.c dh.c)
7925 */
7926 struct tty
7927 {
7928     struct clist t_rawq; /* input chars right off device */
7929     struct clist t_canq; /* input chars after erase and kill */
7930     struct clist t_outq; /* output list to device */
7931     int     t_flags; /* mode, settable by stty call */
7932     int     *t_addr; /* device address (register or
7933                     startup fcn) */
7934     char     t_delct; /* number of delimiters in raw q */
7935     char     t_col; /* printing column of device */
7936     char     t_erase; /* erase character */
7937     char     t_kill; /* kill character */
7938     char     t_state; /* internal state, not visible
7939                     externally */
7940     char     t_char; /* character temporary */
7941     int     t_speeds; /* output+input line speed */
7942     int     t_dev; /* device name */
7943 };
7944 /* ----- */
7945
7946
7947 char partab[]; /* ASCII table: parity, character class */
7948
7949

```

```

7950
7951 #define     TTIPRI 10
7952 #define     TTOPRI 20
7953
7954 #define     CERASE '#' /* default special characters */
7955 #define     CEOT 004
7956 #define     CKILL '@'
7957 #define     CQUIT 034 /* FS, cntl shift L */
7958 #define     CINTR 0177 /* DEL */
7959
7960 /* limits */
7961 #define     TTHIWAT 50
7962 #define     TTLLOWAT 30
7963 #define     TTYHOG 256
7964
7965 /* modes */
7966 #define     HUPCL 01
7967 #define     XTABS 02
7968 #define     LCASE 04
7969 #define     ECHO 010
7970 #define     CRMOD 020
7971 #define     RAW 040
7972 #define     ODDP 0100
7973 #define     EVENP 0200
7974 #define     NLDELAY 001400
7975 #define     TBDELAY 006000
7976 #define     CRDELAY 030000
7977 #define     VTDELAY 040000
7978
7979 /* Hardware bits */
7980 #define     DONE 0200
7981 #define     IENABLE 0100
7982
7983 /* Internal state bits */
7984 #define     TIMEOUT 01 /* Delay timeout in progress */
7985 #define     WOPEN 02 /* Waiting for open to
7986                     complete */
7987 #define     ISOPEN 04 /* Device is open */
7988 #define     SSTART 010 /* Has special start routine
7989                     at addr */
7990 #define     CARR_ON 020 /* Software copy of
7991                     carrier-present */
7992 #define     BUSY 040 /* Output in progress */
7993 #define     ASLEEP 0100 /* Wakeup when output done */
7994
7995
7996
7997
7998
7999

```

```

8000 #
8001 /* KL/DL-11 driver */
8002 #include "../param.h"
8003 #include "../conf.h"
8004 #include "../user.h"
8005 #include "../tty.h"
8006 #include "../proc.h"
8007 /* base address */
8008 #define KLADDR 0177560 /* console */
8009 #define KLBASE 0176500 /* kl and dl11-a */
8010 #define DBASE 0175610 /* dl-e */
8011 #define NK111 1
8012 #define NDL11 0
8013 #define DSRDY 02
8014 #define RDRNB 01
8015 struct tly_kl11[NK111+NDL11];
8016 struct klrregs {
8017     int klrcsr;
8018     int klrbuf;
8019     int klrcsr;
8020     int klrbuf;
8021 }
8022 /* ----- */
8023 #define klopen(dev, flag)
8024 { register char * addr;
8025   register struct tly *tp;
8026   if(dev.d_minor >= (NK111+NDL11)) {
8027     u_n_error = ENXIO;
8028     return;
8029   }
8030   tp = &kl11[dev.d_minor];
8031   if (u_n_procp->p_ttyp == 0) {
8032     u_n_procp->p_ttyp = tp;
8033     tp->t_dev = dev;
8034   }
8035   /* set up minor 0 to address KLADDR
8036      * set up minor 1 thru NK111-1 to address from KLBASE
8037      * set up minor NK111 on to address from DBASE
8038   */
8039   addr = KLADDR + 8*dev.d_minor;
8040   if(dev.d_minor)
8041     addr = + KLBASE-KLADDR-8;
8042   if(dev.d_minor >= NK111)
8043     addr = + DBASE-KLBASE-8*NK111+8;
8044   tp->t_addr = addr;
8045   if ((tp->t_state&ISOBEN) == 0) {
8046     tp->t_state = ISOBEN|CAR ON;
8047     tp->t_flags = XTABS|LCASE|ECHO|CRMOD;
8048     tp->t_erase = Cerase;
8049     tp->t_kill = CKILL;
8050 }

