UNBOS INSTRUCTIONAL OPERATING SYSTEM SOURCE CODE

by

Craig Steven Bruce

TR92-071 August 1992

Faculty of Computer Science
University of New Brunswick
P.O. Box 4400
Fredericton, N.B. E3B 5A3

```
total 9
drwxrwxrwx   2      560 Aug 31 13:02 devlib
drwxrwxrwx   2      416 Aug 30 23:55 h
drwxrwxrwx   2     1184 Aug 31 13:15 kernel
drwxrwxrwx   2      320 Aug 31 12:52 strlib
drwxrwxrwx   2      480 Aug 31 12:55 syslib
drwxrwxrwx   6      272 Aug 31 13:05 user
drwxrwxrwx   2      240 Aug 31 12:56 userboot

./devlib:
total 116
-r--rw-rw-   1     1553 Aug 31 12:04 Makefile
-r--rw-rw-   1     2193 Aug 31 12:04 arthandle.c
-r--rw-rw-   1     4758 Aug 31 12:04 artinit.c
-r--rw-rw-   1     2710 Aug 31 12:04 artint.s
-r--rw-rw-   1     6941 Aug 31 12:05 bootserver.c
-r--rw-rw-   1     2526 Aug 31 12:05 devglobals.c
-rw-rw----   1    21346 Aug 31 13:02 devlib.a
-r--rw-rw-   1     7627 Aug 31 12:05 diskops.c
-r--rw-rw-   1     8958 Aug 31 12:06 diskserver.c
-r--rw-rw-   1     4648 Aug 31 12:06 iocc.c
-r--rw-rw-   1     1527 Aug 31 12:06 ioccint.s
-r--rw-rw-   1     1410 Aug 31 12:06 setvec.c
-r--rw-rw-   1     1740 Aug 31 12:07 ttyinit.c
-r--rw-rw-   1     6188 Aug 31 12:07 ttyrcv.c
-r--rw-rw-   1     5682 Aug 31 12:08 ttyserver.c
-r--rw-rw-   1     2212 Aug 31 12:08 ttyxmit.c

./estdio:
total 68
-rw-rw-rw-   1    59450 Nov 15  1991 stdio.a
-rw-rw-rw-   1     7205 Aug 30  1991 stdio.h

./h:
total 60
-r--rw-rw-   1     4140 Aug 30 23:51 art.h
-r--rw-rw-   1     2898 Aug 30 23:51 as.h
-r--rw-rw-   1     1582 Aug 30 23:52 debug.h
-r--rw-rw-   1      773 Aug 30 23:52 diskserver.h
-r--rw-rw-   1     2186 Aug 30 23:52 errcode.h
-r--rw-rw-   1     1398 Aug 30 23:52 fileserver.h
-r--rw-rw-   1     3347 Aug 30 23:52 icu.h
-r--rw-rw-   1     2749 Aug 30 23:52 iocc.h
-r--rw-rw-   1      874 Aug 30 23:52 kernel.h
-r--rw-rw-   1     2705 Aug 30 23:53 kernserver.h
-r--rw-rw-   1     2211 Aug 30 23:53 memmap.h
-r--rw-rw-   1     1191 Aug 30 23:53 pageframe.h
-r--rw-rw-   1      908 Aug 30 23:53 port.h
-r--rw-rw-   1     1167 Aug 30 23:53 queue.h
-r--rw-rw-   1     1319 Aug 30 23:53 sem.h
-r--rw-rw-   1     1863 Aug 30 23:54 server.h
-r--rw-rw-   1     7675 Aug 30 23:54 stdio.h
-r--rw-rw-   1     1043 Aug 30 23:54 syslib.h
-r--rw-rw-   1     3016 Aug 30 23:54 trap.h
-r--rw-rw-   1     1593 Aug 30 23:54 tty.h
-r--rw-rw-   1     1666 Aug 30 23:54 vm.h
```

```
-r--rw-rw-   1    3180 Aug 30 23:54 xb1401.h

./kernel:
total 374
-rw-rw----   1   33316 Aug 31 13:12 Kernel.im
-r--rw-rw-   1    4002 Aug 30 22:30 Makefile
-r--rw-rw-   1     196 Aug 30 22:30 Xlink
-r--rw-rw-   1    6623 Aug 30 22:31 addrspace.c
-r--rw-rw-   1    2431 Aug 30 22:31 diskswap.c
-r--rw-rw-   1    5445 Aug 30 22:31 freemem.c
-r--rw-rw-   1    9037 Aug 30 22:31 getmem.c
-r--rw-rw-   1    3671 Aug 30 22:32 getty.c
-r--rw-rw-   1    2544 Aug 30 22:32 globals.c
-r--rw-rw-   1    6757 Aug 30 22:32 icu.c
-r--rw-rw-   1    7379 Aug 30 22:32 initkern.c
-r--rw-rw-   1    6863 Aug 30 22:33 initserver.c
-r--rw-rw-   1    4474 Aug 30 22:33 initsys.c
-r--rw-rw-   1    6204 Aug 30 22:33 kernlib.c
-r--rw-rw-   1    4965 Aug 30 22:34 kernserver.c
-r--rw-rw-   1    5885 Aug 30 22:35 kernservpr.c
-r--rw-rw-   1    3563 Aug 30 22:36 lock.c
-r--rw-rw-   1    6709 Aug 30 22:36 msgbuf.c
-r--rw-rw-   1    6196 Aug 30 22:36 msgpass.c
-r--rw-rw-   1   13760 Aug 30 22:37 pageframe.c
-r--rw-rw-   1    1813 Aug 30 22:37 panic.s
-r--rw-rw-   1    5354 Aug 30 22:37 port.c
-r--rw-rw-   1    4527 Aug 30 22:37 queue.c
-r--rw-rw-   1    4067 Aug 30 22:37 ready.c
-r--rw-rw-   1    3157 Aug 30 22:38 realmem.s
-r--rw-rw-   1    2875 Aug 30 22:38 regwork.s
-r--rw-rw-   1    6076 Aug 30 22:38 saveregs.s
-r--rw-rw-   1    4624 Aug 30 22:38 sem.c
-r--rw-rw-   1    2740 Aug 30 22:38 startup.s
-r--rw-rw-   1    3897 Aug 30 22:39 svchandler.c
-r--rw-rw-   1    8201 Aug 30 22:39 thread.c
-r--rw-rw-   1    3006 Aug 30 22:39 trap.c
-r--rw-rw-   1    2825 Aug 30 22:39 trapcatch.s

./strlib:
total 24
-r--rw-rw-   1     448 Aug 31 12:09 Makefile
-r--rw-rw-   1     354 Aug 31 12:09 atou.c
-r--rw-rw-   1     236 Aug 31 12:09 charfuncs.c
-r--rw-rw-   1    1071 Aug 31 12:09 convert.c
-r--rw-rw-   1    1333 Aug 31 12:10 doprint.c
-r--rw-rw-   1    1026 Aug 31 12:10 memcpy.c
-r--rw-rw-   1     680 Aug 31 12:10 memfuncs.c
-r--rw-rw-   1    1564 Aug 31 12:10 strfuncs.c
-rw-rw----   1    4836 Aug 31 12:52 strlib.a

./syslib:
total 58
-r--rw-rw-   1    1135 Aug 31 12:13 Makefile
-r--rw-rw-   1     388 Aug 31 12:13 abort.c
-r--rw-rw-   1    2942 Aug 31 12:13 callsoc.c
-r--rw-rw-   1    2764 Aug 31 12:14 callsrw.c
```

```
-r--rw-rw-   1    1202 Aug 31 12:14 callutil.c
-r--rw-rw-   1    2669 Aug 31 12:14 fexec.c
-r--rw-rw-   1    3509 Aug 31 12:14 malloc.c
-r--rw-rw-   1    1341 Aug 31 12:14 msglib.c
-r--rw-rw-   1    1121 Aug 31 12:14 statmsg.c
-r--rw-rw-   1    2122 Aug 31 12:14 svc.s
-rw-rw----   1   12104 Aug 31 12:55 syslib.a
-r--rw-rw-   1     921 Aug 31 12:14 threadlib.c
-r--rw-rw-   1    1646 Aug 31 12:14 threadtab.c
-r--rw-rw-   1     284 Aug 31 12:15 userpanic.c

./user:
total 66
-r--rw-rw-   1     424 Aug 31 12:19 Makefile
-r--rw-rw-   1     194 Aug 31 12:19 Userlink
-rw-rw----   1   23448 Aug 31 13:05 bootdisk.im
drwxrwxrwx   2     192 Aug 31 12:21 cat
drwxrwxrwx   2      64 Aug 31 12:21 data
drwxrwxrwx   2     160 Aug 31 13:05 hex
-rwxrwxrwx   1   18321 Jun 22 00:58 mkboot
-r--rw-rw-   1    1949 Aug 31 12:19 mkboot.c
-rwxrwxrwx   1   11208 Jul 22 18:16 mkexec
-r--rw-rw-   1    2100 Aug 31 12:19 mkexec.c
drwxrwxrwx   2     288 Aug 31 13:04 shell

./user/cat:
total 2
-r--rw-rw-   1     582 Aug 31 12:21 Makefile
-r--rw-rw-   1     778 Aug 31 12:21 cat.c

./user/data:
total 1
-rw-rw-rw-   1      53 Jul 22 17:42 Userprog.unbos

./user/hex:
total 45
-r--rw-rw-   1     582 Aug 31 12:22 Makefile
-rw-rw----   1    9276 Aug 31 13:05 Userprog.im
-rw-rw----   1    9296 Aug 31 13:05 Userprog.unbos
-r--rw-rw-   1    1103 Aug 31 12:22 hex.c

./user/shell:
total 82
-r--rw-rw-   1     874 Aug 31 12:23 Makefile
-rw-rw----   1   14076 Aug 31 13:04 Userprog.im
-rw-rw----   1   14096 Aug 31 13:04 Userprog.unbos
-r--rw-rw-   1    1310 Aug 31 12:23 builtin.c
-r--rw-rw-   1    3047 Aug 31 12:23 disk.c
-r--rw-rw-   1     519 Aug 31 12:23 external.c
-r--rw-rw-   1    2943 Aug 31 12:23 shell.c
-r--rw-rw-   1     159 Aug 31 12:24 shell.h

./userboot:
total 22
-r--rw-rw-   1     731 Aug 31 12:02 Makefile
-r--rw-rw-   1     122 Aug 31 12:03 bootentry.s
```

```
-r--rw-rw-   1    3853 Aug 31 12:03 loadprog.c
-rw-rw----   1    1428 Aug 31 12:56 useerboot.im
-r--rw-rw-   1    1291 Aug 31 12:03 userstart.c
-r--rw-rw-   1     193 Aug 31 12:03 xlink
```

```
/*-------------------------------------------------------------------------*\
 *  HEADER:  art.h    1.4    92/08/30    23:51:37
 *
 *  Definitions for Signetics 2681 Dual UART.  This file is mostly ripped
 *  off from Xinu-3216.
\*-------------------------------------------------------------------------*/

/*  Register definitions for both channels */
#define       ARTMRA  0       /* mode register channel A              read/write*/
#define       ARTSRA  2       /* Status register ch. A                read    */
#define       ARTCSRA 2       /* Clock select register ch. A          write   */
#define       ARTCRA  4       /* Command register ch. A               write   */
#define       ARTRHRA 6       /* Receiver holding register ch A.      read    */
#define ARTTHRA       6       /* Transmitter holding register ch A.   write   */
#define ARTIPCR 8     /* Input port change register           read    */
#define       ARTACR  8       /* Auxilliary control register          write   */
#define ARTISR        10      /* Interrupt status register            read    */
#define       ARTIMR  10      /* Interrupt mask register              write   */
#define       ARTCTU  12      /* Counter/timer upper half             read    */
#define ARTCTUR       12      /* Counter/timer upper register         write   */
#define ARTCTL        14      /* Counter/timer lower half             read    */
#define ARTCTLR       14      /* Counter/timer lower register         write   */
#define ARTMRB        16      /* Mode register channel B              read/write*/
#define       ARTSRB  18      /* Status register ch. B                read    */
#define ARTCSRB       18      /* Clock select register ch. B          write   */
#define       ARTCRB  20      /* Command register ch. B               write   */
#define       ARTRHRB 22      /* Receiver holding register ch B.      read    */
#define ARTTHRB       22      /* Transmitter holding register ch B.   write   */
#define       ARTIP   26      /* Input port                           read    */
#define       ARTOPCR 26      /* output port configuration            write   */
#define ARTSTCC 28    /* Start counter command address        read !  */
#define       ARTSOPBC 28     /* Set output port bits command         write   */
#define ARTSPCC       30      /* Stop counter command                 read !  */
#define       ARTROPBC 30     /* Reset output port bits command       write   */

/* a channel-generic definition for the UART */
#define       ARTMR   0       /* Mode register                        read/write*/
#define       ARTSR   2       /* Status register                      read    */
#define ARTCSR        2       /* Clock select register                write   */
#define       ARTCR   4       /* Command register                     write   */
#define       ARTRHR  6       /* Receiver holding register            read    */
#define       ARTTHR  6       /* Transmitter holdng register          write   */

/* define the Channels */
#define ARTCHANA      0       /* Channel A */
#define ARTCHANB      1       /* Channel B */

/* definitions for command register fields */
#define       ARTTXDISABLE    0x8     /* Disable transmitter  */
#define       ARTTXENABLE     0x4     /* Enable  Transmitter  */
#define       ARTRXDISABLE    0x2     /* Disable receiver     */
#define ARTRXENABLE   0x1     /* Enable receiver      */
#define ARTRESETMR    0x10    /* Reset the mode register pointer      */
#define ARTRRESET     0x20    /* Reset receiver & flush FIFO  */
#define ARTTRESET     0x30    /* Reset transmitter    */
#define ARTCLEARERR   0x40    /* Reset the error flags*/
```

```
#define ARTRESETBRK   0x50    /* Reset the break change interrupt     */
#define ARTXMTBRK     0x60    /* Start transmitting break     */
#define ARTSTOPBRK    0x70    /* Stop Transmitting break      */

#define       ARTERRMASK      0x80    /* Mask for status register     */
#define ARTRXRDY      0x01    /* Mask to check if reciever ready      */
#define ARTTXRDY      0x04    /* Mask to check if transmitter ready   */

#define ARTTIENABLE   0x01    /* Mask to enable xmit interrupt generation */
#define ARTRIENABLE   0x02    /* Mask to enable rec interrupt generation */
#define ARTTIENABLEA  0x01
#define ARTRIENABLEA  0x02
#define ARTTIENABLEB  0x10
#define ARTRIENABLEB  0x20

/* "hardware"settings these are as close to the rom monitor values as possible*/
#define ARTMODE1      0x93
#define ARTMODE2      0x17
#define ARTOPCONF     0xf0
#define ARTAUXVAL     0x30
#define ARTSOPBVAL    0x0f
#define ARTSPEED      0xbb
#define ARTIMRCLEAR   0x00

#define ARTCHMASK     0377    /*mask to force only 7 bit ascii characters*/

typedef struct {
      void            (*receiveVectorA)();
      void            (*receiveVectorB)();
      void            (*transmitVectorA)();
      void            (*transmitVectorB)();
      Ulong            channelParmA;
      Ulong            channelParmB;
      unsigned char   *ioaddr;
      unsigned char    imrValue;
} ArtDesc;

/*-------------------------------------------------------------------------------*\
 *  END OF HEADER:  art.h
\*-------------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  HEADER:  as.h    1.9    92/08/30    23:51:46
 *
 *  Definitions for address spaces and threads.
\*-----------------------------------------------------------------------------*/

#define  AS_MAGIC_NUM_ADDR_SPACE      0x50534441      /* "ADSP" */
#define  AS_MAGIC_NUM_THREAD          0x44524854      /* "THRD" */

#define  AS_THREAD_STATE_CURRENT      10
#define  AS_THREAD_STATE_READY                11
#define  AS_THREAD_STATE_WAIT_PORT    20
#define  AS_THREAD_STATE_WAIT_SVC     21
#define  AS_THREAD_STATE_WAIT_SEM     30
#define  AS_THREAD_STATE_SUSPENDED    40
#define  AS_THREAD_STATE_BUSY         50

#define  AS_PRIORITY_HIGHEST          0
#define  AS_PRIORITY_SVC_SERVER               2
#define  AS_PRIORITY_RECYCLER_SERVER  3
#define  AS_PRIORITY_DEVICE_SERVER    5
#define  AS_PRIORITY_KERNEL_SERVER    8
#define  AS_PRIORITY_BOOT_THREAD      9
#define  AS_PRIORITY_USER_LEVEL_SERVER        10
#define  AS_PRIORITY_USER_PROCESS     20
#define  AS_PRIORITY_NULL_THREAD      0xFFFFFFFE

#define  AS_THREAD_REG_COUNT          13
#define  AS_USER_TRAP_COUNT           4
#define  AS_REG_R0                    0
#define  AS_REG_GPR_COUNT             8
#define  AS_REG_PC                    8
#define  AS_REG_PSR                   9
#define  AS_REG_SP                    10
#define  AS_REG_FP                    11
#define  AS_REG_MOD                   12
#define  AS_REG_USER_PSR              0xB00
#define  AS_REG_KERNEL_PSR            0xA00
#define  AS_REG_BOOT_PSR              0x200

#define  AS_NULL_STACK_LENGTH           450
#define  AS_BOOT_STACK_LENGTH           2000
#define  AS_THREAD_NAME_LENGTH                33
#define  AS_DEFAULT_STACK_LENGTH      4096

/*== Required forward definitions ============================================*/

typedef struct StructAddrSpace AddrSpace;
typedef struct StructThread Thread;

/*== ADDRESS SPACE ===========================================================*/

struct StructAddrSpace {
      AddrSpace      *nextAddrSpace;
      AddrSpace      *prevAddrSpace;
      Ulong           priority;
```

```
        Ulong           addrSpaceMagicNumber;
        Void           *pageTable;
        Thread         *ownedThreadList;
        Ulong           addrSpaceCapabilityCheck;
        Boolean         swappable;
        Boolean         swappedIn;
        Ulong           memoryPageCount;
        Ulong           firstFreePage;
        Void           *lockingSem;
        Void           *ownedPortList; /* actually of type Queue* */
        Ulong           notificationPortMachineID;
        Ulong           notificationPortPortDesc;
        Ulong           notificationPortPortCheck;
};

/*== THREAD ==========================================================================*/

struct StructThread {
        Thread         *nextThread;
        Thread         *prevThread;
        Ulong           priority;
        Ulong           threadMagicNumber;
        Ulong           state;
        Ulong           threadCapabilityCheck;
        AddrSpace      *ownerAddrSpace;
        Void           *userPtr;
        Ulong           reg [AS_THREAD_REG_COUNT];
        Thread         *nextThreadInAddrSpace;
        Void           *portWaitingOn;
        Void           *semWaitingOn;
        Ulong           semMaxPriority;
        char            name[AS_THREAD_NAME_LENGTH];
};

/*-------------------------------------------------------------------------------------*\
 *  END OF HEADER:  as.h
\*-------------------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------*\
 *  HEADER:  debug.h    1.3    92/08/30    23:51:56
 *
 *  Definitions for debugging.  Uncomment the definitions for the debugging
 *  status information you wish to view on the console as the system runs.
\*-------------------------------------------------------------------------*/

/*=========================================================================*\
 *  Kernel.
\*=========================================================================*/

#define  DEBUG
#define  DEBUG_VERIFY /*for certain non-printing consistency checking*/
/* #define  DEBUG_VERIFY_MUCH */
#define  DEBUG_BANNER
/* #define  DEBUG_INITPFT */
/* #define  DEBUG_GETMEM */
/* #define  DEBUG_IOCC */
/* #define  DEBUG_ADDR_SPACE */
/* #define  DEBUG_PORT */
/* #define  DEBUG_MSG_BUFFER */
/* #define  DEBUG_THREAD */

/*=========================================================================*\
 *  Device and kernel servers.
\*=========================================================================*/

#define  DEBUG_KERN_SERVER
#define  DEBUG_BOOT_DISK_SERVER
/* #define  DEBUG_BOOT_DISK_SERVER_MUCH */
/* #define  DEBUG_MSG_RECYCLER_SERVER */
/* #define  DEBUG_TTY */
#define  DEBUG_DISK_BLOCK_SERVER
/* #define  DEBUG_DISK_BLOCK_SERVER_MUCH */
/* #define  DEBUG_DISK_FILE_SERVER */
/* #define  DEBUG_PREFIX_SERVER */

/*-------------------------------------------------------------------------*\
 *  END OF HEADER:  debug.h
\*-------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  HEADER:  diskserver.h    1.2    92/08/30    23:52:04
 *
 *  Disk server request codes.  Disk servers would probably be better off
 *  without the allocate, free, and sync calls.
\*-----------------------------------------------------------------------------*/

#define  DS_READ_BLOCKS              4501
#define  DS_WRITE_BLOCKS      4502
#define  DS_ALLOCATE_BLOCKS   4503
#define  DS_FREE_BLOCKS              4504
#define  DS_SYNC_DISK         4505
#define  DS_FORMAT_DISK              4506
#define  DS_MOUNT_DISK               4507
#define  DS_UNMOUNT_DISK      4508

/*-----------------------------------------------------------------------------*\
 *  END OF HEADER:  diskserver.h
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------*\
 *  HEADER:  errcode.h    1.5    92/08/30    23:52:11
 *
 *  Error code definitions.  These error codes are returned by servers and
 *  kernel routines.  The POSIX 1003.1 draft error codes are included here
 *  too, but are mostly unused.