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lyons, 1976

```

8050 }
8051     addr->klcsr = | IENABLE|DSRDY|RDRNB;
8052     addr->klcsr = | IENABLE;
8053 }
8054 /* ----- */
8055 #define kclose(dev)
8056 { register struct tly *tp;
8057   tp = &kl11[dev.d_minor];
8058   wtuhshty(tp);
8059   tp->t_state = 0;
8060 }
8061 /* ----- */
8062 #define kread(dev)
8063 { tthead(&kl11[dev.d_minor]);
8064 }
8065 /* ----- */
8066 #define kwrite(dev)
8067 { ttwrite(&kl11[dev.d_minor]);
8068 }
8069 /* ----- */
8070 #define kxint(dev)
8071 { register struct tly *tp;
8072   tp = &kl11[dev.d_minor];
8073   tstart(tp);
8074   if (tp->t_outq.c_cc == 0 || tp->t_outq.c_cc == TTIOWAIT)
8075     wakeup(&tp->t_outq);
8076 }
8077 /* ----- */
8078 #define krint(dev)
8079 { register int c, *addr;
8080   register struct tly *tp;
8081   tp = &kl11[dev.d_minor];
8082   addr = tp->t_addr;
8083   c = addr->klrbuf;
8084   addr->klcsr = | RDRNB;
8085   if ((c&0177) == 0)
8086     addr->klrbuf = c;
8087   ttyinput(c, tp);
8088 }
8089 /* ----- */
8090 #define klgty(dev, v)
8091 int *v;
8092 { register struct tly *tp;
8093   tp = &kl11[dev.d_minor];
8094   tysty(tp, v);
8095 }
8096 /* ----- */

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lyons, 1976

```

8100 # /* general TTY subroutines */
8101
8102 #include "../param.h"
8103 #include "../system.h"
8104 #include "../user.h"
8105 #include "../tty.h"
8106 #include "../proc.h"
8107 #include "../inode.h"
8108 #include "../file.h"
8109 #include "../reg.h"
8110 #include "../conf.h"
8111
8112 /* Input mapping table-- if an entry is non-zero, when the
8113 * corresponding character is typed preceded by "\" the
8114 * escape sequence is replaced by the table value.
8115 * Mostly used for upper-case only terminals.
8116 */
8117 char      maptab[]
8118 {
8119     000,000,000,000,000,004,000,000,000,
8120     000,000,000,000,000,000,000,000,000,
8121     000,000,000,000,000,000,000,000,000,
8122     000,000,000,000,000,000,000,000,000,
8123     000,'|',000,'#',000,000,000,000,000,
8124     '{','}',000,000,000,000,000,000,000,
8125     000,000,000,000,000,000,000,000,000,
8126     000,000,000,000,000,000,000,000,000,
8127     '@',000,000,000,000,000,000,000,000,
8128     000,000,000,000,000,000,000,000,000,
8129     000,000,000,000,000,000,000,000,000,
8130     000,000,000,000,000,000,000,000,000,
8131     000,'A','B','C','D','E','F','G',
8132     'H','I','J','K','L','M','N','O',
8133     'P','Q','R','S','T','U','V','W',
8134     'X','Y','Z',000,000,000,000,000,000,
8135 };
8136 /* ----- */
8137 /* The actual structure of a clist block manipulated by
8138 *getc and putc (mch.s)
8139 */
8140 struct cblock {
8141     struct cblock *c_next;
8142     char info[6];
8143 };
8144 /* ----- */
8145 /* The character lists-- space for 6*NCLIST characters */
8146 struct cblock cfree[NCLIST];
8147
8148 /* List head for unused character blocks. */
8149 struct cblock *cfreelist;

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

8150 /* structure of device registers for KL, DL, and DC
8151 * interfaces-- more particularly, those for which the
8152 * SSTART bit is off and can be treated by general routines
8153 * (that is, not DH).
8154 */
8155 struct {
8156     int ttrcsr;
8157     int ttrbuf;
8158     int tttcsr;
8159     int tttbuf;
8160 };
8161 /* ----- */
8162 /* The routine implementing the gtty system call.
8163 * Just call lower level routine and pass back values.
8164 */
8165 gtty()
8166 {
8167     int v[3];
8168     register *up, *vp;
8169
8170     vp = v;
8171     sgTTY(vp);
8172     if (u.u_error)
8173         return;
8174     up = u.u_arg[0];
8175     suword(up, *vp++);
8176     suword(++up, *vp++);
8177     suword(++up, *vp++);
8178 }
8179 /* ----- */
8180 /* The routine implementing the stty system call.
8181 * Read in values and call lower level.
8182 */
8183 stty()
8184 {
8185     register int *up;
8186
8187     up = u.u_arg[0];
8188     u.u_arg[0] = fuword(up);
8189     u.u_arg[1] = fuword(++up);
8190     u.u_arg[2] = fuword(++up);
8191     sgTTY(0);
8192 }
8193 /* ----- */
8194 /* Stuff common to stty and gtty.
8195 * Check legality and switch out to individual
8196 * device routine.
8197 * v is 0 for stty; the parameters are taken from u.u_arg[].
8198 * c is non-zero for gtty and is the place in which the
8199 * device routines place their information.

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

8200 /*
8201 sgtty(v)
8202 int *v;
8203 {
8204     register struct file *fp;
8205     register struct inode *ip;
8206     if ((fp = getf(u_n_ar0[R0])) == NULL)
8207         return;
8208     ip = fp->f_inode;
8209     if ((ip->i_mode&IFMT) != IFCHR) {
8210         u_n_error = ENOTTY;
8211         return;
8212     }
8213     (*cdevsw[ip->i_addr[0].d_major].d_sgty)(ip->i_addr[0], v);
8214     /* ----- */
8215     /* Wait for output to drain, then flush output waiting. */
8216     wflush(ip);
8217     struct tty *atp;
8218     {
8219         register struct tty *tp;
8220         register struct tty *tpt;
8221         tp = atp;
8222         sp15();
8223         while (tp->t_outq.c_cc) {
8224             tp->t_state = ASLEEP;
8225             sleep(&tp->t_outq, TTOPRI);
8226         }
8227         flush(tp);
8228         sp10();
8229     }
8230     /* ----- */
8231     /* Initialize list by freeing all character blocks, & count
8232     * number of character devices. (Once-only routine)
8233     */
8234     cinit();
8235     register int cpi;
8236     register struct cblock *cp;
8237     register struct cdevsw *cdp;
8238     cpi = cfree;
8239     for (cp = cdevsw + 07; cp <= &tree[NCLIST-1]; cp++) {
8240         for (cpi = cdevsw + 07; cpi <= &tree[NCLIST-1]; cpi++) {
8241             cp->c_next = cfree;
8242             cfree = cp;
8243         }
8244         cpi = 0;
8245         for (cdp = cdevsw; cdp->d_open; cdp++)
8246             cdp++;
8247         nchrdev = cpi;
8248     }
8249     /* ----- */

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

8250 /* Flush all TTY queues
8251 */
8252 flush(atp)
8253 struct tty *atp;
8254 {
8255     register struct tty *tp;
8256     register int sps;
8257     return;
8258     while (getc(&tp->t_canq) >= 0);
8259     while (getc(&tp->t_outq) >= 0);
8260     wakeup(&tp->t_rawq);
8261     wakeup(&tp->t_outq);
8262     sps = ps->integ;
8263     sp15();
8264     while (getc(&tp->t_rawq) >= 0);
8265     tp->t_delc = 0;
8266     ps->integ = sps;
8267 }
8268 /* ----- */
8269 /* transfer raw input list to canonical list,
8270 * doing erase-kill processing and handling escapes.
8271 * It waits until a full line has been typed in cooked mode,
8272 * or until any character has been typed in raw mode.
8273 */
8274 canon(atp)
8275 struct tty *atp;
8276 {
8277     register char *bp;
8278     char *bpl;
8279     register struct tty *tp;
8280     register int c;
8281     tp = atp;
8282     sp15();
8283     while (tp->t_delc==0) {
8284         if ((tp->t_state&CAR_ON)==0)
8285             return(0);
8286         sleep(&tp->t_rawq, TTOPRI);
8287     }
8288     sp10();
8289     loop:
8290     bp = &canonb[2];
8291     while ((c=getc(&tp->t_rawq)) >= 0) {
8292         if (c==0377) {
8293             tp->t_delc--;
8294             break;
8295         }
8296         if ((tp->t_flags&RAW)==0) {
8297             if (bp[-1]!='\\')
8298                 if (c==tp->t_erase)
8299                     if (bp[-1]!='\\')

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

8300             if (bp > &canonb[2])
8301                 bp--;
8302             continue;
8303         }
8304         if (c==tp->t_kill)
8305             goto loop;
8306         if (c==CEOT)
8307             continue;
8308     } else
8309     if (maptab[c] && (maptab[c]==c || (tp->t_flags&LCASE))) {
8310         if (bp[-2] != '\\')
8311             c = maptab[c];
8312         bp--;
8313     }
8314     }
8315     *bp++ = c;
8316     if (bp>=canonb+CANBSIZ)
8317         break;
8318 }
8319 bp1 = bp;
8320 bp = &canonb[2];
8321 c = &tp->t_cang;
8322 while (bp<bp1)
8323     putc(*bp++, c);
8324 return(1);
8325 }
8326 /* ----- */
8327 /* Place a character on raw TTY input queue, putting in
8328 * delimiters and waking up top half as needed.
8329 * Also echo if required.
8330 * The arguments are the character and the appropriate
8331 * tty structure.
8332 */
8333 ttyinput(ac, atp)
8334 struct tty *atp;
8335 {
8336     register int t_flags, c;
8337     register struct tty *tp;
8338
8339     tp = atp;
8340     c = ac;
8341     t_flags = tp->t_flags;
8342     if ((c == 0177) == '\r' && t_flags&CRMOD)
8343         c = '\n';
8344     if ((t_flags&RAW)==0 && (c==CQUIT || c==CINTR)) {
8345         signal(tp, c==CINTR? SIGINT:SIGQUIT);
8346         flushtty(tp);
8347         return;
8348     }
8349     if (tp->t_rawq.c_cc>=TTYHOG) {