\*-----------------------------------------------------------------------*/

typedef int Errcode;

#define  ERR_OK                              0
#define  ERR_UNSPECIFIED              1
#define  ERR_INVALID_OBJECT           2

/* address space errors */

#define  ERR_ABORT_REQUEST            1001

/* thread errors */

#define  ERR_TH_RESUME                       2001
#define  ERR_TH_SUSPEND                      2002

/* message passing errors */

#define  ERR_MSG_BAD_PORT_DESC               3001
#define  ERR_MSG_BAD_PORT_CHECK              3002
#define  ERR_MSG_PORT_NOT_OWNED              3003
#define  ERR_MSG_NO_MESSAGE           3004
#define  ERR_MSG_RECEIVE_WAIT         3005

/* server errors */

#define  ERR_SERVER_UNKNOWN_OPCODE    4001
#define  ERR_SERVER_BAD_CAPABILITY    4002
#define  ERR_SERVER_TOO_MANY_OPEN_FILES      4003
#define  ERR_SERVER_FILE_NOT_FOUND    4004

/* disk server errors */

#define  ERR_DISK_NOT_MOUNTED         5001
#define  ERR_DISK_MOUNTED             5002
#define  ERR_DISK_FULL                       5003
#define  ERR_DISK_BLOCK_ALREADY_FREE  5004

/* POSIX errno codes */

#define  E2BIG                8001
#define  EACCES               8002
#define  EAGAIN               8003
#define  EBADF                8004    /* bad file number */
#define  EBUSY                8005
#define  ECHILD               8006
#define  EDEADLK      8007
#define  EEXIST               8008
#define  EFAULT               8009
```

```
#define  EFBIG                8010
#define  EINTR                8011
#define  EINVAL               8012
#define  EIO          8013
#define  EISDIR               8014
#define  EMFILE               8015    /* too many open files in a process */
#define  EMLINK               8016
#define  ENAMETOOLONG 8017
#define  ENFILE               8018
#define  ENODEV               8019
#define  ENOENT               8020
#define  ENOEXEC      8021
#define  ENOMEM               8022
#define  ENOSPC               8023
#define  ENOTDIR      8024
#define  ENOTTY               8025
#define  ENXIO                8026
#define  EPERM                8027
#define  EPIPE                8028
#define  EROFS                8029
#define  ESPIPE               8030
#define  ESRCH                8031
#define  EXDEV                8032

/*-------------------------------------------------------------------------------*\
 *  END OF HEADER:  errcode.h
\*-------------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  HEADER:  fileserver.h    1.5    92/08/30    23:52:18
 *
 *  Definitions for device servers (except disk block servers).  The POSIX
 *  "fcntl.h" open codes are included also.
\*-----------------------------------------------------------------------------*/

/* file server operation codes */

#define  FS_OPEN               2001
#define  FS_CLOSE              2002
#define  FS_READ               2003
#define  FS_WRITE              2004
#define  FS_DUP                      2005
#define  FS_LSEEK              2006
#define  FS_STAT               2007
#define  FS_FSTAT              2008
#define  FS_IOCTL              2009
#define  FS_LINK               2010
#define  FS_UNLINK             2011
#define  FS_MKNOD              2012
#define  FS_OPENDIR            2013
#define  FS_CLOSEDIR           2014
#define  FS_READDIR            2015
#define  FS_GET_SEARCH_CAP     2016
#define  FS_GET_DEVICE_TYPE    2017

/* file server device type codes */

#define  FS_DEVICE_TYPE_TTY    3001
#define  FS_DEVICE_TYPE_DISK   3002
#define  FS_DEVICE_TYPE_CHAR   3003
#define  FS_DEVICE_TYPE_PIPE   3004

/* fcntl open codes */

#define  O_RDONLY      0
#define  O_WRONLY      1
#define  O_RDWR              2
#define  O_APPEND      0x04
#define  O_CREAT       0x08
#define  O_EXCL              0x10
#define  O_NONBLOCK    0x20
#define  O_TRUNC       0x40

typedef unsigned long int off_t;

/*-----------------------------------------------------------------------------*\
 *  END OF HEADER:  fileserver.h
\*-----------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------*\
 *  HEADER:  icu.h    1.7    92/08/30    23:52:26
 *
 *  Definitions for the ICU.  This file is ripped off from Xinu-3216.
\*-------------------------------------------------------------------------*/

/* icu.h -          lsv  dec 86 */
/*                  -- modified may 87 for inclusion with xinu   */
/*                  -- modified jun 91 for inclusion with UNBOS  */

/* icu counter start values */

#define  ICU_LCSV_L     0xd0          /*   1ms = 1,474    (0x000005c2) */
#define  ICU_LCSV_H     0x3f          /* 100ms = 147,400 (0x00023fd0) */
#define  ICU_HCSV_L     0x02          /* 5 sec =7,370,000(0x008f8af0) */
#define  ICU_HCSV_H     0x00

/* icu device register constants */

#define  ICU_FREEZE     0x82          /* mctl register for freezing   */
#define  ICU_THAW       0x02          /*  and thawing                 */
#define  ICU_START      0xf8          /* cctl register for starting   */
#define  ICU_STOP       0xf0          /*  and stopping                */

#define  ICU_ART1H      0x0004        /* ICU interrupt register bit   */
#define  ICU_ART2H      0x0010        /*  positions                   */
#define  ICU_CLOCK      0x0040
#define  ICU_ART1L      0x0400
#define  ICU_ART2L      0x1000
#define  ICU_SCSI       0x2000
#define  ICU_PRT        0x4000

/* icu device register layout for nsc 3202 interrupt controller  */
typedef struct {
      Ulong   hvct: 8,        /* hardware vector              */
              byte0: 8,
              svct: 8,        /* software vector              */
              byte3: 8;
      Ulong   eltg_l: 8,      /* edge level triggering        */
              byte5: 8,
              eltg_h: 8,
              byte7: 8;
      Ulong   tpl_l: 8,       /* triggering polarity          */
              byte9: 8,
              tpl_h: 8,
              byte11: 8;
      Ulong   ipnd_l: 8,      /* interrupts pending           */
              byte13: 8,
              ipnd_h: 8,
              byte15: 8;
      Ulong   isrv_l: 8,      /* interrupts in service        */
              byte17: 8,
              isrv_h: 8,
              byte19: 8;
      Ulong   imsk_l: 8,      /* interrupt mask               */
              byte21: 8,
```

```
            imsk_h: 8,
            byte23: 8;
    Ulong   csrc_l: 8,      /* cascaded source              */
            byte25: 8,
            csrc_h: 8,
            byte27: 8;
    Ulong   fprt_l: 8,      /* first priority               */
            byte29: 8,
            fprt_h: 8,
            byte31: 8;
    Ulong   mctl: 8,        /* mode control                 */
            byte33: 8,
            ocasn: 8,       /* output clock assignment      */
            byte35: 8;
    Ulong   ciptr: 8,       /* counter interrupt pointer    */
            byte37: 8,
            pdat: 8,        /* port data ( 8 bit bus mode)  */
            byte39: 8;
    Ulong   ips: 8,         /* interrupt/port select        */
            byte41: 8,
            pdir: 8,        /* port direction               */
            byte43: 8;
    Ulong   cctl: 8,        /* counter control              */
            byte45: 8,
            cictl: 8,       /* counter interrupt control    */
            byte47: 8;
    Ulong   lcsv_l: 8,      /* lower counter starting value */
            byte49: 8,
            lcsv_h: 8,
            byte51: 8;
    Ulong   hcsv_l: 8,      /* high counter start value     */
            byte53: 8,
            hcsv_h: 8,
            byte55: 8;
    Ulong   lccv_l: 8,      /* lower counter current value  */
            byte57: 8,
            lccv_h: 8,
            byte59: 8;
    Ulong   hccv_l: 8,      /* high counter current value    */
            byte61: 8,
            hccv_h: 8,
            byte63: 8;
} Icu;

/*-------------------------------------------------------------------------------*\
 *  END OF HEADER:  icu.h
\*-------------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------*\
 *  HEADER:  iocc.h    1.3    92/08/30    23:52:37
 *
 *  Definitions for the ICM-3216 SCSI I/O Channel Controller.  This file
 *  is mostly ripped off from Xinu-3216.
\*-------------------------------------------------------------------------*/

/* I/O Control Block structure */

typedef struct {                /* controller command descriptor block */
      char    ioccode;          /* always 0x0 */
      char    iocdest;          /* target controller & subchan */
      char    iocscsi[12];      /* scsi control block */
      char    ioctarstat;       /* status code from target cont. */
      char    iocerrstat;       /* status code from chan controller */
      char   *iocdata;          /* pointer to data to be transferred */
      char   *ioctpta;          /* transfer pointer table address */
      Ulong   ioclimit;         /* maximum number of bytes to transfer*/
      char   *ioclink;          /* pointer to next scsicdb if linked */
} IoccCommand;

/* Masks for channel controller status register */

#define IOCC_SRBSY    0x80    /* channel controller is busy */
#define IOCC_SRINT    0x10    /* channel controller has interrupted CPU */
#define IOCC_SRRST    0x20    /* Channel controller has done SCSI bus reset*/
#define IOCC_SRCHMASK 0x07    /* subchannel mask */

/* Commands to channel controller */

#define       IOCC_WCTPA      0       /* write command table pointer address */
#define       IOCC_IA         1       /* Interrupt acknowledge */
#define       IOCC_RSTIOC     2       /* Reset I/O controller */
#define IOCC_RSTSCSI  3       /* Reset SCSI bus */
#define IOCC_DIAG     4       /* Execute diagnostic firmware */
#define IOCC_RSTA     5       /* SCSI bus reset acknowledge */
#define IOCC_SIO      0x10    /* start I/O on subchannel n */
                              /* subchan is specified by adding n to IOCSIO */
#define IOCC_AIO      0x20    /* abort I/O on subchannel n */
              /* subchannel is specified by adding n to IOCSIO or IOCAIO*/

/* Channel controller error codes  (in iocerrstat) */

#define       IOCC_OK         0       /* No error occurred */
#define       IOCC_ABORT      1       /* I/O operation aborted by abort command */
#define       IOCC_TIME       2       /* Target select timed out */
#define IOCC_BIG      3       /* Attempt to transfer more than IOCLIMIT */
#define IOCC_BADC     4       /* Bad Channel command */
#define IOCC_NOBUF    5       /* Buffer space too small for transfer table*/
#define IOCC_DISC     6       /* Unexpected target controller disconnect */

/* Command pointer table */

#define IOCC_NSC      8       /* number of subchannels & size of ioccpt */

/* Constants for delay loops for timeouts */
```

```
#define       IOCC_BT1        50000   /* Approx. 100ms for controller busy tests */
#define IOCC_BT2      5000000         /* Approx. 10s for SCSI bus reset */
#define IOCC_IOT      1000000         /* Approx. 2s for disk io */

/*-----------------------------------------------------------------------------*\
 *  END OF HEADER:  iocc.h
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  HEADER:  kernel.h    1.10    92/08/30    23:52:45
 *
 *  Global type definitions and constants used by all kernel modules.
\*-----------------------------------------------------------------------------*/

typedef unsigned long int Ulong;
typedef unsigned long int Boolean;
typedef char Void;                      /* should only be used as Void* */
typedef unsigned char Uchar;
typedef unsigned long int CPint;        /* character pointer -> int */

#define  NULL          0
#define  FALSE                 0
#define  TRUE          1
#define  VERSION       "ICM-3216 1.00.04 (1992/08/31)"
#define  KERN_MACHINE_ID       1
#define  LOCAL                 static
#define  KERN_UNMAGIC_NUMBER   0

/*-----------------------------------------------------------------------------*\
 *  END OF HEADER:  kernel.h
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  HEADER:  kernserver.h    1.6    92/08/30    23:52:58
 *
 *  Definitions for the kernel server and for the system call library.
\*-----------------------------------------------------------------------------*/

/* kernel server operation codes */

#define  KS_CREATE_ADDR_SPACE          10000
#define  KS_REMOVE_ADDR_SPACE          10001
#define  KS_CREATE_MEMORY              10100
#define  KS_REMOVE_MEMORY              10101
#define  KS_CREATE_PORT                      10200
#define  KS_REMOVE_PORT                      10201
#define  KS_CREATE_MSG_BUFFER          10300
#define  KS_CREATE_THREAD              10400
#define  KS_REMOVE_THREAD              10401
#define  KS_SUSPEND_THREAD             10402
#define  KS_RESUME_THREAD              10403
#define  KS_SET_USER_TRAP              10404

#define  KS_NOTIFY_MSG                       10500

/* system call library addresses */

#define  KS_ADDR_ADDR_SPACE_MSG              0xFE0000  /* a.s. creation msg addr */
#define  KS_ADDR_SYS_CALLS             0xFF0000
#define  KS_ADDR_INFO_PAGE             0xFFFC00  /* kernel info page addr */

/* constants */

#define  KS_AMSG_PTR          ((CreateAddrSpaceMsg *) KS_ADDR_ADDR_SPACE_MSG)
#define  KS_CURRENT_THREAD    (* (Ulong *) KS_ADDR_INFO_PAGE)
#define  KS_MAX_THREADS                      10
#define  KS_MAX_FILES                  20
#define  KS_SMALL_MSG_LENGTH           512
#define  KS_FILE_DATA_LENGTH           1024
#define  KS_FILE_DATA_OFFSET           KS_SMALL_MSG_LENGTH
#define  KS_FILE_BUFFER_LENGTH              (KS_FILE_DATA_LENGTH + KS_FILE_DATA_OFFSET)
#define  KS_BOOT_MSG_BUFFER_LENGTH     16384
#define  KS_INIT_MSG_BUFFER_LENGTH     512
#define  KS_EXEC_PROG_MAGIC_NUM              666

/* address space creation message structures */

typedef struct {
      Boolean         used;
      Ulong           threadID;
      Capability      threadCap;
      MsgBuffer      *threadMsgBuffer;
      Port            threadPort;
      Ulong           threadErrno;
} ThreadEntry;

typedef struct {
      Boolean         used;
```

```
      MsgBuffer       *fileMsgBuffer;
      Capability       fileCap;
} FileEntry;

typedef struct {
      Port             destPort;
      Ulong            physicalLength;
      Ulong            msgStartOffset;
      Ulong            msgLength;
      Capability       addrSpaceCap; /* set by OS */
      Ulong            operationCode;
      Port             replyPort;
      Ulong            requestNumber;
      Ulong            errorCode;
      Port             notificationPort;
      Ulong            initThreadStackLength;
      char            *initThreadStackPtr;
      Ulong            requestedPriority;
      ThreadEntry      threadtab [KS_MAX_THREADS];  /* [0]=initThread */

      /* user-relevant fields */
      FileEntry        filetab [KS_MAX_FILES]; /*[0]=stdin,1=stdout,2=stderr */
      Capability       rootSearchCap;
      Capability       currentSearchCap;
      char            *programFilename;
      Ulong            argc;
      char            *argv[1];        /* really "*argv[argc]" */
      /* strings go here */
} CreateAddrSpaceMsg;

/*-------------------------------------------------------------------------------*\
 *  END OF HEADER:  kernserver.h
\*-------------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  HEADER:  memmap.h    1.11    92/08/30    23:53:09
 *
 *  Address of kernel and user address space objects.
\*-----------------------------------------------------------------------------*/

/* kernel address space */

#define  ADDR_PTBO_TABLE              0x8000          /*   1,024 bytes */
#define  ADDR_MOD_TABLE                       0x8400          /*     512 bytes */
#define  ADDR_DISPATCH_TABLE          0x8600          /*     128 bytes */
#define  ADDR_TRAP_STACK              0x8680          /*   1,920 bytes */
#define  ADDR_TRAP_STACK_END          0x8DFC
#define  ADDR_KERNEL_ADDR_SPACE               0x8E00          /*      ~60 bytes */
#define  ADDR_ADDR_SPACE_LIST         0x8E40          /*      ~20 bytes */
#define  ADDR_KERNEL_AS_PORT_QUEUE    0x8E60          /*      ~20 bytes */
#define  ADDR_SEM_KERNEL_AS           0x8E80          /*      ~16 bytes */
#define  ADDR_SEM_KERNEL_AS_QUEUE     0x8EA0          /*      ~20 bytes */
#define  ADDR_SEM_KMALLOC             0x8EC0          /*      ~16 bytes */
#define  ADDR_SEM_KMALLOC_QUEUE               0x8EE0          /*      ~20 bytes */
#define  ADDR_NULL_THREAD             0x8F00          /*     ~172 bytes */
#define  ADDR_SEM_PAGEFRAME           0x8FC0          /*      ~16 bytes */
#define  ADDR_SEM_PAGEFRAME_QUEUE     0x8FE0          /*      ~20 bytes */
#define  ADDR_TEMP_WORK                       0x8FFC
#define  ADDR_UNBOS                   0x9000          /* variable length */
#define  ADDR_BOOTSTRAP_CODE          0x300000        /* up to 64K */
#define  ADDR_BOOT_DISK_IMAGE         0x310000        /* up to 960K */
#define  ADDR_END_RAM_PLUS1           0x400000
#define  ADDR_IO_SPACE                        0xA00000        /*     512 bytes */
#define  ADDR_CONSOLE_DATA_REG                0xA00056
#define  ADDR_CONSOLE_STAT_REG                0xA00052
#define  ADDR_LAST_THREAD_RESTORED    0xFFFC00        /* shared page */
#define  ADDR_ICU_SPACE                       0xFFFE00        /*     512 bytes */
#define  ADDR_END_VM_PLUS1            0x1000000

/* user address space */

#define  ADDR_USER_ZERO_PAGE          0x000000        /* 512 bytes */
#define  ADDR_USER_MOD_TABLE          0x000200        /* 512 bytes */
#define  ADDR_USER_BOOTSTRAP_CODE     0xFF0000
#define  ADDR_USER_BOOTSTRAP_ENTRY    0xFF0004
#define  ADDR_USER_KERN_INFO_PAGE     0xFFFC00
#define  ADDR_USER_CURRENT_THREAD     0xFFFC00
#define  ADDR_USER_THREAD_PTR         0xFFFC04

/*-----------------------------------------------------------------------------*\
 *  END OF HEADER:  memmap.h
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  HEADER:  pageframe.h    1.5    92/08/30    23:53:20
 *
 *  Page frame table definitions.
\*-----------------------------------------------------------------------------*/

#define  PFT_TYPE_FREE                          0
#define  PFT_TYPE_REGULAR               1
#define  PFT_TYPE_POINTER_TABLE                 2
#define  PFT_TYPE_PAGE_TABLE            3
#define  PFT_STATUS_IN_CLEAN            0
#define  PFT_STATUS_IN_DIRTY            1
#define  PFT_STATUS_RESERVED1           2
#define  PFT_STATUS_RESERVED2           3
#define  PFT_POINTER_NULL               0
#define  PFT_WSDELTA_START              0
#define  PFT_DISK_BLOCK_NULL            0
#define  PFT_VIRT_PAGE_NULL             0xFFFF

typedef struct {
      Ulong             nextPage : 13,
                        prevPage : 13,
                        status   : 2,
                        type     : 2,
                        busy     : 1,
                        pagable  : 1;
      Ulong             virtualPage : 16,
                        wsDelta : 16;
      AddrSpace        *addrSpace;
      Ulong             diskBlock;
} PageFrameTableEntry;

#define  PFT_MASTER_FREE_FRAME          0       /* for free list */
#define  PFT_MASTER_LRU_FRAME 1         /* for LRU list */
#define  PFT_FIRST_USABLE_FRAME         2

/*-----------------------------------------------------------------------------*\
 *  END OF HEADER:  pageframe.h
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  HEADER:  port.h    1.5    92/08/30    23:53:30
 *
 *  Port descriptor definitions.
\*-----------------------------------------------------------------------------*/

#define  PORT_MAGIC_NUMBER    0x54524F50      /* "PORT" */

typedef struct StructPortDesc PortDesc;       /* required forward definition */

struct StructPortDesc {
      PortDesc       *nextPortDesc;
      PortDesc       *prevPortDesc;
      Ulong           priority;
      Ulong           portMagicNumber;
      AddrSpace      *ownerAddrSpace;
      Ulong           portCheck;
      Semaphore      *lockingSem;
      Queue          *waitingThreadList;

      AddrSpace      *storeAddrSpace;
      Ulong           waitingMsgCount;
      Ulong           firstMsgSlot;
      Ulong           lastMsgSlot;
};

/*-----------------------------------------------------------------------------*\
 *  END OF HEADER:  port.h
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  HEADER:  queue.h    1.6    92/08/30    23:53:39
 *
 *  Queue package definitions.
\*-----------------------------------------------------------------------------*/

#define QUEUE_LOWEST_PRIORITY 0xFFFFFFFF
#define QUEUE_MAGIC_NUMBER    0x55455551      /* "QUEU" */

/* data structures */

typedef struct StructQueueEntry QueueEntry;
typedef struct StructQueue Queue;

struct StructQueueEntry {
      QueueEntry     *next;
      QueueEntry     *prev;
      Ulong           priority;
};

struct StructQueue {
      QueueEntry     *head;
      QueueEntry     *tail;
      Ulong           lowpriority;
      Ulong           count;
      Ulong           queueMagicNumber;
};

/* function prototypes */

extern Queue         *QueueCreate();
extern void           QueueRemove();
extern void           QueueInit();
extern void           QueueInsert();
extern void           QueueUnlink();
extern Ulong          QueueCount();
extern Ulong          QueueHeadPriority();
extern QueueEntry      *QueueGetHead();
extern QueueEntry      *QueueHead();

/*-----------------------------------------------------------------------------*\
 *  END OF HEADER:  queue.h
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  HEADER:  sem.h    1.7    92/08/30    23:53:47
 *
 *  Semaphore package definitions.
\*-----------------------------------------------------------------------------*/

#define SEM_MAGIC_NUMBER      0x414D4553      /* "SEMA" */

/* priority values - the priority scheme allows you to only lock a semaphore */
/* of greater priority than the ones you already have.  This eliminates the */
/* possibility of deadlock in the kernel */

#define SEM_PRIORITY_PAGE_FRAME_TABLE 0x00000020
#define SEM_PRIORITY_MALLOC           0x00000010
#define SEM_PRIORITY_ADDR_SPACE             0x00000008
#define SEM_PRIORITY_ADDR_SPACE_LIST  0x00000004
#define SEM_PRIORITY_PORT             0x00000002
#define SEM_PRIORITY_PORT_LIST              0x00000001
#define SEM_PRIORITY_NULL             0x00000000

/* data structure */

typedef struct {
      Ulong   semMagicNumber;
      Ulong   count;
      Ulong   semPriority;
      Queue  *waitingThreadList;
} Semaphore;

/* function prototypes */

extern Semaphore *SemCreate();
extern void       SemInit();
extern void      SemRemove();
extern void      SemWait();
extern void      SemSignal();
extern void      SemRestart();

/*-----------------------------------------------------------------------------*\
 *  END OF HEADER:  sem.h
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  HEADER:  server.h    1.5    92/08/30    23:53:55
 *
 *  Standard definitions used by most servers.