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

8350         flushtty(tp);
8351         return;
8352     }
8353     if (t_flags&LCASE && c>='A' && c<='Z')
8354         c += 'a'-'A';
8355     putc(c, &tp->t_rawq);
8356     if (t_flags&RAW || c=='\n' || c==004) {
8357         wakeup(&tp->t_rawq);
8358         if (putc(0377, &tp->t_rawq)==0)
8359             tp->t_delct++;
8360     }
8361     if (t_flags&ECHO) {
8362         ttyoutput(c, tp);
8363         ttstart(tp);
8364     }
8365 }
8366 /* ----- */
8367 /* put character on TTY output queue, adding delays,
8368 * expanding tabs, and handling the CR/NL bit.
8369 * It is called both from the top half for output, and from
8370 * interrupt level for echoing.
8371 * The arguments are the character and the tty structure.
8372 */
8373 ttyoutput(ac, tp)
8374 struct tty *tp;
8375 {
8376     register int c;
8377     register struct tty *rtp;
8378     register char *colp;
8379     int ctype;
8380
8381     rtp = tp;
8382     c = ac&0177;
8383     /* Ignore EOT in normal mode to avoid hanging up
8384      * certain terminals.
8385     */
8386     if (c==004 && (rtp->t_flags&RAW)==0)
8387         return;
8388     /* Turn tabs to spaces as required
8389     */
8390     if (c=='\t' && rtp->t_flags&XTABS) {
8391         do
8392             ttyoutput(' ', rtp);
8393         while (rtp->t_col&07);
8394         return;
8395     }
8396     /* for upper-case-only terminals,
8397      * generate escapes.
8398     */
8399     if (rtp->t_flags&LCASE) {

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

8400 colp = "{|~|~|~|~|";
8401 while(*colp++)
8402     if(c == *colp++)
8403         ttyoutput("\\', rtp);
8404     c = colp[-2];
8405     break;
8406     }
8407     if ('a'<=c && c<='z',
8408         c += 'A' - 'a',
8409     }
8410     /* turn <n1> to <cr><lf> if desired.
8411     */
8412     if (c=='\n' && rtp->_c_flags&CRMOD)
8413         ttyoutput('\r', rtp);
8414     if (putc(c, &rtp->t_outq))
8415         return;
8416     /* Calculate delays.
8417     * The numbers here represent clock ticks
8418     * and are not necessarily optimal for all terminals.
8419     * The delays are indicated by characters above 0200,
8420     * thus (unfortunately) restricting the transmission
8421     * path to 7 bits.
8422     */
8423     colp = &rtp->t_col;
8424     ctype = partab[c];
8425     c = 0;
8426     switch(ctype&07) {
8427     /* ordinary */
8428     case 0:
8429         (*colp)++;
8430         /* non-printing */
8431     case 1:
8432         break;
8433     /* backspace */
8434     case 2:
8435         if (*colp)
8436             (*colp)--;
8437         break;
8438     /* newline */
8439     case 3:
8440         ctype = (rtp->t_flags >> 8) & 03;
8441         if(ctype == 1) { /* tly 37 */
8442             if (*colp)
8443                 c = max((*colp)>>4) + 3, 6);
8444         } else
8445             if(ctype == 2) { /* vt05 */
8446                 c = 6;
8447             }
8448         *colp = 0;
8449         break;

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

8450 /* tab */
8451     case 4:
8452         ctype = (rtp->t_flags >> 10) & 03;
8453         if(ctype == 1) { /* tly 37 */
8454             c = 1 - (*colp | ~07);
8455             if(c > 5)
8456                 c = 0;
8457         }
8458         *colp |= 07;
8459         (*colp)++;
8460         break;
8461     /* vertical motion */
8462     case 5:
8463         if(rtp->_c_flags & VDELAY) /* tly 37 */
8464             c = 0177;
8465         break;
8466     /* carriage return */
8467     case 6:
8468         ctype = (rtp->t_flags >> 12) & 03;
8469         if(ctype == 1) { /* tn 300 */
8470             c = 5;
8471         } else
8472             if(ctype == 2) { /* tl 700 */
8473                 c = 10;
8474             }
8475         *colp = 0;
8476     }
8477     if(c)
8478         putc(c|0200, &rtp->t_outq);
8479     }
8480     /* ----- */
8481     /* Restart typewriter output following a delay
8482     * timeout.
8483     * The name of the routine is passed to the timeout
8484     * subroutine and it is called during a clock interrupt.
8485     */
8486     tstrrt(amp);
8487     }
8488     register struct tly *tp;
8490     tp = atp;
8491     tp->t_state = &~TIMEOUT;
8492     tstart(tp);
8493     }
8494     /* ----- */
8495     /* Start output on the typewriter. It is used from the top
8496     * half after some characters have been put on the output
8497     * queue, from the interrupt routine to transmit the next
8498     * character, and after a timeout has finished.
8499     * If the START bit is off for the tly the work is done