\*-----------------------------------------------------------------------------*/

/* port alias definitions */

#define  SERV_ALIAS_MACHINE            0xFF414B41      /* "AKA_" */
#define  SERV_ALIAS_KERN_SERVER                0
#define  SERV_ALIAS_MSG_RECYCLER       1
#define  SERV_ALIAS_KERN_STATUS                2
#define  SERV_ALIAS_INIT_SERVER                3
#define  SERV_ALIAS_BOOT_DISK          4
#define  SERV_ALIAS_DISK_BLOCK                 5
#define  SERV_ALIAS_CONSOLE            6
#define  SERV_ALIAS_TTY1               7
#define  SERV_ALIAS_TTY2               8
#define  SERV_ALIAS_TTY3               9
#define  SERV_ALIAS_NULL_DEVICE                10
#define  SERV_ALIAS_PREFIX_SERVER      11
#define  SERV_ALIAS_GLOBAL_AS_SERVER   12

#define  SERV_PORT_ALIAS_COUNT                 20
#define  SERV_MAX_MSG_LENGTH           65536
#define  SERV_CAP_RIGHTS_OWNER                 0xFFFFFFFF

/*-- PORT ---------------------------------------------------------------------*/

typedef struct {
      Ulong             machineID;
      Ulong             portDesc;
      Ulong             portCheck;
} Port;


/*-- CAPABILITY ---------------------------------------------------------------*/

typedef struct {
      Port              port;
      Ulong             object;
      Ulong             rights;
      Ulong             check;
} Capability;


/*-- MESSAGE BUFFER -----------------------------------------------------------*/

typedef struct {
      Port              destPort;
      Ulong             physicalLength;
      Ulong             msgStartOffset;
      Ulong             msgLength;
      Capability        capability;
      Ulong             operationCode;
      Port              replyPort;
```

```
    Ulong           requestNumber;
    Ulong           errorCode;
    Ulong           data[1];
} MsgBuffer;

/* message passing function prototypes */

extern int Send();
extern MsgBuffer *Receive();
extern MsgBuffer *ReceiveOpts();

/*-------------------------------------------------------------------------------*\
 *  END OF HEADER:  server.h
\*-------------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  HEADER:  stdio.h    1.3    92/08/30    23:54:03
 *
 *  Definitions used by C.E. Chew's Estdio 2.1 standart I/O library.
\*-----------------------------------------------------------------------------*/

#ifndef _STDIO_H
#define _STDIO_H

/*                    s t d i o
 *
 *            Author: C. E. Chew
 *            Date:   August 1989
 *
 * (C) Copyright C E Chew
 *
 * Feel free to copy, use and distribute this software provided:
 *
 *    1. you do not pretend that you wrote it
 *    2. you leave this copyright notice intact.
 *
 * Definitions and user interface for the stream io package.
 *
 * Patchlevel 2.0
 *
 * Edit History:
 */

/* Site specific definitions */
/*@*/
#ifndef NULL
# define NULL 0
#endif
#define _STDIO_UCHAR_          0
#define _STDIO_VA_LIST_
#define _STDIO_SIZE_T_
#define _STDIO_USIZE_T_                unsigned int
/*=*/

/* Definitions based on ANSI compiler */
#ifdef                  __STDC__
# ifndef       _STDIO_P_
#   define     _STDIO_P_(x)              x
# endif
# ifndef       _STDIO_VA_
#   define     _STDIO_VA_                , ...
# endif
# ifndef       _STDIO_UCHAR_
#   define     _STDIO_UCHAR_             0
# endif
#else
# ifndef       _STDIO_P_
#   define     _STDIO_P_(x)              ()
# endif
# ifndef       _STDIO_VA_
#   define     _STDIO_VA_
```

```
# endif
# ifndef       _STDIO_UCHAR_
#   define     _STDIO_UCHAR_              (0xff)
# endif
#endif

#ifndef                _STDIO_VA_LIST_
#  define      _STDIO_VA_LIST_           void *
#endif

#ifndef                _STDIO_SIZE_T_
#  define      _STDIO_SIZE_T_            unsigned int
#endif

#ifndef                _STDIO_USIZE_T_
#  define      _STDIO_USIZE_T_           unsigned int
#endif

/* ANSI Definitions */
#define BUFSIZ 1024                      /* default buffer size */

#ifndef       NULL
# define NULL          ((void *) 0)      /* null pointer */
#endif

#define EOF            (-1)              /* eof flag */
#define FOPEN_MAX      16                /* minimum guarantee */
#define FILENAME_MAX   127               /* maximum length of file name */

#define SEEK_SET       0                 /* seek from beginning */
#define SEEK_CUR       1                 /* seek from here */
#define SEEK_END       2                 /* seek from end */

#define TMP_MAX                (0xffff)        /* maximum number of temporaries */

#define L_tmpnam       (5 + 8 + 4 + 1 + 1) /* length of temporary file name */

#ifndef _FPOS_T
# define _FPOS_T
  typedef long fpos_t;                         /* stream positioning */
#endif

#ifndef       _SIZE_T
# define _SIZE_T
  typedef _STDIO_SIZE_T_ size_t;       /* sizeof type */
#endif

#define _IOFBF                 000000          /* fully buffered io */
#define _IOREAD                000001          /* opened for reading */
#define _IOWRITE       000002          /* opened for writing */
#define _IONBF                 000004          /* unbuffered */
#define _IOMYBUF       000010          /* allocated buffer */
#define _IOPOOLBUF     000020          /* buffer belongs to pool */
#define _IOEOF                 000040          /* eof encountered */
#define _IOERR                 000100          /* error encountered */
#define _IOSTRING      000200          /* strings */
```

```
#define _IOLBF                  000400          /* line buffered */
#define _IORW           001000          /* opened for reading and writing */
#define _IOAPPEND       002000          /* append mode */
#define _IOINSERT       004000          /* insert into __iop chain */
#define _IOSTDX                 030000          /* standard stream */

#define _IOSTDIN        010000          /* stdin indication */
#define _IOSTDOUT       020000          /* stdout indication */
#define _IOSTDERR       030000          /* stderr indication */

#define _IORETAIN       (_IOSTDX | _IOINSERT)   /* flags to be retained */

/* Implementation Definitions */

typedef char __stdiobuf_t;              /* stdio buffer type */
typedef _STDIO_USIZE_T_ __stdiosize_t;          /* unsigned size_t */

typedef struct __iobuf {
  __stdiobuf_t *__rptr;                         /* pointer into read buffer */
  __stdiobuf_t *__rend;                         /* point at end of read buffer */
  __stdiobuf_t *__wptr;                         /* pointer into write buffer */
  __stdiobuf_t *__wend;                         /* point at end of write buffer */
  __stdiobuf_t *__base;                         /* base of buffer */
  __stdiosize_t __bufsiz;               /* size of buffer */
  short __flag;                                 /* flags */
  char __file;                                  /* channel number */
  __stdiobuf_t __buf;                   /* small buffer */
  int (*__filbuf) _STDIO_P_((struct __iobuf *));       /* fill input buffer */
  int (*__flsbuf) _STDIO_P_((int, struct __iobuf *)); /* flush output buffer */
  int (*__flush) _STDIO_P_((struct __iobuf *));        /* flush buffer */
  struct __iobuf *__next;               /* next in chain */
} FILE;

extern FILE __stdin;                    /* stdin */
extern FILE __stdout;                   /* stdout */
extern FILE __stderr;                   /* stderr */

#define stdin           (&__stdin)
#define stdout                  (&__stdout)
#define stderr                  (&__stderr)

/* ANSI Stdio Requirements */

int   getc              _STDIO_P_((FILE *));
#if   _STDIO_UCHAR_
# define getc(p)        ((p)->__rptr>=(p)->__rend\
                         ?(*(p)->__filbuf)(p)\
                         :(int)(*(p)->__rptr++&_STDIO_UCHAR_))
#else
# define getc(p)        ((p)->__rptr>=(p)->__rend\
                         ?(*(p)->__filbuf)(p)\
                         :(int)((unsigned char)(*(p)->__rptr++)))
#endif

int   getchar           _STDIO_P_((void));
#define getchar()       getc(stdin)
```

```
int   putc              _STDIO_P_((int, FILE *));
#if   _STDIO_UCHAR_
# define putc(x,p)      ((p)->__wptr>=(p)->__wend\
                          ?(*(p)->__flsbuf)((x),(p))\
                         :(int)(*(p)->__wptr++=(x)&_STDIO_UCHAR_))
#else
# define putc(x,p)      ((p)->__wptr>=(p)->__wend\
                          ?(*(p)->__flsbuf)((x),(p))\
                         :(int)((unsigned char)(*(p)->__wptr++=(x))))
#endif

int   putchar           _STDIO_P_((int));
#define       putchar(x)      putc(x,stdout)

int   feof              _STDIO_P_((FILE *));
#define feof(p)                 (((p)->__flag&_IOEOF)!=0)

int   ferror            _STDIO_P_((FILE *));
#define ferror(p)       (((p)->__flag&_IOERR)!=0)

void  clearerr          _STDIO_P_((FILE *));
#define clearerr(p)     ((p)->__flag&=~(_IOEOF|_IOERR))

FILE  *fopen            _STDIO_P_((const char *, const char *));
FILE  *freopen          _STDIO_P_((const char *, const char *, FILE *));
int   fflush            _STDIO_P_((FILE *));
int   fclose            _STDIO_P_((FILE *));

int   fgetpos           _STDIO_P_((FILE *, fpos_t *));
int   fsetpos           _STDIO_P_((FILE *, fpos_t *));
long  ftell             _STDIO_P_((FILE *));
int   fseek             _STDIO_P_((FILE *, long, int));
void  rewind            _STDIO_P_((FILE *));

int   fgetc             _STDIO_P_((FILE *));
int   fputc             _STDIO_P_((int, FILE *));
__stdiosize_t fread     _STDIO_P_((void *, __stdiosize_t,
                                   __stdiosize_t, FILE *));
__stdiosize_t fwrite    _STDIO_P_((void *, __stdiosize_t,
                                   __stdiosize_t, FILE *));

int   getw              _STDIO_P_((FILE *));
int   putw              _STDIO_P_((int, FILE *));
char  *gets             _STDIO_P_((char *));
char  *fgets            _STDIO_P_((char *, int, FILE *));
int   puts              _STDIO_P_((const char *));
int   fputs             _STDIO_P_((const char *, FILE *));

int   ungetc            _STDIO_P_((int, FILE *));

int   printf            _STDIO_P_((const char * _STDIO_VA_));
int   fprintf           _STDIO_P_((FILE *, const char * _STDIO_VA_));
int   sprintf           _STDIO_P_((char *, const char * _STDIO_VA_));
int   vprintf           _STDIO_P_((const char *, _STDIO_VA_LIST_));
int   vfprintf          _STDIO_P_((FILE *, const char *, _STDIO_VA_LIST_));
```

```
int   vsprintf        _STDIO_P_((char *, const char *, _STDIO_VA_LIST_));
int   scanf           _STDIO_P_((const char * _STDIO_VA_));
int   fscanf          _STDIO_P_((FILE *, const char * _STDIO_VA_));
int   sscanf          _STDIO_P_((const char *, const char * _STDIO_VA_));

void  setbuf          _STDIO_P_((FILE *, char *));
int   setvbuf         _STDIO_P_((FILE *, char *, int, __stdiosize_t));

int   rename          _STDIO_P_((const char *, const char *));
int   remove          _STDIO_P_((const char *));

void  perror          _STDIO_P_((const char *));

char *        tmpnam          _STDIO_P_((char *));
FILE *        tmpfile         _STDIO_P_((void));

/* Posix Definitions */
int   unlink          _STDIO_P_((const char *));
#define remove(x)     unlink((x))

#define L_ctermid     9
char *        ctermid         _STDIO_P_((char *s));

#define L_cuserid     9
char *        cuserid         _STDIO_P_((char *s));

FILE  *fdopen         _STDIO_P_((int, const char *));

int   fileno          _STDIO_P_((FILE *));
#define fileno(p)     ((p)->__file)

#undef        _STDIO_P_
#undef        _STDIO_VA_
#undef        _STDIO_VA_LIST_
/*ndef        _STDIO_UCHAR_*/
#undef        _STDIO_SIZE_T_
#undef        _STDIO_USIZE_T_
#endif

/*-------------------------------------------------------------------------------*\
 *  END OF HEADER:  stdio.h
\*-------------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------*\
 *  HEADER:  syslib.h    1.3    92/08/30    23:54:15
 *
 *  Definitions used by some system library functions.  Perhaps some of the
 *  definitions in the "kernserver.h" file should have been put here.
\*-------------------------------------------------------------------------*/

#define  MALLOC_MAGIC_NUMBER  0x424D454D      /* "MEMB" */
#define  MALLOC_GETMEM_SIZE   2048
#define  SYSLIB_RAND_MULT     296212731
#define  SYSLIB_RAND_ADD      87283

/* structure used by the malloc() and free() library calls */
/* The structure is forced to be 16 bytes long for convenience using the */
/* ICM-3216 ROM monitor */

typedef struct StructMemBlock MemBlock;

struct StructMemBlock {
      MemBlock       *nextBlock;
      Ulong           blockLength;
      Ulong           junk1, junk2; /*cheat-make block size 16 bytes*/
};

/*-------------------------------------------------------------------------*\
 *  END OF HEADER:  syslib.h
\*-------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  HEADER:  trap.h    1.6    92/08/30    23:54:24
 *
 *  Definitions for the trap and interrupt mechanisms.
\*-----------------------------------------------------------------------------*/

/*=============================================================================*\
 *  Trap and interrupt vector numbers.
\*=============================================================================*/

#define  TRAP_NON_VECTORED             0
#define  TRAP_NON_MASKABLE             1
#define  TRAP_ABORT                    2
#define  TRAP_FLOATING_POINT           3
#define  TRAP_ILLEGAL_OPERATION                4
#define  TRAP_SVC                      5
#define  TRAP_DIVIDE_BY_ZERO           6
#define  TRAP_FLAG                     7
#define  TRAP_BREAKPOINT               8
#define  TRAP_TRACE                    9
#define  TRAP_UNDEFINED_INSTRUCTION    10
#define  TRAP_ART0_RECEIVE             18
#define  TRAP_ART1_RECEIVE             20
#define  TRAP_CLOCK                    22
#define  TRAP_MINI_BUS                 24
#define  TRAP_ART0_TRANSMIT            26
#define  TRAP_ART1_TRANSMIT            28
#define  TRAP_IOCC                     29
#define  TRAP_PARALLEL_PORT            30
#define  TRAP_TABLE_ENTRY_COUNT                32

/*=============================================================================*\
 *  SuperVisor Call trap register parameters.
\*=============================================================================*/

#define  SVC_REG_OPERATION             0
#define  SVC_OP_SEND                   0x53454E44      /* "SEND" */
#define  SVC_OP_RECEIVE                        0x52454356      /* "RECV" */
#define  SVC_REG_RETURN                        0
#define  SVC_REG_PORT_MACHINE          1
#define  SVC_REG_PORT_DESC             2
#define  SVC_REG_PORT_CHECK            3
#define  SVC_REG_MSG_BUFFER            4
#define  SVC_REG_WAIT_FLAG             5
#define  SVC_REG_MSG_LENGTH            6
#define  SVC_REG_ERRCODE               7

/*=============================================================================*\
 *  Default values for module table fields.
\*=============================================================================*/

#define  TRAP_STATIC_BASE_DEFAULT_VALUE       0
#define  TRAP_LINK_BASE_DEFAULT_VALUE 0
#define  TRAP_RESERVED_DEFAULT_VALUE  0

/*=============================================================================*\
```

```
 *  Module and Dispatch table structures.  The linkBase field of a mod
 *  table entry is used to hold a special parameter for the associated
 *  trap or interrupt so multiple interrupts can be pointed to the same
 *  handler routine which can use the parameter to distinguish which
 *  interrupt occured.
\*=============================================================================*/

typedef struct {
      Ulong   staticBase;
      Ulong   linkBase;       /* used to hold interrupt parameter */
      Void   *programBase;
      Ulong   reserved;
} ModTableEntry;

typedef struct {
      ModTableEntry   entry[TRAP_TABLE_ENTRY_COUNT];
} ModTable;

typedef struct {
      unsigned short  modTablePointer;
      unsigned short  offset;
} DispatchTableEntry;

typedef struct {
      DispatchTableEntry      entry[TRAP_TABLE_ENTRY_COUNT];
} DispatchTable;

/*-----------------------------------------------------------------------------*\
 *  END OF HEADER:  trap.h
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  HEADER:  tty.h    1.4    92/08/30    23:54:33
 *
 *  Definitions used by the TTY server and interrupt routines.
\*-----------------------------------------------------------------------------*/

#define  TTY_IOCTL_SET_MODE   0x1972842d
#define  TTY_MODE_RAW         FALSE
#define  TTY_MODE_COOKED      TRUE
#define  TTY_MODE_SAME                9492334

#define  TTY_INBUF_LENGTH     256
#define  TTY_ECHOBUF_LENGTH   32

/* control characters */

#define  TTY_CH_EOF           '\004'  /* in octal! */
#define  TTY_CH_BELL          '\007'
#define  TTY_CH_LINEFEED      '\012'
#define  TTY_CH_RETURN                '\015'
#define  TTY_CH_FLOW_STOP     '\023'
#define  TTY_CH_FLOW_RESTART  '\021'
#define  TTY_CH_PANIC         '\003'
#define  TTY_CH_BACKSPACE     '\010'
#define  TTY_CH_SPACE         '\040'
#define  TTY_CH_RUBOUT                '\177'
#define  TTY_CH_NONE          0xFF
#define  TTY_CH_MASK          0x7F
#define  TTY_CH_KILL_LINE     '\025'

/* tty control block */

typedef struct {
      ArtDesc         *myart;
      Ulong            artChannel;

      char            *outbufPtr;
      Ulong            outbufCount;
      Boolean          emitLinefeed;
      Semaphore       *outbufEmptySem;

      Ulong            inbufHead;
      Ulong            inbufTail;
      Ulong            inbufCount;
      Semaphore       *inbufReadySem;
      char             inbuf [TTY_INBUF_LENGTH];

      Ulong            echobufHead;
      Ulong            echobufTail;
      Ulong            echobufCount;
      char             echobuf [TTY_ECHOBUF_LENGTH];

      Boolean          cookedMode;
      Boolean          flowStopped;
      Boolean          returnEof;
} TtyDesc;
```

```
/*-----------------------------------------------------------------------------*\
 *  END OF HEADER:  tty.h
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  HEADER:  vm.h    1.9    92/08/30    23:54:41
 *
 *  Definitions used by the memory mapping functions (and many others).
\*-----------------------------------------------------------------------------*/

/* sizes.  The page count definition gives the number of physical page frames */
/* to use.  A maximum of 3M can be used since memory is reserved for the */
/* bootstrapping code and the boot disk */

#define  VM_PAGE_COUNT                          6144    /* 6144 maximum */
#define  VM_PAGE_LENGTH                         512
#define  VM_PAGE_TABLE_ENTRY_COUNT    256
#define  VM_POINTER_TABLE_ENTRY_COUNT 128
#define  VM_ADDR_SPACE_LENGTH         16777216

/* page/pointer table entry masks */

#define  VM_PAGE_MASK                 0x00FFFE00
#define  VM_PAGE_VALID_MASK           0x00000001
#define  VM_SUPER_VRW_MASK            0x00000003
#define  VM_SUPER_VR_MASK             0x00000001
#define  VM_USER_VRW_MASK             0x00000007
#define  VM_USER_VR_MASK              0x00000005
#define  VM_USER_MASK                 0x00000004
#define  VM_MSG_BUFFER_MASK           0x00000080  /*page&pointer T enrties*/
#define  VM_SHARED_PAGE_MASK          0x00000100
#define  VM_NULL_ENTRY                        0xFFFFFFFE
#define  VM_NULL_RESERVED_ENTRY               0xFFFFFFFD
#define  VM_NULL_PAGE_NUM             0xFFFFFFFF
#define  VM_DISK_BLOCK_MASK           0xFFFFFFF8
#define  VM_DISK_BLOCK_RIGHT_SHIFT    3

/* page and pointer table structures */

typedef struct {
     Ulong   entry [VM_PAGE_TABLE_ENTRY_COUNT];
} PageTable;

typedef struct {
     Ulong   entry [VM_POINTER_TABLE_ENTRY_COUNT];
} PointerTable;

/*-----------------------------------------------------------------------------*\
 *  END OF HEADER:  vm.h
\*-----------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------*\
 *  HEADER:  xb1401.h    1.4    92/08/30    23:54:51
 *
 *  Definitions for the Xebec S1401 5.25 inch Floppy Disk Controller.
 *  Mostly ripped off of Xinu-3216.