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976



```

8500 * here, using the protocol of the single-line interfaces
8501 * (kl, dl, dc); otherwise the address word of the tty
8502 * structure is taken to be the name of the device-dependent
8503 * start-up routine.
8504 */
8505 ttstart(atp)
8506 struct tty *atp;
8507 {
8508     register int *addr, c;
8509     register struct tty *tp;
8510     struct { int (*func)(); };
8511
8512     tp = atp;
8513     addr = tp->t_addr;
8514     if (tp->t_state&SSTART) {
8515         (*addr.func)(tp);
8516         return;
8517     }
8518     if ((addr->tttcsr&DONE)==0 || tp->t_state&TIMEOUT)
8519         return;
8520     if ((c=getc(&tp->t_outq)) >= 0) {
8521         if (c<=0177)
8522             addr->tttbuf = c | (partab[c]&0200);
8523         else {
8524             timeout(ttrstrt, tp, c&0177);
8525             tp->t_state |= TIMEOUT;
8526         }
8527     }
8528 }
8529 /* ----- */
8530 /* Called from device's read routine after it has
8531 * calculated the tty-structure given as argument.
8532 * The pc is backed up for the duration of this call.
8533 * In case of a caught interrupt, an RTI will re-execute.
8534 */
8535 ttread(atp)
8536 struct tty *atp;
8537 {
8538     register struct tty *tp;
8539
8540     tp = atp;
8541     if ((tp->t_state&CARR_ON)==0)
8542         return;
8543     if (tp->t_canq.c_cc || canon(tp))
8544         while (tp->t_canq.c_cc && passc(getc(&tp->t_canq))>=0);
8545 }
8546 /* ----- */
8547 /* Called from the device's write routine after it has
8548 * calculated the tty-structure given as argument.
8549 */

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

8550 ttwrite(atp)
8551 struct tty *atp;
8552 {
8553     register struct tty *tp;
8554     register int c;
8555     tp = atp;
8556     if ((tp->t_state&CARR_ON)==0)
8557         return;
8558     while ((c=cpass())>=0) {
8559         spl5();
8560         while (tp->t_outq.c_cc > TTHIWAT) {
8561             ttstart(tp);
8562             tp->t_state |= ASLEEP;
8563             sleep(&tp->t_outq, TTOPRI);
8564         }
8565         spl0();
8566         ttyoutput(c, tp);
8567     }
8568     ttstart(tp);
8569 }
8570 /* ----- */
8571 /* Common code for gtty and stty functions on typewriters.
8572 * If v is non-zero then gtty is being done and information
8573 * is passed back therein;
8574 * if it is zero stty is being done and the input inform-
8575 * ation is in the u_arg array.
8576 */
8577 ttystty(atp, av)
8578 int *atp, *av;
8579 {
8580     register *tp, *v;
8581     tp = atp;
8582     if (v = av) {
8583         *v++ = tp->t_speeds;
8584         v->lobyte = tp->t_erase;
8585         v->hibyte = tp->t_kill;
8586         v[1] = tp->t_flags;
8587         return(1);
8588     }
8589     wflushtty(tp);
8590     v = u.u_arg;
8591     tp->t_speeds = *v++;
8592     tp->t_erase = v->lobyte;
8593     tp->t_kill = v->hibyte;
8594     tp->t_flags = v[1];
8595     return(0);
8596 }
8597 /* ----- */
8598
8599

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

8600 #
8601 /* PC-11 Paper tape reader/punch driver */
8602
8603 #include "../param.h"
8604 #include "../conf.h"
8605 #include "../user.h"
8606
8607 #define PCADDR 0177550
8608
8609 #define CLOSED 0
8610 #define WAITING 1
8611 #define RADING 2
8612 #define EOF 3
8613
8614 #define RDRENB 01
8615 #define IRENABLE 0100
8616 #define DONE 0200
8617 #define BUSY 04000
8618 #define ERROR 0100000
8619
8620 #define PCIPRI 30
8621 #define PCOPRI 40
8622 #define PCOWMAT 50
8623 #define PCOHMAT 100
8624 #define PCIHMAT 250
8625
8626 struct {
8627     int pcrsr;
8628     int pcbuft;
8629     int ppcsr;
8630     int pcbpuf;
8631 };
8632 /* -----
8633 */
8634 struct clist {
8635     int cc;
8636     int cf;
8637     int cl;
8638 };
8639 /* -----
8640 */
8641 struct pcll {
8642     int pcstate;
8643     struct clist pcin;
8644     struct clist pcout;
8645 } pcll;
8646 /* -----
8647 */
8648 pcopen(dev, flag)
8649 {

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lyons, 1976