\*-------------------------------------------------------------------------*/

/* Xebec disk controller control block map to iocscsi.        */
#define       xop     iocscsi[0]              /* command class/opcode         */
#define       xunit   iocscsi[1]              /* unit/high-order address bits */
#define       xmaddr  iocscsi[2]              /* middle-order address bits    */
#define       xladdr  iocscsi[3]              /* low-order address bits       */
#define       xcount  iocscsi[4]              /* block count                  */
#define       xcntl   iocscsi[5]              /* control field                */

/* Xebec controller operation codes */

#define       XOTDR    0              /* Test Drive Ready              */
/* (A floppy is always ready--even if no disk is in the drive.)       */
#define       XORCAL   1              /* Recalibrate                   */
#define       XORSS    3              /* Request Sense Status          */
#define       XOFMTD   4              /* Format Drive                  */
#define       XOFMTT   6              /* Format Track                  */
#define       XOREAD   8              /* Read                          */
#define       XOWRITE 10              /* Write                         */
#define       XOSEEK  11              /* Seek                          */
#define       XOINIT  12              /* Initialize Drive Character.   */
#define       XORAMD  0xe0            /* Perform RAM Diagnostic Test   */
#define       XODRVD  0xe3            /* Perform Drive Diagnostic Test*/
#define       XOCOND  0xe4            /* Perform Controller Diagnostic Test   */

#define       XRETRY  0               /* retry 8 times before reporting error */
#define XNORETRY 0x80         /* no automatic retry after error       */
#define XLOGAD        0               /* disk addresses are logical block numbers*/
#define XPHYSAD       0x40            /* use physical disk address-cyl, trk,sector*/

/* Xebec 1401 -Panasonic JU-455 disk drive parameter values */
/* Xebec 1401 - Panasonic JU-455 Disk Drive Characteristics */

#define       XFCYL   40              /* Number of cylinders */
#define       XFST    0x02            /* Track step time (0 =6ms), high nibble of */
#define XFMT  0x00            /* Motor start time, low byte of motor start*/
                              /* time 0x200 = 512 ms */
#define XFDH  0x52            /* Drive type (5=5.25") No. of heads (2) */
#define XFSIZE        0x02            /* Sector size (2 = 512 bytes) */
#define XFUT  20              /* time to wait before unloading head in .1s */
#define XFSECT        9               /* Number of sectors/track */
#define XFDEN 0xc0            /* Record dens. (c= MFM double dens. all trks*/

/* Status byte masks and codes */
#define XERR  0x02            /* mask for error flag bit */
#define XOK   0               /* no error */

#define XB_BLOCK_LENGTH               512     /* block length in bytes */
#define XB_BLOCK_COUNT                720     /* total number of blocks */
#define XB_BITMAP_LEN         90      /* XB_BL_CT / 8 */
```

```
#define XB_FILLER_LEN          410     /* unused bytes in BAM block */
#define XB_BITMAP_BLOCK                0       /* block number containing bitmap */
#define XB_FIRST_BLOCK                 1       /* file system 'starter' block */
#define XB_MAGIC_NUMBER                0xC723D72E      /* to verify disk */

/* the block allocation map.  This structure must be the size of a disk block */
typedef struct {
      Ulong    magicNumber;
      Ulong    firstFreeBlock;
      Ulong    freeBlockCount;
      Uchar    bitmap [XB_BITMAP_LEN];
      Uchar    filler [XB_FILLER_LEN];
} XebecAllocMap;

/*-------------------------------------------------------------------------------*\
 *  END OF HEADER:  xb1401.h
\*-------------------------------------------------------------------------------*/
```

```
#    Makefile    1.14    92/08/30    22:30:31

TARGET          = 2
COREBLD         = /unbl/cs4605/cmd/corebld
DOWNLOAD= /unbl/cs4605/cmd/dl
TARGCMD         = /unbl/cs4605/cmd/target
LD      = /bin/ld
H       = ../h
U       = ../estdio
CC      = /usr/bin/scc
CFLAGS          = -I../h
AS      = /bin/as
ASFLAGS         =
LIBS  = ../devlib/devlib.a ../syslib/syslib.a ../strlib/strlib.a
XLINK = Xlink
XLINKFILE = Xlink

.c.o:
        $(CC) $(CFLAGS) -c $<
.s.o:
        $(AS) $(ASFLAGS) -o $*.o $<

OBJFILES = startup.o panic.o realmem.o trapcatch.o saveregs.o regwork.o \
        initkern.o trap.o pageframe.o icu.o addrspace.o \
        thread.o getmem.o kernlib.o globals.o ready.o queue.o sem.o \
        svchandler.o port.o msgpass.o msgbuf.o lock.o initserver.o \
        kernserver.o kernservpr.o initsys.o getty.o freemem.o diskswap.o

system: kernel userboot bootdisk go
kernel: Kernel.im
        $(TARGCMD) $(TARGET) "d 9000"
        $(DOWNLOAD) Kernel.im $(TARGET)
        @echo
userboot:
        $(TARGCMD) $(TARGET) "d 300000"
        $(DOWNLOAD) ../userboot/userboot.im $(TARGET)
        @echo
bootdisk:
        $(TARGCMD) $(TARGET) "d 310000"
        $(DOWNLOAD) ../user/bootdisk.im $(TARGET)
        @echo
go:
        $(TARGCMD) $(TARGET) "g 9000"
Kernel.im: $(XLINKFILE) Kernel.oo $(LIBS)
        $(LD) Kernel.oo $(LIBS) $(XLINK) -e _start -o Kernel.co
        $(COREBLD) <Kernel.co >Kernel.im
Kernel.oo: $(OBJFILES)
        $(LD) $(OBJFILES) -r -o Kernel.oo

all: strlib syslib userboot.im devlib user system
strlib:
        -cd ../strlib; make;
syslib:
        -cd ../syslib; make;
userboot.im:
        -cd ../userboot; make;
```

```
devlib:
	-cd ../devlib; make;
user:
	-cd ../user; make;
clean:
	rm -f *.o *.co *.im *.oo

panic.o: panic.s
realmem.o: realmem.s
regwork.o: regwork.s
saveregs.o: saveregs.s
startup.o: startup.s
trapcatch.o: trapcatch.s
addrspace.o: addrspace.c $H/kernel.h $H/as.h $H/vm.h $H/memmap.h $H/queue.h \
	$H/sem.h $H/server.h $H/pageframe.h $H/errcode.h $H/kernserver.h \
	$H/debug.h $H/port.h
diskswap.o: diskswap.c $H/kernel.h $H/as.h $H/vm.h
freemem.o: freemem.c $H/kernel.h $H/vm.h $H/as.h $H/pageframe.h
getmem.o: getmem.c $H/kernel.h $H/vm.h $H/as.h $H/pageframe.h $H/debug.h
getty.o: getty.c $H/kernel.h $H/memmap.h $H/vm.h $H/as.h $H/pageframe.h \
	$H/debug.h $H/server.h
globals.o: globals.c $H/kernel.h $H/vm.h $H/as.h $H/pageframe.h $H/queue.h \
	$H/sem.h $H/server.h
icu.o: icu.c $H/kernel.h $H/memmap.h $H/icu.h $H/debug.h
initkern.o: initkern.c $H/kernel.h $H/memmap.h $H/vm.h $H/as.h $H/queue.h \
	$H/sem.h $H/pageframe.h $H/debug.h
initserver.o: initserver.c $H/kernel.h $H/as.h $H/server.h $H/art.h $H/queue.h \
	$H/sem.h $H/tty.h $H/xb1401.h $H/kernserver.h $H/debug.h
initsys.o: initsys.c $H/kernel.h $H/as.h $H/server.h $H/kernserver.h \
	$H/fileserver.h $H/debug.h
kernlib.o: kernlib.c $H/kernel.h $H/memmap.h $H/vm.h $H/as.h $H/queue.h \
	$H/sem.h $H/art.h $H/syslib.h $H/server.h
kernserver.o: kernserver.c $H/kernel.h $H/as.h $H/server.h $H/kernserver.h \
	$H/errcode.h $H/vm.h $H/debug.h
kernservpr.o: kernservpr.c $H/kernel.h $H/as.h $H/server.h $H/kernserver.h \
	$H/errcode.h $H/debug.h $H/vm.h $H/memmap.h $H/pageframe.h
lock.o: lock.c $H/kernel.h $H/as.h $H/queue.h $H/sem.h $H/port.h $H/vm.h
msgbuf.o: msgbuf.c $H/kernel.h $H/as.h $H/queue.h $H/sem.h $H/port.h \
	$H/server.h $H/vm.h $H/pageframe.h $H/errcode.h
msgpass.o: msgpass.c $H/kernel.h $H/as.h $H/queue.h $H/sem.h $H/port.h \
	$H/server.h $H/errcode.h $H/vm.h $H/trap.h
pageframe.o: pageframe.c $H/kernel.h $H/memmap.h $H/vm.h $H/as.h $H/queue.h \
	$H/sem.h $H/pageframe.h $H/debug.h
port.o: port.c $H/kernel.h $H/as.h $H/queue.h $H/sem.h $H/port.h $H/server.h \
	$H/vm.h $H/pageframe.h $H/errcode.h $H/trap.h
queue.o: queue.c $H/kernel.h $H/queue.h
ready.o: ready.c $H/kernel.h $H/as.h $H/queue.h $H/errcode.h $H/debug.h
sem.o: sem.c $H/kernel.h $H/as.h $H/queue.h $H/sem.h
svchandler.o: svchandler.c $H/kernel.h $H/as.h $H/queue.h $H/sem.h $H/trap.h \
	$H/server.h $H/errcode.h
thread.o: thread.c $H/kernel.h $H/as.h $H/memmap.h $H/queue.h $H/sem.h \
	$H/port.h $H/server.h $H/pageframe.h $H/errcode.h $H/kernserver.h \
	$H/debug.h
trap.o: trap.c $H/kernel.h $H/memmap.h $H/as.h $H/trap.h $H/debug.h
```

```
/*   Xlink    1.7    92/08/30.22:30:49 */

MEMORY {
      kernel : o=0x9000 l=2000000
}

SECTIONS {
      .text : { Kernel.oo(.text) }
      .data ALIGN(512) : { Kernel.oo(.data) }
      .bss : { Kernel.oo(.bss) }
}
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  addrspace.c    1.5    92/08/30    22:30:59
 *
 *  This module implements routines for manipulating address space control
 *  blocks (as opposed to address space memory).
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <as.h>
#include <vm.h>
#include <memmap.h>
#include <queue.h>
#include <sem.h>
#include <server.h>
#include <pageframe.h>
#include <errcode.h>
#include <kernserver.h>
#include <debug.h>
#include <port.h>

/*------------------------------------------------------- Exported Functions ----*/

AddrSpace      *CreateAddrSpace();
Errcode              RemoveAddrSpace();
void           InitAddrSpaceDesc();
void           GetAddrSpaceCap();
AddrSpace      *CheckAddrSpaceCap();

/*------------------------------------------------------- Imported Functions ----*/

extern Ulong  GetRand();
extern Void    *Kmalloc();
extern void   Kfree();
extern Void    *GetPageFrame();
extern void   RealMemFill();
extern void   Suicide();
void          SetCap();
extern MsgBuffer *CreateMsgBuffer();
extern Queue   *QueueCreate();

/*------------------------------------------------------- External Variables ----*/

extern Queue *svcWaitingList;
extern Semaphore *svcPickupSem;
extern Thread *currentThread;
extern AddrSpace *kernelAddrSpace;
extern Queue *addrSpaceList;
extern Queue   *addrSpaceList;
extern Port   kernelServerPort;

/*=============================================================================*\
 *  Allocate and initialize an address space control block and a page table.
\*=============================================================================*/

AddrSpace *CreateAddrSpace( priority )
     Ulong   priority;
```

```
{
      AddrSpace *as;
      Void      *pageTable;
      Queue     *asPortQueue;

      as = (AddrSpace *) Kmalloc( sizeof(AddrSpace) );
      pageTable = GetPageFrame( as, (Void*) 0, PFT_TYPE_PAGE_TABLE );
      RealMemFill( pageTable, VM_PAGE_TABLE_ENTRY_COUNT, VM_NULL_ENTRY );
      asPortQueue = QueueCreate();
      InitAddrSpaceDesc( as, pageTable, TRUE, priority, asPortQueue );
      as->lockingSem = (Void *) SemCreate( 1, SEM_PRIORITY_ADDR_SPACE );
      return( as );
}

/*============================================================================*\
 *  Initialize an address space control block.  Separated from CreateAddr-
 *  Space to allow for the statically allocated kernel address space for
 *  system bootstrapping.  The notification of death port is set to the
 *  message buffer recycler by default.
\*============================================================================*/

void InitAddrSpaceDesc( as, pageTable, swappable, priority, asPortQueue )
      AddrSpace      *as;
      Void           *pageTable;
      Boolean         swappable;
      Ulong           priority;
      Queue          *asPortQueue;
{
      as->addrSpaceMagicNumber = AS_MAGIC_NUM_ADDR_SPACE;
      as->priority = priority;
      as->addrSpaceCapabilityCheck = GetRand ();
      as->swappable = swappable;
      as->swappedIn = TRUE;
      as->ownedThreadList = (Thread *) NULL;
      as->memoryPageCount = 0;
      as->pageTable = pageTable;
      as->firstFreePage = 0;
      as->ownedPortList = (Void*) asPortQueue;
      as->notificationPortMachineID = SERV_ALIAS_MACHINE;
      as->notificationPortPortDesc  = SERV_ALIAS_MSG_RECYCLER;
      as->notificationPortPortCheck = 0;
      QueueInsert( addrSpaceList, as );
}

/*============================================================================*\
 *  Remove an address space.  This routine sends the notification of death
 *  message.  Sending the notification message would be better done by the
 *  kernel server.
\*============================================================================*/

Errcode RemoveAddrSpace( as )
      AddrSpace *as;
{
      Thread *th;
      MsgBuffer *notify;
      PortDesc *port;
```

```
#       ifdef DEBUG_ADDR_SPACE
        StatMsg("RemoveAddrSpace( as=0x%X )\n",as);
#       endif
        notify = CreateMsgBuffer( kernelAddrSpace, KS_SMALL_MSG_LENGTH );
        LockAddrSpace( as );

        /* remove from the address space list */
        QueueUnlink( addrSpaceList, (QueueEntry*) as );

        /* send the notification of death message */
        notify->destPort.machineID = as->notificationPortMachineID;
        notify->destPort.portDesc = as->notificationPortPortDesc;
        notify->destPort.portCheck = as->notificationPortPortCheck;
        notify->operationCode = KS_NOTIFY_MSG;
        GetAddrSpaceCap( &(notify->capability), as );
#       ifdef DEBUG_KERNEL_SERVER
        Kprintf("KernServ: notify: port=[%d 0x%X %d], as=0x%X\n",
                notify->destPort.machineID, notify->destPort.portDesc,
                notify->destPort.portCheck, as);
#       endif
        if (Send( notify ) != ERR_OK) {
                GetPortAlias( SERV_ALIAS_MSG_RECYCLER, &(notify->destPort) );
                Send( notify );
        }

        /* kill all threads in address space */
        while( (th=as->ownedThreadList) != NULL ) {
                SubRemoveThread( th );
        }

        /* remove all ports owned by the address space */
        while( QueueCount(as->ownedPortList) > 0 ) {
                port = (PortDesc*) QueueGetHead( as->ownedPortList );
                UnlockAddrSpace( as );  /* must be done */
                RemovePort( port );
                LockAddrSpace( as );
        }
        QueueRemove( as->ownedPortList );

        /* free all memory held by address space */
        FreePageTable( as );

        /* remove locking semaphore - unlock it to release its priority */
        SemSignal( as->lockingSem );
        SemRemove( (Semaphore *) as->lockingSem );

        /* remove address space descriptor */
        as->addrSpaceMagicNumber = KERN_UNMAGIC_NUMBER;
        Kfree( (Void *) as, sizeof(AddrSpace) );
        return( ERR_OK );
}

/*============================================================================*\
 *  Manufacture a capability for an address space.  Note that this function
 *  expects a capability pointer.
```

```
\*=============================================================================*/

void GetAddrSpaceCap( cap, addrSpace )
        Capability *cap;
        AddrSpace *addrSpace;
{
        SetCap( cap, kernelServerPort, (Ulong) addrSpace, SERV_CAP_RIGHTS_OWNER,
                addrSpace->addrSpaceCapabilityCheck );
}

/*=============================================================================*\
 *  Verify that a given capability is ok.  Return the corresponding address
 *  space control block pointer.
\*=============================================================================*/

AddrSpace *CheckAddrSpaceCap( cap )
        Capability cap;
{
        AddrSpace *as;

        as = (AddrSpace *) cap.object;
        if( /*Mapped(as)*/ TRUE ) {
                if( as->addrSpaceMagicNumber == AS_MAGIC_NUM_ADDR_SPACE ) {
                        /* run rights and check through encr.func & cmp */
                        if( cap.check == as->addrSpaceCapabilityCheck ) {
                                return( as );
                        }
                }
        }
        return( NULL );
}

/*-----------------------------------------------------------------------------*\
 *  END OF MODULE:  addrspace.c
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  diskswap.c    1.2    92/08/30    22:31:20
 *
 *  This module contains stubs for some of the routines that would be needed
 *  to implement disk paging and swapping.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <as.h>
#include <vm.h>

/*---------------------------------------------------- Exported Functions -----*/

void  SwapIn();                 /*( as )*/
void  SwapOut();                /*( as )*/
void  LoadPointerTable();       /*( as, virtAddr, pageTable )*/
void  UnloadPageFrame();        /*( realAddr )*/
void  ReleaseVmDiskBlock();     /*( blockNum )*/

/*=============================================================================*\
 *  Swap an address space in from disk.
\*=============================================================================*/

void SwapIn( as )
      AddrSpace *as;
{
      Kprintf("SwapIn( as=0x%X )\n", as);
      Panic("SwapIn function not implemented");
}

/*=============================================================================*\
 *  Swap an address space out to disk.
\*=============================================================================*/

void SwapOut( as )
      AddrSpace *as;
{
      Kprintf("SwapOut( as=0x%X )\n", as);
      Panic("SwapOut function not implemented");
}

/*=============================================================================*\
 *  Load a pointer table in from disk.  Modifies page table entry.
\*=============================================================================*/

void LoadPointerTable( as, virtAddr, pageTable )
      AddrSpace *as;
      Void      *virtAddr;
      PageTable *pageTable;
{
      Kprintf("LoadPointerTable( as=0x%X, virtaddr=0x%X, pageTable=0x%X\n",
            as, virtAddr, pageTable);
      Panic("LoadPointerTable: Don't know how to load a page");
}

/*=============================================================================*\
```

```
 *  Release a disk block that holds a page's contents, since the page has
 *  been modified or freed.
\*=============================================================================*/

void ReleaseVmDiskBlock( blockNum )
     Ulong   blockNum;
{
     Kprintf("ReleaseVmDiskBlock( blockNum=%d )\n", blockNum );
     Panic("ReleaseVmDiskBlock: function not impelemented");
}

/*-----------------------------------------------------------------------------*\
 *  END OF MODULE:  diskswap.c
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  freemem.c    1.4    92/08/30    22:31:27
 *
 *  This module implements the routines for freeing page frames from
 *  address spaces.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <vm.h>
#include <as.h>
#include <pageframe.h>

/*---------------------------------------------------- Exported Functions ----*/

void  FreePages();              /*( as, vaddr, npages )*/
void  FreePageTable();          /*( as )*/
void  FreePointerTable();       /*( as, pageTableEntry )*/

/*------------------------------------------------------ Local Functions ----*/

LOCAL void    FreePage();

/*---------------------------------------------------- Imported Functions ----*/

extern Ulong  RealMemPeek ();          /* realmem.s */
extern void   RealMemPoke ();          /* realmem.s */
extern void   LockPageFrameTable();    /* pageframe.c */
extern void   UnlockPageFrameTable(); /* pageframe.c */
extern void   UnreadyPageFrame();      /* pageframe.c */
extern void   FreePageFrame();         /* pageframe.c */
extern Ulong  LockPointerTable();      /* lock.c */
extern void   UnlockPointerTable();    /* lock.c */
extern void   InvalidateKernelAddr(); /* regwork.s */

/*---------------------------------------------------- Imported Variables ----*/

extern AddrSpace *kernelAddrSpace;

/*=============================================================================*\
 *  Free the specified pages of an address space.
 *  THIS FUNCTION IS NOT IMPLEMENTED.
 *  To implement this, change FreePointerTable to accept starting and ending
 *  pointer table entry numbers and free all pages between them, and remove
 *  the pointer table frame if there are no active pages left.  Then make
 *  this routine call FreePointerTable once for each pointer table involved
 *  with the freeing with appropriate parameters.
\*=============================================================================*/

void FreePages( as, addr, npages )
      AddrSpace *as;
      Void   *addr;
      Ulong   npages;
{
      Kprintf("FreePages( as=0x%X, addr=0x%X, npages=%d )\n",as,addr,npages);
      Panic("FreePages: function not implemented");
}
```

```
/*=============================================================================*\
 *  Free all of the descendent pages of a page table (ie. free all pages of
 *  an entire address space)
 *  ASSUMES THE OWNING ADDRESS SPACE IS LOCKED AND THE PAGE TABLE PAGE
 *  FRAMES ARE IN "UNREADY" STATE.
\*=============================================================================*/

void FreePageTable( as )
      AddrSpace *as;
{
      Ulong i, value;
      PointerTable *pointerTable;

      for( i=0; i<VM_PAGE_TABLE_ENTRY_COUNT; i++ ) {
              FreePointerTable( as, i );
      }
      FreePageFrame( (Void*) as->pageTable );
}

/*=============================================================================*\
 *  Free all of the descendent pages of a pointer table and then the
 *  pointer table itself.  This routine is called directly for removing
 *  message buffers.
\*=============================================================================*/

void FreePointerTable( as, pageEntry )
      AddrSpace      *as;
      Ulong           pageEntry;
{
      register Ulong  i, value;
      PointerTable   *pointerTable;
      Void           *pageEntryAddr, *pointerEntryAddr;
      CPint           invalidateAddress;

      /* lock and locate pointer table */
      value = LockPointerTable( as, pageEntry );
      if( value == VM_NULL_ENTRY ) return;
      pointerTable = (PointerTable*) (value & VM_PAGE_MASK);

      LockPageFrameTable();
      for( i=0; i<VM_POINTER_TABLE_ENTRY_COUNT; i++ ) {
              pointerEntryAddr = (Void*) &(pointerTable->entry[i]);
              value = RealMemPeek( pointerEntryAddr );
              if( value != VM_NULL_ENTRY ) {
                      RealMemPoke( pointerEntryAddr, VM_NULL_ENTRY );
                      if( as == kernelAddrSpace ) {
                              InvalidateKernelAddr( (pageEntry *
                                  VM_POINTER_TABLE_ENTRY_COUNT + i) *
                                  VM_PAGE_LENGTH );
                      }
                      FreePage( value );
              }
      }
      UnlockPageFrameTable();
```

```
	/* unmap the pointer table from the page table */
	pageEntryAddr=(Void*)&(((PageTable*)(as->pageTable))->entry[pageEntry]);
	RealMemPoke( pageEntryAddr, VM_NULL_ENTRY );

	/* free the pointer table page frame */
	/* The pointer table page frame is already "unready" from being locked*/
	FreePageFrame( (Void*) pointerTable );
}

/*============================================================================*\
 *  Free a page given the value in its pointer table entry.
\*============================================================================*/

LOCAL void FreePage( pointerValue )
	Ulong		pointerValue;
{
	Ulong	diskBlock;
	Void	*page;

	if( pointerValue & VM_PAGE_VALID_MASK ) {
		/* free the page frame */
		if( pointerValue & VM_SHARED_PAGE_MASK ) {
			/* but do not free shared pages.  Shared pages are */
			/* owned by the kernel address space and must never */
			/* be freed */
			return;
		}
		page = (Void*) (pointerValue & VM_PAGE_MASK);
		UnreadyPageFrame( page );
		UnlockPageFrameTable();
		FreePageFrame( page );
		LockPageFrameTable();
	} else {
		/* the page has been paged out to disk - release the block */
		/* (disk block numbers are assumed to be <= 29 bits) */
		diskBlock = (pointerValue & VM_DISK_BLOCK_MASK)
				>> VM_DISK_BLOCK_RIGHT_SHIFT;
		UnlockPageFrameTable();
		ReleaseVmDiskBlock( diskBlock );
		LockPageFrameTable();
	}
}

/*----------------------------------------------------------------------------*\
 *  END OF MODULE:  freemem.c
\*----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  getmem.c    1.10    92/08/30    22:31:43
 *
 *  This module implements the functions for mapping page frames into
 *  address spaces.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <vm.h>
#include <as.h>
#include <pageframe.h>
#include <debug.h>

/*------------------------------------------------------------ Exported Functions -----*/

Void   *GetPages();            /*( as, npages, access )*/
void  GetVaddrPages();         /*( as, virtAddr, npages, access )*/
void  MapPage();               /*( as, virtAddr, realAddr, access )*/
void  InitPageTable();         /*( pageTable )*/

/*------------------------------------------------------------ Local Functions -----*/

LOCAL Void     *GetPageRange();
LOCAL Ulong   FindFreePageSpot();
LOCAL Ulong   SearchPointerTable();
LOCAL void    CreatePointerTable();

/*------------------------------------------------------------ Imported Functions -----*/

extern Ulong  RealMemPeek();          /* realmem.c */
extern void   RealMemPoke();          /* realmem.c */
extern void   RealMemFill();          /* realmem.c */
extern Void    *GetPageFrame();               /* pageframe.c */
extern void   ReadyPageFrame();       /* pageframe.c */
extern Ulong  LockPointerTable();     /* lock.c */
extern void   UnlockPointerTable();   /* lock.c */

/*=============================================================================*\
 *  Maps a number of pages into an address space with certain access rights.
 *  Finds the first-free virtual address range large enough.
\*=============================================================================*/

Void *GetPages( as, pageCount, access )
      AddrSpace *as;
      Ulong   pageCount;
      Ulong   access;
{
      Void   *startAddr;

      LockAddrSpace( as );
      startAddr = GetPageRange( as, pageCount );
      GetVaddrPages( as, startAddr, pageCount, access );
      UnlockAddrSpace( as );
      return( startAddr );
}
```

```
/*=============================================================================*\
 *  Maps a number of pages into an address space with certain access rights
 *  starting from a certain virtual address.  The pages are zero-filled.
 *  ASSUMES ADDRESS SPACE IS ALREADY LOCKED.
\*=============================================================================*/

void GetVaddrPages( as, startAddr, pageCount, access )
      AddrSpace *as;
      Void   *startAddr;
      Ulong   pageCount;
      Ulong   access;
{
      Ulong   i, vpage, rpage;
      Void   *vaddr, *raddr;

      vaddr = startAddr;
      for( i=1; i<=pageCount; i++ ) {
              raddr = GetPageFrame( as, vaddr, PFT_TYPE_REGULAR );
              RealMemFill( raddr, VM_PAGE_LENGTH/sizeof(Ulong), 0 );
              MapPage( as, vaddr, raddr, access );
              ReadyPageFrame( raddr, /*modified*/ FALSE );
              vaddr = (Void*) ((char*)vaddr + VM_PAGE_LENGTH);
      }
}

/*=============================================================================*\
 *  Finds a virtual address range long enough to hold the given number of
 *  pages.  Note that this routine works with virtual page numbers
 *  internally rather than virtual addresses.
 *  ASSUMES THAT ADDRESS SPACE IS ALREADY LOCKED.
\*=============================================================================*/

LOCAL Void *GetPageRange( as, pageCount )
      AddrSpace *as;
      Ulong pageCount;
{
      Ulong firstPage, startPage, nextPage, newPage;
      Ulong findCountDown;

      firstPage = as->firstFreePage;
      newPage = FindFreePageSpot( as, firstPage );
      do {
              startPage = nextPage = newPage;
              findCountDown = pageCount;
              while (newPage == nextPage && --findCountDown != 0) {
                      nextPage++;
                      newPage = FindFreePageSpot (as, nextPage);
              }
      } while (newPage != nextPage && newPage != VM_NULL_PAGE_NUM);

      if (newPage == VM_NULL_PAGE_NUM) {
              Panic ("GetPageRange: no free range large enough");
              return( NULL );
      }
      if (startPage == firstPage) {
              as->firstFreePage = nextPage + 1;
```

```
	}
	return ( (Void*) (startPage*VM_PAGE_LENGTH) );
}

/*=============================================================================*\
 *  Searches for the next free page spot in a virtual address space, starting
 *  from the given virtual page number.  Works with page numbers rather than
 *  addresses.
 *  ASSUMES THAT ADDRESS SPACE IS ALREADY LOCKED.
\*=============================================================================*/

LOCAL Ulong FindFreePageSpot( as, findPage )
	AddrSpace *as;
	Ulong findPage;
{
	PageTable *pageTable;
	PointerTable *pointerTable;
	Ulong pageEntry, pointerEntry;
	register Ulong pageValue;
	Boolean look;

	pageTable = (PageTable *) as->pageTable;
	pageEntry = findPage / VM_POINTER_TABLE_ENTRY_COUNT;
	pointerEntry = findPage % VM_POINTER_TABLE_ENTRY_COUNT;
	look = TRUE;
	while (look && pageEntry < VM_PAGE_TABLE_ENTRY_COUNT) {
		pageValue = LockPointerTable( as, pageEntry );
		if (pageValue == VM_NULL_ENTRY) {
			look = FALSE;
		} else {
			if ((pageValue & VM_MSG_BUFFER_MASK) == 0) {
				pointerTable = (PointerTable *)
					(pageValue & VM_PAGE_MASK);
				pointerEntry = SearchPointerTable
					( pointerTable, pointerEntry );
				look=pointerEntry>=VM_POINTER_TABLE_ENTRY_COUNT;
			}
			UnlockPointerTable( (Void*)pointerTable, /*mod*/FALSE );
		}
		if (look) {
			pageEntry++;
			pointerEntry = 0;
		}
	}
	if (look) {
		return( VM_NULL_PAGE_NUM );
	}
	return( pageEntry * VM_POINTER_TABLE_ENTRY_COUNT + pointerEntry );
}

/*=============================================================================*\
 *  Searches the given pointer table for a single free page entry.
\*=============================================================================*/

LOCAL Ulong SearchPointerTable( pointerTable, pointerEntry )
	PointerTable *pointerTable;
```

```
        Ulong   pointerEntry;
{
        register Ulong  p, pv;
        Boolean         look;

        look = TRUE;
        p = pointerEntry;
        while (look && p < VM_POINTER_TABLE_ENTRY_COUNT) {
                  pv = RealMemPeek( &(pointerTable->entry[p]) );
                if (pv == VM_NULL_ENTRY) {
                        look = FALSE;
                } else {
                        p++;
                }
        }
        return( p );
}

/*=============================================================================*\
 *  Maps a physical page frame into a virtual address space.
 *  ASSUMES THE ADDRESS SPACE IS ALREADY LOCKED.
\*=============================================================================*/

void MapPage( as, virtAddr, realAddr, access )
        AddrSpace *as;
        Void    *virtAddr;
        Void    *realAddr;
        Ulong    access;
{
        Ulong   virtPage, pageFrame;
        PageTable *pageTable;
        PointerTable *pointerTable;
        Ulong   pageEntry, pointerEntry;
        Ulong   pageValue;
        Ulong   pointerValue;

#       ifdef DEBUG_VERIFY
        if( (CPint)virtAddr % VM_PAGE_LENGTH ) {
                Panic("MapPage: vaddr not aligned");
        }
        if( (CPint)realAddr % VM_PAGE_LENGTH ) {
                Panic("MapPage: raddr not aligned");
        }
#       endif
        virtPage  = (CPint)virtAddr / VM_PAGE_LENGTH;
        pageFrame = (CPint)realAddr / VM_PAGE_LENGTH;
        pageTable = (PageTable *) as->pageTable;
        pageEntry = virtPage / VM_POINTER_TABLE_ENTRY_COUNT;
        pointerEntry = virtPage % VM_POINTER_TABLE_ENTRY_COUNT;
        pageValue = LockPointerTable( as, pageEntry );
        if (pageValue == VM_NULL_ENTRY) {
                CreatePointerTable( as, pageEntry, access );
                pageValue = LockPointerTable( as, pageEntry );
        }
        pointerTable = (PointerTable *) (pageValue & VM_PAGE_MASK);
        pointerValue = RealMemPeek( &(pointerTable->entry[pointerEntry]) );
```

```
    if (pointerValue != VM_NULL_ENTRY) {
            Kprintf("MapPage: attempt to map to already allocated page!\n");
            Kprintf("virtualPage=%d, pageFrame=%d\n",virtPage,pageFrame);
            Panic ("Get your act together!");
    }
    RealMemPoke ( &(pointerTable->entry[pointerEntry]),
                  (pageFrame * VM_PAGE_LENGTH) | access );
    UnlockPointerTable( (Void*) pointerTable, /*modified*/ TRUE );
}

/*=============================================================================*\
 *  Allocates and initializes a pointer table.  Does not lock it.
\*=============================================================================*/

LOCAL void CreatePointerTable( as, pageEntry, pageAccess )
    AddrSpace *as;
    Ulong   pageEntry;
    Ulong   pageAccess;
{
    PageTable *pageTable;
    Ulong   i, pageValue;
    PointerTable *pointerTable;

    /* get a page frame for the pointer table */
    pointerTable = (PointerTable*) GetPageFrame( as,
        pageEntry * VM_POINTER_TABLE_ENTRY_COUNT * VM_PAGE_LENGTH,
        PFT_TYPE_POINTER_TABLE );

    /* initialize pointer table to empty */
#   ifdef DEBUG_GETMEM
    Kprintf ("Initializing pointer table real addr 0x%X...\n",p);
#   endif
    RealMemFill( pointerTable, VM_POINTER_TABLE_ENTRY_COUNT,VM_NULL_ENTRY );

    /* make page table entry point to new pointer table */
    pageValue = (CPint)pointerTable | VM_SUPER_VRW_MASK
        | (pageAccess & VM_USER_MASK) | (pageAccess & VM_MSG_BUFFER_MASK);
#   ifdef DEBUG_GETMEM
    Kprintf ("Poking page value (0x%X)...\n", pageValue);
#   endif
    pageTable = (PageTable *) as->pageTable;
    RealMemPoke( &(pageTable->entry[pageEntry]), pageValue );

    /* is wasteful but consistent to ready the pointer table frame */
    ReadyPageFrame( (Void*) pointerTable );
}

/*-----------------------------------------------------------------------------*\
 *  END OF MODULE:  getmem.c
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  getty.c    1.6    92/08/30    22:32:01
 *
 *  This module implements functions for starting the user processes and
 *  the message recycler server.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <memmap.h>
#include <vm.h>
#include <as.h>
#include <pageframe.h>
#include <debug.h>
#include <server.h>

/*---------------------------------------------------------- Exported Functions ----*/

void  Banner();
void  Getty();
void  RecyclerServer();

/*---------------------------------------------------------- Imported Functions ----*/

extern void   fexec();
extern Ulong  Align();

/*---------------------------------------------------------- External Variables ----*/

extern PageFrameTableEntry    pageFrameTable[VM_PAGE_COUNT];
extern Ulong  freePageFrameCount;
extern char   start, etext, edata, end;
extern Port   messageRecyclerPort;
extern Thread  *currentThread;
extern Boolean        kernelIsReady;
extern AddrSpace *kernelAddrSpace;
extern Ulong  gettyProcessCount;

/*=============================================================================*\
 *  Print the banner on the console.
\*=============================================================================*/

void Banner ()
{
      Ulong p, freeStart;

      StatMsg ("\nUNBOS version %s\n\n", VERSION);
      StatMsg ("%i bytes of real memory\n", VM_PAGE_COUNT * VM_PAGE_LENGTH);

#     ifdef DEBUG
      StatMsg("%s code=0x%X, data=0x%X, end_data=0x%X, end=0x%X\n",
            "kernel:", &start, &etext, &edata, &end );
#     endif

      freeStart = pageFrameTable[PFT_MASTER_FREE_FRAME].nextPage
            * VM_PAGE_LENGTH;
#     ifdef DEBUG_BANNER
```

```
	StatMsg ("0x%X first heap page address\n", Align(&end,VM_PAGE_LENGTH));
	StatMsg ("0x%X first free page address\n", freeStart);
#	endif
	StatMsg ("%i bytes free\n\n", freePageFrameCount * VM_PAGE_LENGTH);
}

/*============================================================================*\
 *  Start user processes (executing the command shell).
\*============================================================================*/

void Getty()
{
	int i;
	Capability termCap;
	char *passArgv[1];

	kernelIsReady = TRUE;
	gettyProcessCount = 0;
#	ifdef DEBUG
	StatMsg("BootupTh: Starting user-level processes\n");
#	endif

	/* start up user command shell(s) */
	passArgv[0] = "shell";
	GetPortAlias( SERV_ALIAS_TTY3, &termCap );
	fexec( "shell", 1, passArgv, NULL, termCap, termCap, termCap,
		AS_PRIORITY_USER_PROCESS, /*async*/ 1 );
	gettyProcessCount++;

	/* this is how tty port 2 would be started with a command shell */
	GetPortAlias( SERV_ALIAS_TTY2, &termCap );
	fexec( "shell", 1, passArgv, NULL, termCap, termCap, termCap,
		AS_PRIORITY_USER_PROCESS, /*async*/ 1 );
	gettyProcessCount++;

	/* the boot process now becomes the message recycler */
	/* (it would be better to make it become an "init" server) */
	currentThread->priority = AS_PRIORITY_RECYCLER_SERVER;
	RecyclerServer();
}

/*============================================================================*\
 *  Message buffer recycler server code.
\*============================================================================*/

void RecyclerServer()
{
	MsgBuffer *msg;

#	ifdef DEBUG
	StatMsg("Recycler: Message Recycler Server Ready\n");
#	endif
	while( TRUE ) {
		msg = Receive( messageRecyclerPort );
#		ifdef DEBUG_MSG_RECYCLER_SERVER
		StatMsg("Recycler: Recycling message at 0x%X\n", msg);
```

```
#               endif
                RemoveMsgBuffer( kernelAddrSpace, msg );
        }
}

/*-----------------------------------------------------------------------------*\
 *  END OF MODULE:  getty.c
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  globals.c    1.8    92/08/30    22:32:13
 *
 *  This module contains the global variable declarations for the kernel
 *  address space.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <vm.h>
#include <as.h>
#include <pageframe.h>
#include <queue.h>
#include <sem.h>
#include <server.h>

/*=============================================================================*\
 *  Address space and thread-related global variables.
\*=============================================================================*/

AddrSpace      *kernelAddrSpace;
Thread              *currentThread;
Thread              *previousThread;         /* for debugging */
Queue         *addrSpaceList;
Queue         *readyList;
Void            *kernelInfoPageAddr;
Boolean               kernelIsReady;
Ulong          gettyProcessCount;

/*=============================================================================*\
 *  Port-related global variables.
\*=============================================================================*/

Port           portAliasTable [SERV_PORT_ALIAS_COUNT];
Port           kernelServerPort;
Port           messageRecyclerPort;

/*=============================================================================*\
 *  Trap-related variables.
\*=============================================================================*/

Ulong          romMonIntbase;
Queue         *svcWaitingList;
Semaphore       *svcPickupSem;
Ulong          previousTrapMod;        /* for debugging */
Ulong          previousTrapModSave;    /* for debugging */
Ulong          previousRetiMP;         /* for debugging */
Ulong          previousRetiPC;         /* for debugging */

/*=============================================================================*\
 *  Kernel library global variable.
\*=============================================================================*/

Ulong          randSeed;
Semaphore      *kmallocSem;

/*=============================================================================*\
```

```
 *  Page frame table-related variables.
\*=========================================================================*/

PageFrameTableEntry   pageFrameTable [VM_PAGE_COUNT];
PageFrameTableEntry   masterFreeFrame;        /* head/tail pointers */
Ulong                 freePageFrameCount;
PageFrameTableEntry   masterLruFrame;         /* head/tail pointers */
Ulong                 lruPageFrameCount;
Semaphore            *pageFrameSem;

/*-------------------------------------------------------------------------*\
 *  END OF MODULE:  globals.c
\*-------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------*\
 *  MODULE:  icu.c    1.7    92/08/30    22:32:29
 *
 *  Interrupt Control Unit package
 *  written for Xinu        lsv  may 87
 *  modified for UNBOS      csb  jun 91
\*-------------------------------------------------------------------------*/

#include <kernel.h>
#include <memmap.h>
#include <icu.h>
#include <debug.h>

/*------------------------------------------------------- Exported Functions ----*/

void  IcuClockStart();          /*( )*/
void  IcuShutdown();            /*( )*/
void  IcuEnable();              /*( interruptMask )*/
void  IcuDisable();             /*( interruptMask )*/
Ulong IcuReadMask();            /*( ) : activeInterruptsMask */
void  IcuInit();                /*( )*/

/*=========================================================================*\
 * IcuClockStart - start counter for UNBOS clock interrupts
\*=========================================================================*/

void IcuClockStart()
{
      register Icu  *icuptr;   /* pointer to ICU registers */

#     ifdef DEBUG
      Kprintf ("Starting quantum clock running...\n");
#     endif
      icuptr = (Icu *) ADDR_ICU_SPACE;

      icuptr->mctl = ICU_FREEZE;          /* freeze current counters  */
      icuptr->cctl = ICU_STOP;            /* halt counters            */

      icuptr->lccv_l = ICU_LCSV_L;        /* reload current counters  */
      icuptr->lccv_h = ICU_LCSV_H;        /*  with start values       */
      icuptr->hccv_l = ICU_HCSV_L;
      icuptr->hccv_h = ICU_HCSV_H;

      icuptr->mctl = ICU_THAW;            /* thaw current counters    */
      icuptr->cctl = ICU_START;           /* start counters           */

      IcuEnable (ICU_CLOCK);
}

/*=========================================================================*\
 * IcuShutdown - stop clock and disable all interrupts
\*=========================================================================*/

void IcuShutdown()
{
      register Icu  *icuptr;   /* pointer to ICU registers */
```

```
        icuptr = (Icu *) ADDR_ICU_SPACE;
        icuptr->mctl = ICU_FREEZE;
        icuptr->cctl = ICU_STOP;
        IcuDisable (0xffff);
}

/*=============================================================================*\
 * IcuEnable - Clear a bit in the  the ICU interrupt mask register.
 *
 * Note:        Clearing a bit in this register causes interrupts on
 *              the corresponding pin to be enabled.
\*=============================================================================*/

void IcuEnable( imask )
        Ulong imask;
{
        register Icu  *icuptr;                  /* pointer to ICU registers */

        icuptr = (Icu *) ADDR_ICU_SPACE;
        icuptr->imsk_l &= ~(imask & 0x00ff);            /* pins 0 - 7       */
        icuptr->imsk_h &= ~((imask & 0xff00) >> 8);     /* pins 8 - 15      */
}

/*=============================================================================*\
 * IcuDisable - Set a bit in the ICU interrupt mask register.
 *
 * Note:         Setting a bit in this register causes interrupts on
 *               the corresponding pin to be disabled.
\*=============================================================================*/

void IcuDisable( imask )
        Ulong imask;
{
        register Icu  *icuptr;                 /* pointer to ICU registers */

        icuptr = (Icu *) ADDR_ICU_SPACE;
        icuptr->imsk_l |= imask & 0x00ff;           /* pins 0 - 7           */
        icuptr->imsk_h |= (imask & 0xff00) >> 8;    /* pins 8 - 15          */
}

/*=============================================================================*\
 * IcuReadMask - Read the interrupt mask registers of the ICU.
\*=============================================================================*/

Ulong IcuReadMask()
{
        register Icu  *icuptr;    /* pointer to ICU registers */

        icuptr = (Icu *) ADDR_ICU_SPACE;
        return( icuptr->imsk_h << 8 | icuptr->imsk_l );
}

/*=============================================================================*\
 * IcuInit - initialize interrupt control unit according
 *           to nsc recommended initialization sequence.
```

```
 *           Counters are concatenated.