```

8650 extern lbolt;
8651
8652 if (flag==0) {
8653     if (pcll.pcstate!=CLOSED)
8654         u_n_error = ENXIO;
8655     return;
8656 }
8657 pcll.pcstate = WAITING;
8658 while (pcll.pcstate==WAITING) {
8659     PCADDR->pccsr = IRENABLE|RDRENB;
8660     sleep(&lbolt, PCIPRI);
8661 }
8662 else {
8663     PCADDR->pcpcsr = | IRENABLE;
8664     pcleader();
8665 }
8666 }
8667 /* -----
8668 */
8669 pclose(dev, flag)
8670 {
8671     if (flag==0) {
8672         spi4();
8673         while (getc(&pcll.pcin) >= 0) {
8674             PCADDR->pccsr = 0;
8675             pcll.pcstate = CLOSED;
8676         }
8677     } else
8678         pcleader();
8679 }
8680 /* -----
8681 */
8682 pcread()
8683 {
8684     register int c;
8685     spi4();
8686     do {
8687         while ((c = getc(&pcll.pcin)) < 0) {
8688             if (pcll.pcstate==EOF)
8689                 goto out;
8690             if ((PCADDR->pccsr&(ERROR|BUSY|DONE))!=0)
8691                 PCADDR->pccsr = | IRENABLE|RDRENB;
8692             sleep(&pcll.pcin, PCIPRI);
8693         }
8694     } while (passc(c)>=0);
8695     out:
8696     spi0();
8697 }
8698 /* -----
8699 */

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lyons, 1976

```

8700
8701 pcwrite()
8702 {
8703     register int c;
8704
8705     while ((c=cpass())>=0)
8706         pcoutput(c);
8707 }
8708 /* ----- */
8709
8710 pcstart()
8711 {
8712     register int c;
8713
8714     if (PCADDR->pcpcsr&DONE && (c = getc(&pc11.pcout)) >= 0)
8715         PCADDR->pcpbuf = c;
8716 }
8717 /* ----- */
8718
8719 pprint()
8720 {
8721     if (pc11.pcstate==WAITING) {
8722         if (PCADDR->pcrcsr&ERROR)
8723             return;
8724         pc11.pcstate = READING;
8725     }
8726     if (pc11.pcstate==READING) {
8727         if (PCADDR->pcrcsr&ERROR)
8728             pc11.pcstate = EOF;
8729         else {
8730             putc(PCADDR->pcrbuf, &pc11.pcin);
8731             if (pc11.pcin.cc < PCIHWAT)
8732                 PCADDR->pcrcsr = | IENABLE|RDRENB;
8733         }
8734         wakeup(&pc11.pcin);
8735     }
8736 }
8737 /* ----- */
8738
8739 pcpint()
8740 {
8741
8742     pcstart();
8743     if (pc11.pcout.cc <= PCOLWAT)
8744         wakeup(&pc11.pcout);
8745 }
8746 /* ----- */
8747
8748 pcoutput(c)
8749 {

```

```

8750     if (PCADDR->pcpcsr&ERROR) {
8751         u.u_error = EIO;
8752         return;
8753     }
8754     if (pc11.pcout.cc >= PCOHWAT)
8755         sleep(&pc11.pcout, PCOPRI);
8756     putc(c, &pc11.pcout);
8757     spl4();
8758     pcstart();
8759     spl0();
8760 }
8761 /* ----- */
8762
8763 pcleader()
8764 {
8765     register int i;
8766
8767     i = 100;
8768     do
8769         pcoutput(0);
8770     while (--i);
8771 }
8772 /* ----- */
8773
8774
8775
8776
8777
8778
8779
8780
8781
8782
8783
8784
8785
8786
8787
8788
8789
8790
8791
8792
8793
8794
8795
8796
8797
8798
8799