\*=============================================================================*/

void IcuInit()
{
      register Icu  *icuptr;

#     ifdef DEBUG
      Kprintf ("Initializing ICU...\n");
#     endif
      icuptr = (Icu *) ADDR_ICU_SPACE;

      icuptr->mctl = 0xc2;           /* fixed priority, 8 bit bus   */
                                     /* current values frozen       */
      icuptr->cctl = 0xf0;           /* connected counters, halted  */

      /* load counters with initial values                          */
      icuptr->ciptr  = 0x60;         /* pin 6 counter interrupt     */
      icuptr->lcsv_l = ICU_LCSV_L;   /* 100 ms = 147,400  (base 10) */
      icuptr->lcsv_h = ICU_LCSV_H;   /*        = 00023fd0 (base 16) */
      icuptr->hcsv_l = ICU_HCSV_L;
      icuptr->hcsv_h = ICU_HCSV_H;

      /* place start value in current value for initial count       */
      icuptr->lccv_l = ICU_LCSV_L;
      icuptr->lccv_h = ICU_LCSV_H;
      icuptr->hccv_l = ICU_HCSV_L;
      icuptr->hccv_h = ICU_HCSV_H;

      icuptr->cictl   = 0x31;         /* H + L counter enable       */

      /* initialze for 8 bit bus mode                               */
      icuptr->ipsl   = 0xff;         /* interrupt/port select reg.  */
      icuptr->pdat   = 0x00;         /* port data                   */
      icuptr->pdir   = 0xff;         /* port direction - output     */
      icuptr->ocasn  = 0x00;         /* no clock output to pins     */

      icuptr->csrc_l = 0x00;         /* noncascaded icu             */
      icuptr->csrc_h = 0x00;
      icuptr->ipnd_l = 0x40;         /* clear all pending interrupts*/
      icuptr->ipnd_h = 0x40;

      /* These are the triggering states of the interrupt inputs.   */
      /*
              0  (jumper)             LL
              1  (sw)                 SW
              2  (uart 1, hi prio)    LL
              3  (sw)                 SW
              4  (uart 2, hi prio)    LL
              5  (sw)                 SW
              6  (clock)              CL
              7  (sw)                 SW
              8  (CIM gate array)     LL
              9  (sw)                 SW
              10 (uart 1, lo prio)    LL
              11 (sw)                 SW
```

```
                12 (uart 2, lo prio)    LL
                13 (scsi)               FE
                14 (printer)            HL
                15 (sw)                 SW

        LL = low level    FE = falling edge
        HL = high level   RE = rising edge
        SW = software     CL = clock (counter output)
        */

        icuptr->svct   = 0x10;          /* bias set to 1 - xxx1 vvvv   */
        icuptr->eltg_l = 0xff;          /* level triggering            */
        icuptr->eltg_h = 0xdf;
        icuptr->tpl_l  = 0x00;          /* triggering polarity         */
        icuptr->tpl_h  = 0x40;

        icuptr->fprt_l = 0x00;          /* pin 0 highest priority      */
        icuptr->fprt_h = 0x00;

        icuptr->mctl   = 0x82;          /* enable internal interrupts  */

        icuptr->imsk_l = 0xff;          /* all interrupt positions masked */
        icuptr->imsk_h = 0xff;

        icuptr->isrv_l = 0x00;          /* clear interrupts in service - csb */
        icuptr->isrv_h = 0x00;

        /* counters are initialized, halted and frozen */
}

/*-------------------------------------------------------------------------*\
 *  END OF MODULE:  icu.c
\*-------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  initkern.c    1.9    92/08/30    22:32:48
 *
 *  This module implements routines for initializing the kernel address
 *  space and the kernel threads.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <memmap.h>
#include <vm.h>
#include <as.h>
#include <queue.h>
#include <sem.h>
#include <pageframe.h>
#include <debug.h>

/*------------------------------------------------------ Exported Functions ----*/

void  InitKernelAddrSpace();
Ulong  *InitKernelThreads();
void  NullProgram();

/*------------------------------------------------------ Imported Functions ----*/

extern void   InitKmalloc();
extern Void    *Kmalloc();
extern void   Panic();
extern void   SvcPickup();
extern Ulong  GetPageCount();
extern void   InitPortAliasTable();
extern void   InitPageFrameTable();
extern Void    *GetPageFrame();
extern void   MapPage();
extern Ulong  RealMemPeek();

/*------------------------------------------------------ External Variables ----*/

extern char          *end;
extern char          *edata;
extern char          *etext;
extern Ulong          randSeed;
extern AddrSpace       *kernelAddrSpace;
extern Queue         *addrSpaceList;
extern Queue         *readyList;
extern Thread        *currentThread;
extern Queue         *svcWaitingList;
extern Semaphore       *svcPickupSem;
extern Semaphore       *pageFrameSem;
extern Void          *kernelInfoPageAddr;
extern PageFrameTableEntry pageFrameTable[VM_PAGE_COUNT];

/*=============================================================================*\
 *  Initializes the basic control blocks for the kernel address space and
 *  maps in the contents.
\*=============================================================================*/
```

```
void InitKernelAddrSpace()
{
      Ulong           i, startData, startHeap, bootDiskLimit;
      Ulong           sharedvPage;
      PageTable      *ptb0Table;

#     ifdef DEBUG
      Kprintf ("Initializing kernel address space...\n");
#     endif
      randSeed = 8324261;     /* a randomly generated seed may be better */

      /* initialize some statically allocated structures */
      /* they are statically allocated since they are needed to allow the */
      /* dynamic allocation mechanism to work */
      kernelAddrSpace = (AddrSpace *) ADDR_KERNEL_ADDR_SPACE;
      addrSpaceList = (Queue *) ADDR_ADDR_SPACE_LIST;
      QueueInit( addrSpaceList );
      QueueInit( (Queue*) ADDR_KERNEL_AS_PORT_QUEUE );
      InitAddrSpaceDesc (kernelAddrSpace, ADDR_PTB0_TABLE, FALSE,
              AS_PRIORITY_HIGHEST, (Queue*) ADDR_KERNEL_AS_PORT_QUEUE );
      kernelAddrSpace->lockingSem = (Void *) ADDR_SEM_KERNEL_AS;
      QueueInit( (Queue *) ADDR_SEM_KERNEL_AS_QUEUE );
      SemInit( (Semaphore *) ADDR_SEM_KERNEL_AS,
               (Queue *) ADDR_SEM_KERNEL_AS_QUEUE, 1,
               SEM_PRIORITY_ADDR_SPACE );
      pageFrameSem = (Semaphore *) ADDR_SEM_PAGEFRAME;
      QueueInit( (Queue *) ADDR_SEM_PAGEFRAME_QUEUE );
      SemInit( pageFrameSem, (Queue *) ADDR_SEM_PAGEFRAME_QUEUE, 1,
               SEM_PRIORITY_PAGE_FRAME_TABLE );

      /* allow semaphore locking */
      currentThread = (Thread *) ADDR_NULL_THREAD;
      currentThread->semMaxPriority = 0;

      InitPageFrameTable( kernelAddrSpace, (Void *) (&end) );

#     ifdef DEBUG
      Kprintf ("Initializing virtual memory tables...\n");
#     endif
      ptb0Table = (PageTable *) ADDR_PTB0_TABLE;
      for (i=0; i<VM_PAGE_TABLE_ENTRY_COUNT; i++) {
              ptb0Table->entry[i] = VM_NULL_ENTRY;
      }
      startData = (((Ulong)(&etext)) / VM_PAGE_LENGTH) * VM_PAGE_LENGTH;
      startHeap = Align (&end, VM_PAGE_LENGTH);

      /* map pre-code section as read-write */
      for (i=ADDR_PTB0_TABLE ; i<ADDR_UNBOS; i+=VM_PAGE_LENGTH)
              MapPage(kernelAddrSpace, (Void*)i, (Void*)i, VM_SUPER_VRW_MASK);
      /* map kernel code as read-only */
      for (i=ADDR_UNBOS; i<startData; i+=VM_PAGE_LENGTH)
              MapPage(kernelAddrSpace, (Void*)i, (Void*)i, VM_SUPER_VR_MASK);
      /* map kernel data and bss sections as read-write */
      for (i=startData; i<startHeap; i+=VM_PAGE_LENGTH)
              MapPage(kernelAddrSpace, (Void*)i, (Void*)i, VM_SUPER_VRW_MASK);
      /* map I/O devices as read-write */
```

```
        MapPage( kernelAddrSpace, (Void*)ADDR_IO_SPACE, (Void*)ADDR_IO_SPACE,
                 VM_SUPER_VRW_MASK );
        MapPage( kernelAddrSpace, (Void*)ADDR_ICU_SPACE, (Void*)ADDR_ICU_SPACE,
                 VM_SUPER_VRW_MASK );
        /* get and map the kernel information page frame */
        kernelInfoPageAddr = GetPageFrame( kernelAddrSpace,
                (Void*)ADDR_LAST_THREAD_RESTORED, PFT_TYPE_REGULAR );
        MapPage( kernelAddrSpace, ADDR_LAST_THREAD_RESTORED,
                kernelInfoPageAddr, VM_SUPER_VRW_MASK );

        /* map in the boot disk image as read-only*/
        bootDiskLimit = Align( ADDR_BOOT_DISK_IMAGE + RealMemPeek(
                        (Ulong*) ADDR_BOOT_DISK_IMAGE ), VM_PAGE_LENGTH );
        for (i=ADDR_BOOT_DISK_IMAGE; i<bootDiskLimit; i+=VM_PAGE_LENGTH)
                MapPage(kernelAddrSpace, (Void*)i, (Void*)i, VM_SUPER_VR_MASK);

        /* set start of dynamic memory heap space */
        kernelAddrSpace->firstFreePage
                = pageFrameTable[PFT_MASTER_FREE_FRAME].nextPage;
}

/*=============================================================================*\
 *  Initialize the kernel threads and related control blocks.
\*=============================================================================*/

Ulong *InitKernelThreads()
{
        Thread *nullThread, *bootThread, *svcThread;
        char *nullStack, *bootStack, *svcStack;

#       ifdef DEBUG
        Kprintf ("Initializing kernel threads...\n");
#       endif

        /* initialize important control blocks */
        InitKmalloc();
        readyList       = QueueCreate();
        svcWaitingList  = QueueCreate();
        svcPickupSem    = SemCreate( 0, SEM_PRIORITY_NULL );
        InitPortAliasTable();

        /* create null thread */
        nullThread = (Thread *) ADDR_NULL_THREAD;
        nullStack = Kmalloc( AS_NULL_STACK_LENGTH );
        InitThreadDesc (nullThread, kernelAddrSpace, NullProgram,
                        AS_REG_KERNEL_PSR, AS_PRIORITY_NULL_THREAD,
                        nullStack, AS_NULL_STACK_LENGTH, "Null");
        nullThread->state = AS_THREAD_STATE_READY;
        QueueInsert (readyList, (QueueEntry *) nullThread);

        /* create the SVC server thread */
        svcThread = (Thread *) Kmalloc( sizeof(Thread) );
        svcStack = Kmalloc( AS_BOOT_STACK_LENGTH );
        InitThreadDesc (svcThread, kernelAddrSpace, SvcPickup,
                        AS_REG_KERNEL_PSR, AS_PRIORITY_SVC_SERVER,
                        svcStack, AS_BOOT_STACK_LENGTH, "SvcPickup");
```

```
	QueueInsert ( readyList, (QueueEntry *) svcThread );

	/* create the boot thread control block and make it current */
	bootThread = (Thread *) Kmalloc( sizeof(Thread) );
	bootStack = Kmalloc( AS_BOOT_STACK_LENGTH );
	InitThreadDesc (bootThread, kernelAddrSpace, Panic,
			AS_REG_BOOT_PSR, AS_PRIORITY_BOOT_THREAD,
			bootStack, AS_BOOT_STACK_LENGTH, "Boot");
	currentThread = bootThread;
	*((Thread **) (ADDR_LAST_THREAD_RESTORED)) = bootThread;
	bootThread->state = AS_THREAD_STATE_CURRENT;

	/* return the address of the current thread register set so */
	/* the calling routine can easily initialize the registers */
	return ( &(bootThread->reg[0]) );
}

/*=============================================================================*\
 *  This is the function the null thread executes when the operating system
 *  has nothing else to do.  All it does is wait for an interrupt which will
 *  potentially wake up another process.
\*=============================================================================*/

void NullProgram()
{
	while (TRUE) {
		asm ("wait");
	}
}

/*-----------------------------------------------------------------------------*\
 *  END OF MODULE:  initkern.c
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------*\
 *  MODULE:  initserver.c    1.6    92/08/30    22:33:04
 *
 *  This module starts all of the device and local kernel servers.  Note
 *  that the configuration of the servers is hard-coded here.
\*-----------------------------------------------------------------------*/

#include <kernel.h>
#include <as.h>
#include <server.h>
#include <art.h>
#include <queue.h>
#include <sem.h>
#include <tty.h>
#include <xbi401.h>
#include <kernserver.h>
#include <debug.h>

/*------------------------------------------------- Exported Functions ----*/

void  StartLocalServers();

/*------------------------------------------------- Imported Functions ----*/

extern void   TtyServer();
extern void   AddrSpaceServer();
extern void   BootServer();
extern void   DiskServer();
extern Void    *Kmalloc();
extern int    Resume();
extern MsgBuffer *CreateMsgBuffer();

/*------------------------------------------------- External Variables ----*/

extern AddrSpace *kernelAddrSpace;
extern Port   kernelServerPort;
extern Port   messageRecyclerPort;
extern TtyDesc        ttyTable[];
extern XebecAllocMap xebecAllocMap[];

/*=======================================================================*\
 *  Start the local servers.  The code for starting a thread is rather ugly
 *  since formal routines for creating a thread are not implemented.  All of
 *  the servers that use the system call library routines are registered into
 *  the thread table in the kernel address space creation message buffer.
\*=======================================================================*/

void StartLocalServers()
{
      Thread *servthread;
      Port    servport;
      Void   *servstack;
      Ulong   servsp;
      Ulong   i;
      Void   *ttyptr;
      MsgBuffer      *servmsg;
```

```
    Capability      servcap;

    /* start Console Server */
    /* must be done first for displaying system status messages */
#   ifdef DEBUG
    Kprintf("Starting Local Console Server for 'tty4'\n");
#   endif
    CreatePort( kernelAddrSpace, &servport );
    SetPortAlias( SERV_ALIAS_CONSOLE, servport );
    SetPortAlias( SERV_ALIAS_KERN_STATUS, servport );
    servthread = (Thread *) Kmalloc( sizeof(Thread) );
    servstack = Kmalloc( 512 );
    InitThreadDesc( servthread, kernelAddrSpace, TtyServer,
            AS_REG_KERNEL_PSR, AS_PRIORITY_DEVICE_SERVER, servstack,
            512, "ConServer");
    ttyptr = (Void *) &ttyTable[0];
    servsp=InitStack( servstack, 512, servport, (MsgBuffer *) NULL, 1,
            ttyptr);
    servthread->reg [AS_REG_SP] = servsp;
    Resume( servthread );

    /* start local kernel server */
#   ifdef DEBUG
    StatMsg("Starting Local Kernel Server\n");
#   endif
    servport = kernelServerPort; /* port already exists */
    SetPortAlias( SERV_ALIAS_GLOBAL_AS_SERVER, servport );
    SetPortAlias( SERV_ALIAS_KERN_SERVER, servport );
    servthread = (Thread *) Kmalloc( sizeof(Thread) );
    servstack = Kmalloc( 512 );
    InitThreadDesc( servthread, kernelAddrSpace, AddrSpaceServer,
            AS_REG_KERNEL_PSR, AS_PRIORITY_KERNEL_SERVER, servstack,
            512, "LocalKernServer");
    servsp=InitStack( servstack, 512, servport, (MsgBuffer *) NULL, 0);
    servthread->reg [AS_REG_SP] = servsp;
    /* register the thread for syscall library */
    CreatePort( kernelAddrSpace, &servport );
    servmsg = CreateMsgBuffer( kernelAddrSpace, KS_INIT_MSG_BUFFER_LENGTH );
    GetThreadCap( &servcap, servthread );
    _RegisterThread( servthread, servmsg, servport, servcap );
    Resume( servthread );

    /* make the message recycler server port */
#   ifdef DEBUG
    StatMsg("Starting Local Message Recycler Server\n");
#   endif
    CreatePort( kernelAddrSpace, &servport );
    SetPortAlias( SERV_ALIAS_MSG_RECYCLER, servport );
    messageRecyclerPort = servport;

    /* start Floppy Disk Block Server */
#   ifdef DEBUG
    StatMsg("Starting Local Floppy Disk Block Server\n");
#   endif
    CreatePort( kernelAddrSpace, &servport );
    SetPortAlias( SERV_ALIAS_DISK_BLOCK, servport );
```

```
        servthread = (Thread *) Kmalloc( sizeof(Thread) );
        servstack = Kmalloc( 512 );
        InitThreadDesc( servthread, kernelAddrSpace, DiskServer,
                AS_REG_KERNEL_PSR, AS_PRIORITY_DEVICE_SERVER, servstack,
                512, "FloppyServer");
        servsp=InitStack( servstack, 512, servport, (MsgBuffer *) NULL, 5,
                (char *) 0xA000A0, /*subch*/3, /*target*/1, /*drive*/0,
                &(xebecAllocMap[0]) );
        servthread->reg [AS_REG_SP] = servsp;
        /* register the thread for syscall library */
        CreatePort( kernelAddrSpace, &servport );
        servmsg = CreateMsgBuffer( kernelAddrSpace, KS_INIT_MSG_BUFFER_LENGTH );
        GetThreadCap( &servcap, servthread );
        _RegisterThread( servthread, servmsg, servport, servcap );
        Resume( servthread );

        /* start Terminal Server for terminal on serial port 3 */
#       ifdef DEBUG
        StatMsg("Starting Local Terminal Server for 'tty3'\n");
#       endif
        CreatePort( kernelAddrSpace, &servport );
        SetPortAlias( SERV_ALIAS_TTY3, servport );
        servthread = (Thread *) Kmalloc( sizeof(Thread) );
        servstack = Kmalloc( 512 );
        InitThreadDesc( servthread, kernelAddrSpace, TtyServer,
                AS_REG_KERNEL_PSR, AS_PRIORITY_DEVICE_SERVER, servstack,
                512, "Tty3Server");
        ttyptr = (Void *) &ttyTable[1];
        servsp=InitStack( servstack, 512, servport, (MsgBuffer *) NULL, 1,
                ttyptr);
        servthread->reg [AS_REG_SP] = servsp;
        Resume( servthread );

        /* start Terminal Server for terminal on serial port 2 */
#       ifdef DEBUG
        StatMsg("Starting Local Terminal Server for 'tty2'\n");
#       endif
        CreatePort( kernelAddrSpace, &servport );
        SetPortAlias( SERV_ALIAS_TTY2, servport );
        servthread = (Thread *) Kmalloc( sizeof(Thread) );
        servstack = Kmalloc( 512 );
        InitThreadDesc( servthread, kernelAddrSpace, TtyServer,
                AS_REG_KERNEL_PSR, AS_PRIORITY_DEVICE_SERVER, servstack,
                512, "Tty2Server");
        ttyptr = (Void *) &ttyTable[2];
        servsp=InitStack( servstack, 512, servport, (MsgBuffer *) NULL, 1,
                ttyptr);
        servthread->reg [AS_REG_SP] = servsp;
        Resume( servthread );

        /* start Boot Disk File Server */
#       ifdef DEBUG
        StatMsg("Starting Local Boot Disk File Server\n");
#       endif
        CreatePort( kernelAddrSpace, &servport );
        SetPortAlias( SERV_ALIAS_BOOT_DISK, servport );
```

```
        SetPortAlias( SERV_ALIAS_PREFIX_SERVER, servport );
        servthread = (Thread *) Kmalloc( sizeof(Thread) );
        servstack = Kmalloc( 512 );
        InitThreadDesc( servthread, kernelAddrSpace, BootServer,
                AS_REG_KERNEL_PSR, AS_PRIORITY_DEVICE_SERVER, servstack,
                512, "BootServer");
        servsp=InitStack( servstack, 512, servport, (MsgBuffer *) NULL, 0 );
        servthread->reg [AS_REG_SP] = servsp;
        /* register the thread for syscall library */
        CreatePort( kernelAddrSpace, &servport );
        servmsg = CreateMsgBuffer( kernelAddrSpace, KS_INIT_MSG_BUFFER_LENGTH );
        GetThreadCap( &servcap, servthread );
        _RegisterThread( servthread, servmsg, servport, servcap );
        Resume( servthread );
}

/*-------------------------------------------------------------------------------*\
 *  END OF MODULE:  initserver.c
\*-------------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  initsys.c    1.4    92/08/30    22:33:21
 *
 *  This module implements the code for initializing the kernel address space
 *  creation message that is used by the system call library.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <as.h>
#include <server.h>
#include <kernserver.h>
#include <fileserver.h>
#include <debug.h>

/*---------------------------------------------------------- Exported Functions ----*/

void  InitKernelSystemCalls();/*( )*/
void  InitAmsg();             /*( asCreationMsg, as, th, initMsgSize )*/
void  InitKernelFileEntries();/*( asCreationMsg )*/

/*---------------------------------------------------------- Imported Functions ----*/

extern void CreatePort();
extern MsgBuffer *CreateMsgBuffer();

/*---------------------------------------------------------- External Variables ----*/

extern AddrSpace *kernelAddrSpace;
extern AddrSpace *currentThread;
extern Port   kernelServerPort;

/*=============================================================================*\
 *  Get and initialize the kernel address space creation message buffer.
 *  (This is not done by the kernel server since the kernel address space
 *  already exists when the operating system starts.)
\*=============================================================================*/

void InitKernelSystemCalls()
{
      CreateAddrSpaceMsg *amsg;

#     ifdef DEBUG
      Kprintf("Initializing Kernel Address Space system call library...\n");
#     endif

      /* have to create kernel service port now for making capabilities */
      CreatePort( kernelAddrSpace, &kernelServerPort );

      /* create the creation message and make sure it gets mapped in */
      /* at the correct fixed address */
      amsg = (CreateAddrSpaceMsg *) CreateMsgBuffer( kernelAddrSpace,
              sizeof(CreateAddrSpaceMsg) );
      if ((CPint)amsg != KS_ADDR_ADDR_SPACE_MSG) {
              Panic("KernSyscalls: amsg did not map in at 0xFE0000");
      }
```

```
	/* initialize the creation message buffer */
	InitAmsg( amsg, kernelAddrSpace, currentThread,
		KS_INIT_MSG_BUFFER_LENGTH );
	InitKernelFileEntries( amsg );
#	ifdef DEBUG
	Kprintf("finished initializing kernel sys calls\n");
#	endif
}

/*============================================================================*\
 *  Initialize an address space creation message buffer.
\*============================================================================*/

void InitAmsg( amsg, as, th, initMsgSize )
	CreateAddrSpaceMsg *amsg;
	AddrSpace *as;
	Thread *th;
	Ulong   initMsgSize;
{
	Port port;
	MsgBuffer *msg;
	ThreadEntry *the;
	Ulong i;

	GetAddrSpaceCap( &(amsg->addrSpaceCap), as );

	/* initialize thread table */
	for (i=1; i<KS_MAX_THREADS; i++) {
		amsg->threadtab[i].used = FALSE;
	}
	the = &(amsg->threadtab[0]);

	/* register the boot thread for syscall library usage */
	the->used = TRUE;
	the->threadID = (Ulong) th;
	GetThreadCap( &(the->threadCap), th );
	the->threadMsgBuffer = CreateMsgBuffer( as, initMsgSize );
	CreatePort( as, &(the->threadPort) );
	the->threadErrno = 0;
	amsg->initThreadStackLength = AS_DEFAULT_STACK_LENGTH;

	/* initialize inherited file table */
	GetPortAlias( SERV_ALIAS_PREFIX_SERVER, &(amsg->rootSearchCap.port) );
	amsg->currentSearchCap = amsg->rootSearchCap;
	for (i=0; i<KS_MAX_FILES; i++) {
		amsg->filetab[i].fileMsgBuffer = (MsgBuffer *) NULL;
	}
}

/*============================================================================*\
 *  Initialize the kernel address space as having already opened its "std"
 *  files to the console.  Initializes the inherited "SearchCap" current
 *  and root directories to the boot disk server.
\*============================================================================*/

void InitKernelFileEntries( amsg )
```

```
        CreateAddrSpaceMsg *amsg;
{
        FileEntry *file;
        Ulong i;

        /* initialize file table */
        for (i=0; i<KS_MAX_FILES; i++) {
                amsg->filetab[i].used = FALSE;
        }
        file = &(amsg->filetab[0]);
        file->used = TRUE;
        file->fileMsgBuffer = (MsgBuffer *) NULL;
        GetPortAlias( SERV_ALIAS_CONSOLE, &(file->fileCap.port) );
        amsg->filetab[1] = *file;
        amsg->filetab[2] = *file;

        /* initialize directory capabilities */
        GetPortAlias( SERV_ALIAS_PREFIX_SERVER, &(amsg->rootSearchCap) );
        amsg->currentSearchCap = amsg->rootSearchCap;
}

/*-------------------------------------------------------------------------------*\
 *  END OF MODULE:  initsys.c
\*-------------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  kernlib.c    1.10    92/08/30    22:33:45
 *
 *  This module implements library-type functions used throughout the kernel.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <memmap.h>
#include <vm.h>
#include <as.h>
#include <queue.h>
#include <sem.h>
#include <art.h>
#include <syslib.h>
#include <server.h>

/*---------------------------------------------------------- Exported Functions -----*/

int   Kprintf();      /*( fmt, ... ) : charCount */
void  InitKmalloc();  /*( )*/
Void   *Kmalloc();    /*( size ) : ptr */
void  Kfree();        /*( ptr, size ) */
Ulong Align();        /*( address, boundary ) : alignedAddress */
Ulong GetPageCount(); /*( byteCount ) : pageCount */
Ulong GetRand();      /*( ) : randNum */
void  SetCap();       /*( cap, port, object, rights, check )*/

/*---------------------------------------------------------- Local Functions -----*/

LOCAL void    Kputc();

/*---------------------------------------------------------- Imported Functions -----*/

extern int    _DoPrint();     /* strlib/doprint.c */
extern Void    *_rawMalloc(); /* syslib/malloc.c */
extern void   _rawFree();     /* syslib/malloc.c */
extern Void    *GetPages();   /* pageframe.c */

/*---------------------------------------------------------- External Variables -----*/

extern Ulong  randSeed;
extern Void    *kernelAddrSpace;
extern MemBlock mallocMasterBlock;
extern Semaphore *kmallocSem;

/*=============================================================================*\
 *  Polled printf function.
\*=============================================================================*/

int Kprintf( format, args )
      char    format[];
      Ulong   args;
{
      int     charCount;
      Boolean intsave;
```

```
        intsave = DisableInterrupts();
        charCount = _DoPrint( Kputc, 0, format, &args );
        RestoreInterrupts( intsave );
        return( charCount );
}

/*=============================================================================*\
 *  Initialize the dynamically allocated memory for the kernel address space.
\*=============================================================================*/

void InitKmalloc()
{
        Queue *kmallocQueue;

        mallocMasterBlock.nextBlock   = NULL;
        mallocMasterBlock.blockLength = 0;
        kmallocSem = (Semaphore *) ADDR_SEM_KMALLOC;
        kmallocQueue = (Queue *) ADDR_SEM_KMALLOC_QUEUE;
        QueueInit( kmallocQueue );
        SemInit( kmallocSem, kmallocQueue, 1, SEM_PRIORITY_MALLOC );
#       ifdef DEBUG_KMALLOC
        Kprintf("InitKmalloc completed\n");
#       endif
}

/*=============================================================================*\
 *  Dynamically allocate memory for kernel structures.  This routine differs
 *  from the system call library malloc function in that a semaphore is used
 *  for locking and the kernel server cannot be called to map new pages
 *  into the address space (pages are mapped directly).
\*=============================================================================*/

Void *Kmalloc( size )
        Ulong   size;
{
        Void   *p;
        Ulong   npages;

#       ifdef DEBUG_KMALLOC
        Kprintf("Kmalloc( size=%d )\n", size);
#       endif
        SemWait( kmallocSem );
        p = _rawMalloc( size );
        SemSignal( kmallocSem );
        if( p == NULL ) {
                npages = GetPageCount(
                    (size>MALLOC_GETMEM_SIZE) ? size : MALLOC_GETMEM_SIZE );
                p = GetPages( kernelAddrSpace, npages, VM_SUPER_VRW_MASK );
                SemWait( kmallocSem );
                _rawFree( p, npages * VM_PAGE_LENGTH );
                p = _rawMalloc( size );
                SemSignal( kmallocSem );
#               ifdef DEBUG_KMALLOC
                Kprintf("Kmalloc: got %d more pages\n", npages);*/
#               endif
        }
```

```
#     ifdef DEBUG_KMALLOC
      Kprintf("Kmalloc: finished: p=0x%X, size=%d\n",p, size);
#     endif
      return( p );
}

/*=============================================================================*\
 *  Dynamically deallocate memory for kernel structures.
\*=============================================================================*/

void Kfree( p, size )
      Void   *p;
      Ulong   size;
{
#     ifdef DEBUG_KMALLOC
      Kprintf("Kfree( p=0x%X, size=%d )\n", p, size );
#     endif
      SemWait( kmallocSem );
      _rawFree( (Void*)p, size );
      SemSignal( kmallocSem );
#     ifdef DEBUG_KMALLOC
      Kprintf("Kfree exiting\n");
#     endif
}

/*=============================================================================*\
 *  Return the given address aligned (with round-up) to the given boundary.
\*=============================================================================*/

Ulong Align( address, boundry )
      Ulong   address;
      Ulong   boundry;
{
      if (address % boundry == 0) return (address);
      return( (address / boundry + 1) * boundry );
}

/*=============================================================================*\
 *  Return the number of pages (with round-up) specified by the given number
 *  of bytes.
\*=============================================================================*/

Ulong GetPageCount( byteCount )
      Ulong   byteCount;
{
      return( (byteCount + (VM_PAGE_LENGTH - 1)) / VM_PAGE_LENGTH );
}

/*=============================================================================*\
 *  Return a 32-bit random number.  Used for manufacturing secret numbers.
\*=============================================================================*/

Ulong GetRand()
{
      randSeed = randSeed * SYSLIB_RAND_MULT + SYSLIB_RAND_ADD;
      return( randSeed );
```

```
}

/*============================================================================*\
 *  Print a single character to the console in polled mode.
\*============================================================================*/

LOCAL void Kputc (c)
      char c;
{
      if (c == '\n') Kputc( '\r' );
      while ( (* (Uchar*) (ADDR_CONSOLE_STAT_REG) & ARTTXRDY)==0 ) ;
      * (Uchar*) (ADDR_CONSOLE_DATA_REG) = c;
}

/*============================================================================*\
 *  Set the fields for manufacturing a capability.
\*============================================================================*/

void SetCap( cap, port, object, rights, check )
      Capability *cap;
      Port    port;
      Ulong   object, rights, check;
{
      cap->port = port;
      cap->object = object;
      cap->rights = rights;
      cap->check = check;
}

/*----------------------------------------------------------------------------*\
 *  END OF MODULE:  kernlib.c
\*----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  kernserver.c    1.5    92/08/30    22:34:21
 *
 *  This module implements most of the kernel server functions.  Note that
 *  some of the kernel server calls are not implemented (since they are not
 *  needed by the rest of the system).
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <as.h>
#include <server.h>
#include <kernserver.h>
#include <errcode.h>
#include <vm.h>
#include <debug.h>

/*------------------------------------------------------- Exported Functions ----*/

void          AddrSpaceServer();

/*-------------------------------------------------------- Local Functions ----*/

LOCAL void    ReqCreatePort();
LOCAL void    ReqCreateMsgBuffer();
LOCAL void    ReqRemoveThread();
LOCAL void    ReqCreateMemory();

/*------------------------------------------------------ Imported Functions ----*/

extern MsgBuffer *ReqCreateAddrSpace();
extern void   ReqRemoveAddrSpace();
extern AddrSpace *CheckAddrSpaceCap();
extern Thread  *CheckThreadCap();
extern MsgBuffer *CreateMsgBuffer();
extern void   CreatePort();
extern Errcode  RemoveAddrSpace();
extern void   Kill();
extern Ulong  GetPageCount();
extern Void    *GetPages();
extern Void    *GetPageFrame();
extern void   GetVaddrPages();

/*=============================================================================*\
 *  Main routine of the kernel server.  Accept request, process, and reply.
\*=============================================================================*/

void AddrSpaceServer( port )
      Port port;
{
      MsgBuffer *request;

#     ifdef DEBUG
      StatMsg("Kernel Server ready.\n");
#     endif
      while( TRUE ) {
            request = Receive( port );
```

```
                request->destPort = request->replyPort;
                request->errorCode = ERR_OK;
                switch (request->operationCode) {
                case KS_CREATE_ADDR_SPACE:
                        request = ReqCreateAddrSpace( request );
                        break;
                case KS_REMOVE_ADDR_SPACE:
                        ReqRemoveAddrSpace( request );
                        break;
                case KS_REMOVE_THREAD:
                        ReqRemoveThread( request );
                        break;
                case KS_CREATE_MEMORY:
                        ReqCreateMemory( request );
                        break;
                case KS_CREATE_PORT:
                        ReqCreatePort( request );
                        break;
                case KS_CREATE_MSG_BUFFER:
                        ReqCreateMsgBuffer( request );
                        break;
                default:
                        request->errorCode = ERR_SERVER_UNKNOWN_OPCODE;
                }
                Send( request );
        }
}

/*============================================================================*\
 *  Handle request for the creation of a port.
\*============================================================================*/

LOCAL void ReqCreatePort( request )
        MsgBuffer *request;
{
        Errcode error;
        AddrSpace *as;

        as = CheckAddrSpaceCap( request->capability );
        if( as != (AddrSpace *) NULL ) {
                CreatePort( as, &(request->capability.port) );
        } else {
                request->errorCode = ERR_SERVER_BAD_CAPABILITY;
        }
}

/*============================================================================*\
 *  Handle request for the creation of a message buffer.
\*============================================================================*/

LOCAL void ReqCreateMsgBuffer( request )
        MsgBuffer *request;
{
        Errcode error;
        AddrSpace *as;
        MsgBuffer *msg;
```

```
	as = CheckAddrSpaceCap( request->capability );
	if( as != (AddrSpace *) NULL ) {
		msg = CreateMsgBuffer( as, request->data[0] );
		request->data[0] = (Ulong) msg;
	} else {
		request->errorCode = ERR_SERVER_BAD_CAPABILITY;
	}
}

/*============================================================================*\
 *  Handle request for the removal of a thread.
\*============================================================================*/

LOCAL void ReqRemoveThread( request )
	MsgBuffer *request;
{
	Thread *th;

	th = CheckThreadCap( request->capability );
#	ifdef DEBUG_KERN_SERVER
	StatMsg("KernServ: RemoveThread: th=0x%X\n", th);
#	endif
	if( th != NULL ) {
		request->errorCode = RemoveThread( th );
	} else {
		request->errorCode = ERR_SERVER_BAD_CAPABILITY;
	}
}

/*============================================================================*\
 *  Handle request for the allocation and mapping of memory into an
 *  address space.
\*============================================================================*/

LOCAL void ReqCreateMemory( request )
	MsgBuffer *request;
{
	AddrSpace *as;
	Ulong pages;
	Void *start;

	as = CheckAddrSpaceCap( request->capability );
	if( as == (AddrSpace *) NULL ) {
		request->errorCode = ERR_SERVER_BAD_CAPABILITY;
		return;
	}
	pages = GetPageCount( request->data[0] );
	start = GetPages( as, pages, VM_USER_VRW_MASK );
#	ifdef DEBUG_KERN_SERVER
	StatMsg("%s CreateMemory: as=0x%X, rqlen=%d; gave %i pages from 0x%X\n",
		"KernServ:", as, request->data[0], pages, start);
#	endif
	request->data[0] = (Ulong) start;
	request->data[1] = pages * VM_PAGE_LENGTH;
}
```

```
/*-------------------------------------------------------------------------*\
 *  END OF MODULE:  kernserver.c
\*-------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  kernservpr.c    1.2    92/08/30    22:35:24
 *
 *  This module implements the kernel server functions for creating and
 *  removing user processes.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <as.h>
#include <server.h>
#include <kernserver.h>
#include <errcode.h>
#include <debug.h>
#include <vm.h>
#include <memmap.h>
#include <pageframe.h>

/*------------------------------------------------------ Exported Functions ----*/

MsgBuffer *ReqCreateAddrSpace();
void  ReqRemoveAddrSpace();

/*------------------------------------------------------ Imported Functions ----*/

extern AddrSpace *CheckAddrSpaceCap();
extern Thread  *CheckThreadCap();
extern MsgBuffer *CreateMsgBuffer();
extern void   CreatePort();
extern Errcode  RemoveAddrSpace();
extern void   Kill();
extern Ulong  GetPageCount();
extern Void    *GetPages();
extern Void    *GetPageFrame();
extern void   GetVaddrPages();
extern AddrSpace *CreateAddrSpace();
extern char    *Kmalloc();

/*------------------------------------------------------ External Variables ----*/

extern Thread  *currentThread;
extern Void    *kernelInfoPageAddr;
extern AddrSpace *kernelAddrSpace;
extern char   end;

/*=============================================================================*\
 *  Handle kernel server request for the creation of a user process.
\*=============================================================================*/

MsgBuffer *ReqCreateAddrSpace( inRequest )
      MsgBuffer *inRequest;
{
      register CreateAddrSpaceMsg *request;
      Errcode error;
      AddrSpace *as;
      Thread *th;
      Ulong stack, stacklen;
```

```
    Ulong pageValue;
    MsgBuffer *reply;
    Ulong startPage, virtPage, realPage, pageCount, i;
    Void *startAddr, *virtAddr, *realAddr;

    request = (CreateAddrSpaceMsg *) inRequest;
#   ifdef DEBUG_KERN_SERVER
    StatMsg("KernServ: create: received message at 0x%X\n", request);
#   endif

    /* create addr space and set notification of death port */
    as = CreateAddrSpace( request->requestedPriority );
    as->notificationPortMachineID = request->notificationPort.machineID;
    as->notificationPortPortDesc = request->notificationPort.portDesc;
    as->notificationPortPortCheck = request->notificationPort.portCheck;
#   ifdef DEBUG_KERN_SERVER
    StatMsg("%s: new addr space 0x%X, pagetab=0x%X, priority=%d\n",
            "KernServ: create", as, as->pageTable,
            request->requestedPriority);
#   endif

    /* map in the bootstrapping code.  Only eight pages (4K) is mapped */
    /* so the bootstrapping code cannot be longer than that (without */
    /* modifying this code */
    for( i=0; i<VM_PAGE_LENGTH * 8; i+=VM_PAGE_LENGTH ) {
            MapPage( as, (Void*) (ADDR_USER_BOOTSTRAP_CODE + i),
                    (Void*) (ADDR_BOOTSTRAP_CODE + i),
                    VM_USER_VR_MASK | VM_SHARED_PAGE_MASK );
    }

    /* map in the gosh darned mod table - unutilized but must be present */
    MapPage( as, (Void*) ADDR_USER_MOD_TABLE, (Void*) ADDR_MOD_TABLE,
            VM_USER_VR_MASK | VM_SHARED_PAGE_MASK );

    /* reserve zero page as unaccessible */
    MapPage( as, (Void*) ADDR_USER_ZERO_PAGE, (Void*) ADDR_MOD_TABLE,
        VM_SUPER_VR_MASK | VM_SHARED_PAGE_MASK);

    /* map in the system status page */
    MapPage( as, (Void*) KS_ADDR_INFO_PAGE, kernelInfoPageAddr,
            VM_USER_VR_MASK | VM_SHARED_PAGE_MASK );

    /* create initial thread */
    th = (Thread *) Kmalloc( sizeof(Thread) );
    stacklen = Align( request->initThreadStackLength, VM_PAGE_LENGTH );
    request->initThreadStackLength = stacklen;
    stack = KS_ADDR_ADDR_SPACE_MSG - stacklen;
    startAddr = (Void *) ADDR_USER_BOOTSTRAP_ENTRY;
    InitThreadDesc( th, as, startAddr, AS_REG_USER_PSR,
            request->requestedPriority, stack, stacklen,
            request->argv[0] - KS_ADDR_ADDR_SPACE_MSG + (CPint)request );
    th->reg[AS_REG_MOD] = ADDR_USER_MOD_TABLE;
#   ifdef DEBUG_KERN_SERVER
    StatMsg("KernServ: create: th=0x%X, stack=0x%X, stacklen=%d\n",
            th,stack,stacklen);
#   endif
```

```
	/* map in initial thread stack */
	pageCount = stacklen / VM_PAGE_LENGTH;
	GetVaddrPages( as, (Void*) stack, pageCount, VM_USER_VRW_MASK );

	/* mark creating msg buffer area as reserved */
	RealMemPoke( &(((PageTable *) as->pageTable)->entry
			[KS_ADDR_ADDR_SPACE_MSG / VM_PAGE_LENGTH /
			VM_POINTER_TABLE_ENTRY_COUNT]),
		VM_NULL_RESERVED_ENTRY );

	/* initialize message buffer contents */
	InitAmsg( request, as, th, KS_BOOT_MSG_BUFFER_LENGTH );
	request->initThreadStackPtr = (char *) stack;
#	ifdef DEBUG_KERN_SERVER
	StatMsg("KernServ: create: called InitAmsg\n");
#	endif

	/* create and fill in reply message */
	reply = CreateMsgBuffer( kernelAddrSpace, KS_SMALL_MSG_LENGTH );
#	ifdef DEBUG_KERN_SERVER
	StatMsg("KernServ: create: got reply message buffer at 0x%X\n", reply);
#	endif
	reply->destPort = request->replyPort;
	reply->capability = request->addrSpaceCap;
	reply->requestNumber = request->requestNumber;
	reply->errorCode = ERR_OK;

	/* map in the creation message buffer */
	pageValue = UnmapMsgBuffer( kernelAddrSpace, request );
	MapMsgBuffer( as, KS_ADDR_ADDR_SPACE_MSG, pageValue );

	/* finish up */
	Resume( th );
	return( reply );
}

/*=============================================================================*\
 *  Handle kernel server request for the removal of a user process.
\*=============================================================================*/

void ReqRemoveAddrSpace( request )
	MsgBuffer *request;
{
	Errcode error;
	AddrSpace *as;
	Thread *th;

	as = CheckAddrSpaceCap( request->capability );
#	ifdef DEBUG_KERN_SERVER
	StatMsg("KernServ: RemoveAddrSpace: as=0x%X\n", as);
#	endif
	if( as != NULL ) {
		request->errorCode = RemoveAddrSpace( as );
	} else {
		request->errorCode = ERR_SERVER_BAD_CAPABILITY;
```

```
        }
}

/*-------------------------------------------------------------------------*\
 *  END OF MODULE:  kernservpr.c
\*-------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  lock.c    1.5    92/08/30    22:36:09
 *
 *  This module implements the functions for locking various structures.
 *  These functions should probably have been included in other modules.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <as.h>
#include <queue.h>
#include <sem.h>
#include <port.h>
#include <vm.h>

/*---------------------------------------------------------- Exported Functions ----*/

void  LockAddrSpace();
void  UnlockAddrSpace();
void  LockPortDesc();
void  UnlockPortDesc();
Ulong LockPointerTable();

/*---------------------------------------------------------- External Variables ----*/

extern AddrSpace *kernelAddrSpace;

/*=============================================================================*\
 *  Lock an address space.  Should load it if it is not swapped in.
\*=============================================================================*/

void LockAddrSpace( as )
      AddrSpace *as;
{
      if (!as->swappedIn) {
            Panic ("LockAddrSpace: addr space is swapped out!");
      }
      SemWait( as->lockingSem );
}

/*=============================================================================*\
 *  Unlock an address space.
\*=============================================================================*/

void UnlockAddrSpace( as )
      AddrSpace *as;
{
      SemSignal( as->lockingSem );
}

/*=============================================================================*\
 *  Lock a port descriptor.
\*=============================================================================*/

void LockPortDesc (port)
      PortDesc *port;
{
```

```
        SemWait( port->lockingSem );
}

/*=============================================================================*\
 *  Unlock a port descriptor.
\*=============================================================================*/

void UnlockPortDesc (port)
        PortDesc *port;
{
        SemSignal( port->lockingSem );
}

/*=============================================================================*\
 *  Lock a pointer table.  Makes sure it is loaded into memory and returns
 *  the value in corresponding the page table entry.  The page frame
 *  containing the pointer table is put into "unready" state so it cannot
 *  be paged out.
 *  ASSUMES THE ADDRESS SPACE IS ALREADY LOCKED.
\*=============================================================================*/

Ulong LockPointerTable( as, pageEntry )
        AddrSpace *as;
        Ulong   pageEntry;
{
        PageTable *pageTable;
        Ulong   pageValue;
        Void   *pointerTable;

        pageTable = (PageTable*) as->pageTable;
        LockPageFrameTable();
        pageValue = RealMemPeek( &(pageTable->entry[pageEntry]) );
        if (!(pageValue & VM_PAGE_VALID_MASK) && pageValue != VM_NULL_ENTRY) {
                /* load the stupid thing into memory */
                Kprintf("pageValue=0x%X\n", pageValue);
                Panic("LockPointerTable: pointer table is paged out");
        }
        pointerTable = (Void*) (pageValue & VM_PAGE_MASK);
        if( pageValue != VM_NULL_ENTRY ) {
                UnreadyPageFrame( pointerTable );
        }
        UnlockPageFrameTable();
        return( pageValue );
}

/*=============================================================================*\
 *  Unlock a pointer table.