```

```

8800 #
8801 /*
8802 */
8803 /*
8804 /*
8805 * LP-11 line printer driver
8806 */
8807 #include "../param.h"
8808 #include "../conf.h"
8809 #include "../user.h"
8810 #define LPADDR 0177514
8811 #define LPADDR 0177514
8812 #define LPADDR->lpdr = | IENABLE;
8813 #define LPADDR->lpdr = | IENABLE;
8814 #define LPADDR->lpdr = | IENABLE;
8815 #define LPADDR->lpdr = | IENABLE;
8816 #define LPADDR->lpdr = | IENABLE;
8817 #define LPADDR->lpdr = | IENABLE;
8818 #define LPADDR->lpdr = | IENABLE;
8819 #define LPADDR->lpdr = | IENABLE;
8820 #define LPADDR->lpdr = | IENABLE;
8821 #define LPADDR->lpdr = | IENABLE;
8822 #define LPADDR->lpdr = | IENABLE;
8823 #define LPADDR->lpdr = | IENABLE;
8824 #define LPADDR->lpdr = | IENABLE;
8825 #define LPADDR->lpdr = | IENABLE;
8826 #define LPADDR->lpdr = | IENABLE;
8827 #define LPADDR->lpdr = | IENABLE;
8828 #define LPADDR->lpdr = | IENABLE;
8829 #define LPADDR->lpdr = | IENABLE;
8830 #define LPADDR->lpdr = | IENABLE;
8831 #define LPADDR->lpdr = | IENABLE;
8832 #define LPADDR->lpdr = | IENABLE;
8833 #define LPADDR->lpdr = | IENABLE;
8834 #define LPADDR->lpdr = | IENABLE;
8835 #define LPADDR->lpdr = | IENABLE;
8836 #define LPADDR->lpdr = | IENABLE;
8837 #define LPADDR->lpdr = | IENABLE;
8838 #define LPADDR->lpdr = | IENABLE;
8839 #define LPADDR->lpdr = | IENABLE;
8840 #define LPADDR->lpdr = | IENABLE;
8841 #define LPADDR->lpdr = | IENABLE;
8842 #define LPADDR->lpdr = | IENABLE;
8843 #define LPADDR->lpdr = | IENABLE;
8844 #define LPADDR->lpdr = | IENABLE;
8845 #define LPADDR->lpdr = | IENABLE;
8846 #define LPADDR->lpdr = | IENABLE;
8847 #define LPADDR->lpdr = | IENABLE;
8848 #define LPADDR->lpdr = | IENABLE;
8849 #define LPADDR->lpdr = | IENABLE;

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

8850 #define LPADDR->lpdr = | IENABLE;
8851 #define LPADDR->lpdr = | IENABLE;
8852 #define LPADDR->lpdr = | IENABLE;
8853 #define LPADDR->lpdr = | IENABLE;
8854 #define LPADDR->lpdr = | IENABLE;
8855 #define LPADDR->lpdr = | IENABLE;
8856 #define LPADDR->lpdr = | IENABLE;
8857 #define LPADDR->lpdr = | IENABLE;
8858 #define LPADDR->lpdr = | IENABLE;
8859 #define LPADDR->lpdr = | IENABLE;
8860 #define LPADDR->lpdr = | IENABLE;
8861 #define LPADDR->lpdr = | IENABLE;
8862 #define LPADDR->lpdr = | IENABLE;
8863 #define LPADDR->lpdr = | IENABLE;
8864 #define LPADDR->lpdr = | IENABLE;
8865 #define LPADDR->lpdr = | IENABLE;
8866 #define LPADDR->lpdr = | IENABLE;
8867 #define LPADDR->lpdr = | IENABLE;
8868 #define LPADDR->lpdr = | IENABLE;
8869 #define LPADDR->lpdr = | IENABLE;
8870 #define LPADDR->lpdr = | IENABLE;
8871 #define LPADDR->lpdr = | IENABLE;
8872 #define LPADDR->lpdr = | IENABLE;
8873 #define LPADDR->lpdr = | IENABLE;
8874 #define LPADDR->lpdr = | IENABLE;
8875 #define LPADDR->lpdr = | IENABLE;
8876 #define LPADDR->lpdr = | IENABLE;
8877 #define LPADDR->lpdr = | IENABLE;
8878 #define LPADDR->lpdr = | IENABLE;
8879 #define LPADDR->lpdr = | IENABLE;
8880 #define LPADDR->lpdr = | IENABLE;
8881 #define LPADDR->lpdr = | IENABLE;
8882 #define LPADDR->lpdr = | IENABLE;
8883 #define LPADDR->lpdr = | IENABLE;
8884 #define LPADDR->lpdr = | IENABLE;
8885 #define LPADDR->lpdr = | IENABLE;
8886 #define LPADDR->lpdr = | IENABLE;
8887 #define LPADDR->lpdr = | IENABLE;
8888 #define LPADDR->lpdr = | IENABLE;
8889 #define LPADDR->lpdr = | IENABLE;
8890 #define LPADDR->lpdr = | IENABLE;
8891 #define LPADDR->lpdr = | IENABLE;
8892 #define LPADDR->lpdr = | IENABLE;
8893 #define LPADDR->lpdr = | IENABLE;
8894 #define LPADDR->lpdr = | IENABLE;
8895 #define LPADDR->lpdr = | IENABLE;
8896 #define LPADDR->lpdr = | IENABLE;
8897 #define LPADDR->lpdr = | IENABLE;
8898 #define LPADDR->lpdr = | IENABLE;
8899 #define LPADDR->lpdr = | IENABLE;

```

Reproduced under license from the Western Electric Company, NY  
Copyright, J. Lions, 1976

```

8900
8901     case '|':
8902         c2 = '!';
8903         goto esc;
8904
8905     case '~':
8906         c2 = '^';
8907
8908     esc:
8909         lp canon(c2);
8910         lp11.ccc--;
8911         c1 = '-';
8912     }
8913 }
8914
8915 switch(c1) {
8916
8917     case '\t':
8918         lp11.ccc = (lp11.ccc+8) & ~7;
8919         return;
8920
8921     case FORM:
8922     case '\n':
8923         if((lp11.flag&EJECT) == 0 ||
8924            lp11.mcc!=0 || lp11.mlc!=0) {
8925             lp11.mcc = 0;
8926             lp11.mlc++;
8927             if(lp11.mlc >= EJLINE && lp11.flag&EJECT)
8928                 c1 = FORM;
8929             lpoutput(c1);
8930             if(c1 == FORM)
8931                 lp11.mlc = 0;
8932         }
8933
8934     case '\r':
8935         lp11.ccc = 0;
8936         if(lp11.flag&IND)
8937             lp11.ccc = 8;
8938         return;
8939
8940     case 010:
8941         if(lp11.ccc > 0)
8942             lp11.ccc--;
8943         return;
8944
8945     case ' ':
8946         lp11.ccc++;
8947         return;
8948
8949     default:

```

```

8950         if(lp11.ccc < lp11.mcc) {
8951             lpoutput('\r');
8952             lp11.mcc = 0;
8953         }
8954         if(lp11.ccc < MAXCOL) {
8955             while(lp11.ccc > lp11.mcc) {
8956                 lpoutput(' ');
8957                 lp11.mcc++;
8958             }
8959             lpoutput(c1);
8960             lp11.mcc++;
8961         }
8962         lp11.ccc++;
8963     }
8964 }
8965 /* ----- */
8966
8967 lpstart()
8968 {
8969     register int c;
8970
8971     while (LPADDR->lpsr&DONE && (c = getc(&lp11)) >= 0)
8972         LPADDR->lpbuf = c;
8973 }
8974 /* ----- */
8975
8976 lpint()
8977 {
8978     register int c;
8979
8980     lpstart();
8981     if (lp11.cc == LPLWAT || lp11.cc == 0)
8982         wakeup(&lp11);
8983 }
8984 /* ----- */
8985
8986 lpoutput(c)
8987 {
8988     if (lp11.cc >= LPHWAT)
8989         sleep(&lp11, LPPRI);
8990     putc(c, &lp11);
8991     spl4();
8992     lpstart();
8993     spl0();
8994 }
8995 /* ----- */
8996
8997
8998
8999

```

```

9000 #
9001 /*
9002 */
9003
9004 /*
9005 * Memory special file
9006 * minor device 0 is physical memory
9007 * minor device 1 is kernel memory
9008 * minor device 2 is EOF/RATHOLE
9009 */
9010
9011 #include "../param.h"
9012 #include "../user.h"
9013 #include "../conf.h"
9014 #include "../seg.h"
9015
9016 mmread(dev)
9017 {
9018     register c, bn, on;
9019     int a, d;
9020     if(dev.d_minor == 2)
9021         return;
9022     do {
9023         bn = lshft(u.u_offset, -6);
9024         on = u.u_offset[1] & 077;
9025         a = UISA->r[0];
9026         d = UISD->r[0];
9027         spl7();
9028         UISA->r[0] = bn;
9029         UISD->r[0] = 077406;
9030         if(dev.d_minor == 1)
9031             UISA->r[0] = (ka6-6)->r[(bn>>7)&07]
9032                 + (bn & 0177);
9033         c = fubyte(on);
9034         UISA->r[0] = a;
9035         UISD->r[0] = d;
9036         spl0();
9037     } while(u.u_error==0 && passc(c)>=0);
9038 }
9039 /* ----- */
9040
9041 mmwrite(dev)
9042 {
9043     register c, bn, on;
9044     int a, d;
9045     if(dev.d_minor == 2)
9046         u.u_count = 0;
9047     c = u.u_count;
9048     u.u_count = 0;
9049 }

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976

```

9050 u.u_base += c;
9051 dpadd(u.u_offset, c);
9052 return;
9053 }
9054 {
9055     for(;;)
9056     {
9057         bn = lshft(u.u_offset, -6);
9058         on = u.u_offset[1] & 077;
9059         if ((c=pass())<0 || u.u_error!=0)
9060             break;
9061         a = UISA->r[0] = bn;
9062         d = UISD->r[0] = 077406;
9063         spl7();
9064         UISA->r[0];
9065         UISD->r[0];
9066         if(dev.d_minor == 1)
9067             UISA->r[0] = (ka6-6)->r[(bn>>7)&07]
9068                 + (bn & 0177);
9069         subbyte(on, c);
9070         UISA->r[0] = a;
9071         UISD->r[0] = d;
9072         spl0();
9073     }
9074 }
9075 /* ----- */
9076
9077
9078
9079
9080
9081
9082
9083
9084
9085
9086
9087
9088
9089
9090
9091
9092
9093
9094
9095
9096
9097
9098
9099

```

Reproduced under license from the Western Electric Company, NY  
 Copyright, J. Lions, 1976