\*=============================================================================*/

void UnlockPointerTable( realAddr, modified )
        Void   *realAddr;
        Boolean modified;
{
        ReadyPageFrame( realAddr, modified );
}
```

```
/*-------------------------------------------------------------------------*\
 *  END OF MODULE:  lock.c
\*-------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  msgbuf.c    1.6    92/08/30    22:36:21
 *
 *  This module implements all of the operations on message buffers except
 *  message passing.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <as.h>
#include <queue.h>
#include <sem.h>
#include <port.h>
#include <server.h>
#include <vm.h>
#include <pageframe.h>
#include <errcode.h>

/*------------------------------------------------------------ Exported Functions -----*/

MsgBuffer      *CreateMsgBuffer();
void           RemoveMsgBuffer();
MsgBuffer      *FindMsgBufferSpot();
Ulong          UnmapMsgBuffer();
Ulong          MapMsgBuffer();

/*------------------------------------------------------------ Imported Functions ----*/

extern MsgBuffer *FindMsgBufferSpot();
extern Ulong  GetPageCount();
extern Void    *Kmalloc();
extern Void    *GetPageFrame();
extern Ulong  GetRand();
extern AddrSpace *CreateAddrSpace();
extern Ulong  LockPointerTable();

/*------------------------------------------------------------ External Variables -----*/

extern Queue   *addrSpaceList;
extern AddrSpace *kernelAddrSpace;
extern Port   portAliasTable[];
extern AddrSpace *kernelAddrSpace;
extern PageFrameTableEntry pageFrameTable[];

/*=============================================================================*\
 *  Create and map a message buffer into an address space.
\*=============================================================================*/

MsgBuffer *CreateMsgBuffer( as, byteCount )
      AddrSpace *as;
      Ulong      byteCount;
{
      MsgBuffer *msg;
      Ulong pageCount;
      Ulong vpage, rpage, spage;
      Void *vaddr, *raddr;
```

```
        pageCount = GetPageCount( byteCount );
        LockAddrSpace( as );

        /* find a free page table entry */
        msg = FindMsgBufferSpot( as );
        if (msg == (MsgBuffer *) NULL) {
                Panic("CreateMsgBuffer: no free page table entries");
                return( msg );
        }

        /* allocate and map pages at that point */
        GetVaddrPages( as, (Void*) msg, pageCount,
                        (VM_USER_VRW_MASK | VM_MSG_BUFFER_MASK) );
        UnlockAddrSpace( as );
        return( msg );
}

/*=============================================================================*\
 *  Remove a message buffer from an address space.
\*=============================================================================*/

void RemoveMsgBuffer( as, addr )
        AddrSpace *as;
        Void      *addr;
{
        Ulong   pageEntry;

        pageEntry = (CPint)addr / VM_PAGE_LENGTH / VM_POINTER_TABLE_ENTRY_COUNT;
        LockAddrSpace( as );
        FreePointerTable( as, pageEntry );
        UnlockAddrSpace( as );
}

/*=============================================================================*\
 *  Search for a free page table entry to put the message buffer at.
 *  Searches from high addresses to low addresses.
\*=============================================================================*/

MsgBuffer *FindMsgBufferSpot( as )
        AddrSpace *as;
{
        register PageTable *pt;
        register long int pte;

        pt = (PageTable *) as->pageTable;
        for ( pte = VM_PAGE_TABLE_ENTRY_COUNT - 1;
              pte>=0 && RealMemPeek( &(pt->entry[pte]) ) != VM_NULL_ENTRY;
              pte-- ) ;
        if (pte < 0) {
                return( (MsgBuffer *) NULL );
        }
        return( (MsgBuffer *) (pte * VM_POINTER_TABLE_ENTRY_COUNT
                                   * VM_PAGE_LENGTH) );
}

/*=============================================================================*\
```

```
 *  Unmaps a message buffer from an address space.  The value in the page
 *  table entry is returned.
 *  ASSUMES THE ADDRESS SPACE IS ALREADY LOCKED.
\*=============================================================================*/

Ulong UnmapMsgBuffer( as, msg )
      AddrSpace *as;
      MsgBuffer *msg;
{
      PageTable      *pgt;
      PointerTable   *ptt;
      register Ulong  pte;
      Ulong           pageValue;
      register Ulong  pointerValue;
      register Ulong  oldaddr, pageFrame;

      /* locate and lock the pointer table of the message buffer */
      pgt = (PageTable *) as->pageTable;
      pte = (Ulong)msg / VM_POINTER_TABLE_ENTRY_COUNT / VM_PAGE_LENGTH;
      pageValue = LockPointerTable( as, pte );

      /* unmap the message buffer */
      RealMemPoke( &(pgt->entry[pte]), VM_NULL_ENTRY );

      /* update the page frame table entries to be temporarily unpagable */
      LockPageFrameTable();
      pageFrame = (pageValue & VM_PAGE_MASK) / VM_PAGE_LENGTH;
      pageFrameTable[pageFrame].pagable = FALSE;

      ptt = (PointerTable *) (pageValue & VM_PAGE_MASK);
      for (pte=0, oldaddr=(Ulong)msg;
           pte<VM_POINTER_TABLE_ENTRY_COUNT
              && (pointerValue = RealMemPeek( &(ptt->entry[pte]) ))
                    != VM_NULL_ENTRY;
           pte++, oldaddr += VM_PAGE_LENGTH) {

              if (pointerValue & VM_PAGE_VALID_MASK) {
                      pageFrame = (pointerValue & VM_PAGE_MASK)
                                      / VM_PAGE_LENGTH;
                      pageFrameTable[pageFrame].pagable = FALSE;
                      /* invalidate kernel addresses */
                      if (as == kernelAddrSpace) {
                              InvalidateKernelAddr( oldaddr );
                      }
              }
      }
      UnlockPageFrameTable();
      return( pageValue );
}

/*=============================================================================*\
 *  Map a message buffer into an address space.  The number of pages in the
 *  message buffer is returned.
 *  ASSUMES THE ADDRESS SPACE IS ALREADY LOCKED.
\*=============================================================================*/
```

```
Ulong MapMsgBuffer (as, msg, pageValue)
      AddrSpace *as;
      MsgBuffer *msg;
      Ulong pageValue;
{
      PageTable      *pgt;
      PointerTable   *ptt;
      register PageFrameTableEntry *pfte;
      register Ulong  pte;
      register Ulong  pointerValue;
      register Ulong  pageFrame;
      Ulong           newpage;

      /* locate the page table entry and map the message buffer */
      pgt = (PageTable *) as->pageTable;
      pte = (Ulong)msg / VM_POINTER_TABLE_ENTRY_COUNT / VM_PAGE_LENGTH;
      RealMemPoke( &(pgt->entry[pte]), pageValue );

      /* update the page frame table information */
      LockPageFrameTable();
      pageFrame = (pageValue & VM_PAGE_MASK) / VM_PAGE_LENGTH;
      pageFrameTable[pageFrame].pagable = as->swappable;
      pageFrameTable[pageFrame].virtualPage =pte*VM_POINTER_TABLE_ENTRY_COUNT;
      pageFrameTable[pageFrame].addrSpace = as;
      ptt = (PointerTable *) (pageValue & VM_PAGE_MASK);
      for (pte=0, newpage=(Ulong)msg / VM_PAGE_LENGTH;
           pte<VM_POINTER_TABLE_ENTRY_COUNT
              && (pointerValue = RealMemPeek( &(ptt->entry[pte]) ))
                        != VM_NULL_ENTRY;
           pte++, newpage++) {

              if (pointerValue & VM_PAGE_VALID_MASK) {
                    pageFrame =(pointerValue & VM_PAGE_MASK)/VM_PAGE_LENGTH;
                    pfte = &( pageFrameTable[pageFrame] );
                    pfte->pagable = as->swappable;
                    pfte->virtualPage = newpage;
                    pfte->wsDelta = PFT_WSDELTA_START;
                    pfte->addrSpace = as;
              }
      }
      UnlockPageFrameTable();
      return( pte * VM_PAGE_LENGTH );
}

/*-------------------------------------------------------------------------*\
 *  END OF MODULE:  msgbuf.c
\*-------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  msgpass.c    1.6    92/08/30    22:36:35
 *
 *  This routine implements the functions for sending and receiving messages.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <as.h>
#include <queue.h>
#include <sem.h>
#include <port.h>
#include <server.h>
#include <errcode.h>
#include <vm.h>
#include <trap.h>

/*------------------------------------------------------- Exported Functions ----*/

Errcode       KernSend();
Errcode       KernReceive();

/*------------------------------------------------------- Imported Functions ----*/

extern Ulong  UnmapMsgBuffer();
extern Ulong  MapMsgBuffer();
extern Errcode        VerifyPort ();
extern MsgBuffer *FindMsgBufferSpot();
extern void   MakeReady();

/*------------------------------------------------------- External Variables ----*/

extern Port   portAliasTable[];

/*=============================================================================*\
 *  Send a message buffer to a port or to a receiving thread.
\*=============================================================================*/

Errcode KernSend (as, msgBuffer, portName)
      AddrSpace *as;
      MsgBuffer *msgBuffer;
      Port portName;
{
      PortDesc *port;
      Thread *th;
      Errcode error;
      MsgBuffer *receiveMsgBuffer;
      Ulong msgLength;
      Ulong msgSlot;
      Ulong pageValue;
      Ulong alias;
      Boolean intsave;

      /* if the destination port name is an alias, translate it */
      if (portName.machineID == SERV_ALIAS_MACHINE) {
            /*look up proper port in translation table*/
            alias = portName.portDesc;
```

```
        if (alias >= SERV_PORT_ALIAS_COUNT ||
            portAliasTable[alias].machineID == SERV_ALIAS_MACHINE) {
                Panic("KernSend: invalid port alias");
        }
        portName = portAliasTable[portName.portDesc];
}

/* if dest port is not on this machine, panic */
if (portName.machineID != KERN_MACHINE_ID) {
        Kprintf("KernSend( as=0x%X, msg=0x%X, port=[%d 0x%X %d] )\n",
                as, msgBuffer, portName );
        Panic ("Send: message destined for some other machine");
        /* should use the message forwarding server's port name */
}
if (error = VerifyPort(portName)) return( error );

/* grab the message buffer */
port = (PortDesc *) portName.portDesc;
LockPortDesc( port );
LockAddrSpace( as );
pageValue = UnmapMsgBuffer( as, msgBuffer );
if( !(pageValue & VM_MSG_BUFFER_MASK) ) {
Panic("KernSend: Attempt to pass regular pointer table as msg buffer!");
}
UnlockAddrSpace( as );

if (QueueCount( port->waitingThreadList ) > 0) {
        /* give message buffer directly to a waiting thread */
        intsave = DisableInterrupts();
        th = (Thread *) QueueGetHead( port->waitingThreadList );
        th->state = AS_THREAD_STATE_BUSY;
        RestoreInterrupts( intsave );

        /* map in the message buffer */
        LockAddrSpace( th->ownerAddrSpace );
        receiveMsgBuffer = FindMsgBufferSpot( th->ownerAddrSpace );
        msgLength = MapMsgBuffer( th->ownerAddrSpace, receiveMsgBuffer,
                                  pageValue );
        UnlockAddrSpace( th->ownerAddrSpace );

        /* wake up the receiving thread */
        th->reg[SVC_REG_RETURN] = (Ulong) receiveMsgBuffer;
        th->reg[SVC_REG_MSG_LENGTH] = msgLength;
        th->reg[SVC_REG_ERRCODE] = ERR_OK;
        intsave = DisableInterrupts();
        MakeReady( th );
        RestoreInterrupts( intsave );
} else {
        /* move message into port */

        /* get slot to map message buffer in at */
        if (port->waitingMsgCount == 0) {
                msgSlot = port->firstMsgSlot = port->lastMsgSlot = 0;
        } else {
                if( port->waitingMsgCount > VM_PAGE_TABLE_ENTRY_COUNT ){
                        Panic("KernSend: port store addr space full");
```

```
                }
                msgSlot = port->lastMsgSlot = (port->lastMsgSlot+1)
                                        % VM_PAGE_TABLE_ENTRY_COUNT;
        }

        /* map in the message buffer */
        LockAddrSpace( port->storeAddrSpace );
        MapMsgBuffer( port->storeAddrSpace,
                      (MsgBuffer *) (msgSlot * SERV_MAX_MSG_LENGTH),
                      pageValue );
        UnlockAddrSpace( port->storeAddrSpace );
        (port->waitingMsgCount)++;
    }
    UnlockPortDesc( port );
    return( ERR_OK );
}

/*=============================================================================*\
 *  Receive a message buffer from a port.
\*=============================================================================*/

Errcode KernReceive (th, portName, wait, outAddress, outPhysicalLength)
    Thread         *th;
    Port            portName;
    Boolean         wait;
    MsgBuffer     **outAddress;
    Ulong         *outPhysicalLength;
{
    PortDesc *port;
    AddrSpace *as;
    Errcode error;
    MsgBuffer *msgBuffer;
    Ulong msgLength;
    Ulong pageValue;

    /* check port */
    *outAddress = (MsgBuffer *) NULL;
    if (portName.machineID != KERN_MACHINE_ID) {
        return (ERR_MSG_PORT_NOT_OWNED);
    }
    if (error = VerifyPort(portName)) return( error );
    port = (PortDesc *) portName.portDesc;
    LockPortDesc( port );

    /* port must be owned by receiving address space */
    as = th->ownerAddrSpace;
    if (port->ownerAddrSpace != as) {
        UnlockPortDesc( port );
        return( ERR_MSG_PORT_NOT_OWNED );
    }

    if (port->waitingMsgCount > 0) {
        /* message waiting in port - give to thread */

        /* unmap from port address space and into receiver addr space */
        LockAddrSpace( port->storeAddrSpace );
```

```
            pageValue = UnmapMsgBuffer( port->storeAddrSpace,
                (MsgBuffer *)(port->firstMsgSlot * SERV_MAX_MSG_LENGTH));
            UnlockAddrSpace( port->storeAddrSpace );
            LockAddrSpace( th->ownerAddrSpace );
            msgBuffer = FindMsgBufferSpot( th->ownerAddrSpace );
            msgLength = MapMsgBuffer( th->ownerAddrSpace, msgBuffer,
                                      pageValue );
            UnlockAddrSpace( th->ownerAddrSpace );

            /* finish up */
            (port->waitingMsgCount)--;
            port->firstMsgSlot = (port->firstMsgSlot + 1)
                                    % VM_PAGE_TABLE_ENTRY_COUNT;
            *outAddress = msgBuffer;
            *outPhysicalLength = msgLength;
            error = ERR_OK;
        } else {
            /* no messages in port - thread must wait for one */
            if (!wait) return( ERR_MSG_NO_MESSAGE );
            QueueInsert( port->waitingThreadList, (QueueEntry *) th );
            th->portWaitingOn = (Void *) port;
            th->state = AS_THREAD_STATE_WAIT_PORT;
            error = ERR_MSG_RECEIVE_WAIT;
        }
        UnlockPortDesc( port );
        return( error );
}

/*-------------------------------------------------------------------------------*\
 *  END OF MODULE:  msgpass.c
\*-------------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  pageframe.c    1.9    92/08/30    22:36:49
 *
 *  This module implements the functions for maintaining the page frame
 *  table (which is meant to be used for demand disk paging).  The term
 *  "unready" means that the page frame has its "busy" flag set.  The busy
 *  should be set when a page frame is not in the free list nor the LRU list.
 *  When a page frame is "unready", the demand paging system is not meant to
 *  page it out.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <memmap.h>
#include <vm.h>
#include <as.h>
#include <queue.h>
#include <sem.h>
#include <pageframe.h>
#include <debug.h>

/*------------------------------------------------------- Exported Functions ----*/

void  InitPageFrameTable();   /*( kernAddrSpace, kernLimit )*/
Void   *GetPageFrame();              /*( as, virtAddr, pageType ) : realAddr */
void  FreePageFrame();        /*( realAddr )*/
void  ReadyPageFrame();       /*( realAddr, modified )*/
void  UnreadyPageFrame();     /*( realAddr )*/
void  LockPageFrameTable();   /*( )*/
void  UnlockPageFrameTable(); /*( )*/

/*-------------------------------------------------------- Local Functions ----*/

LOCAL void    PrintPftAddr();
LOCAL Void     *GetPageTableFrames();
LOCAL void    SubFreePageFrame();
LOCAL void    InitPageFrame();
LOCAL void    UnlinkPageFrameTableEntry();
LOCAL void    InsertPageFrameTableEntry();

/*------------------------------------------------------ External Variables ----*/

extern PageFrameTableEntry    pageFrameTable [VM_PAGE_COUNT];
extern Ulong                  freePageFrameCount;
extern Ulong                  lruPageFrameCount;
extern Semaphore             *pageFrameSem;
extern Boolean kernelIsReady;

/*=============================================================================*\
 *  Initialize the page frame table with the static kernel pages already
 *  allocated as being unpagable.
\*=============================================================================*/

void InitPageFrameTable( kernelAddrSpace, kernelAddrLimit )
     AddrSpace      *kernelAddrSpace;
     Void           *kernelAddrLimit;
{
```

```
    register PageFrameTableEntry *p;
    register Ulong  i;
    Ulong           kernelPageTableFrame;
    Ulong           kernelPageFrameLimit;

#   ifdef DEBUG
    Kprintf ("Initializing page frame table...\n");
#   endif

    /* reserve page frames that kernel occupies */
    kernelPageFrameLimit = GetPageCount( (CPint)kernelAddrLimit );
    p = &(pageFrameTable[0]);
    for( i=0; i<kernelPageFrameLimit; i++, p++ ) {
            p->pagable      = FALSE;
            p->busy         = TRUE;
            p->type         = PFT_TYPE_REGULAR;
            p->status       = PFT_STATUS_IN_CLEAN;
            p->wsDelta      = PFT_WSDELTA_START;
            p->virtualPage  = i;
            p->addrSpace    = kernelAddrSpace;
            p->diskBlock    = PFT_DISK_BLOCK_NULL;
    }

    /* re-do the page frames for the kernel page table */
    kernelPageTableFrame =
           (CPint)kernelAddrSpace->pageTable / VM_PAGE_LENGTH;
    pageFrameTable[kernelPageTableFrame].type = PFT_TYPE_PAGE_TABLE;
    pageFrameTable[kernelPageTableFrame].virtualPage = 0;
    pageFrameTable[kernelPageTableFrame+1].type = PFT_TYPE_PAGE_TABLE;
    pageFrameTable[kernelPageTableFrame+1].virtualPage = 0;

    /* set rest of page frames as free and linked into free list */
    p = &(pageFrameTable[kernelPageFrameLimit]);
    for( i=kernelPageFrameLimit; i<VM_PAGE_COUNT; i++, p++ ) {
            p->type = PFT_TYPE_FREE;
            p->busy = FALSE;
            p->nextPage = i + 1;
            p->prevPage = i - 1;
    }
    pageFrameTable[kernelPageFrameLimit].prevPage = PFT_MASTER_FREE_FRAME;
    pageFrameTable[VM_PAGE_COUNT-1].nextPage = PFT_MASTER_FREE_FRAME;

    /* initialize free and lru page frame lists */
    pageFrameTable[PFT_MASTER_FREE_FRAME].nextPage = kernelPageFrameLimit;
    pageFrameTable[PFT_MASTER_FREE_FRAME].prevPage = VM_PAGE_COUNT - 1;
    freePageFrameCount = VM_PAGE_COUNT - kernelPageFrameLimit;
    pageFrameTable[PFT_MASTER_LRU_FRAME].nextPage = PFT_MASTER_LRU_FRAME;
    pageFrameTable[PFT_MASTER_LRU_FRAME].prevPage = PFT_MASTER_LRU_FRAME;
    lruPageFrameCount= 0;

#   ifdef DEBUG_INITPFT
    PrintPftAddr ("kernelPageTableFrame", kernelPageTableFrame);
    PrintPftAddr ("kernelPageFrameLimit", kernelPageFrameLimit);
    PrintPftAddr ("pageFrameFreeHead",
            pageFrameTable[PFT_MASTER_FREE_FRAME].nextPage);
    PrintPftAddr ("pageFrameFreeTail",
```

```
                pageFrameTable[PFT_MASTER_FREE_FRAME].prevPage);
#       endif
}

/*=============================================================================*\
 *  Take a free page frame.  Puts it into "unready" state for manipulating.
\*=============================================================================*/

Void *GetPageFrame( as, virtAddr, pageType )
        AddrSpace *as;
        Void    *virtAddr;
        Ulong    pageType;
{
        Ulong p;

        if( pageType == PFT_TYPE_PAGE_TABLE ) {
                return( GetPageTableFrames( as, virtAddr, pageType ) );
        }

        LockPageFrameTable();
        if( freePageFrameCount > 0 ) {
                /* there are free page frames; grab the first of those */
                p = pageFrameTable[PFT_MASTER_FREE_FRAME].nextPage;
                UnlinkPageFrameTableEntry( p, PFT_MASTER_FREE_FRAME,
                        &freePageFrameCount );
        } else {
                /* there are no free page frames; we will have to steal */
                /* one from the LRU list by paging one out */
                Panic("GetPageFrame: no free page frames");
                /* THIS IS WHERE A PAGE FRAME GETS PAGED OUT */
                return( NULL );
        }
        InitPageFrame( p, as, virtAddr, pageType );
        UnlockPageFrameTable();
        (as->memoryPageCount)++;
        return( (Void *) (p * VM_PAGE_LENGTH) );
}

/*=============================================================================*\
 *  Link the given page frame into the free list.  Also handles page table
 *  frame pairs.
 *  ASSUMES THE PAGE FRAME TABLE ENTRY IS ALREADY IN "UNREADY" STATE.
\*=============================================================================*/

void FreePageFrame( realAddr )
        Void *realAddr;
{
        Ulong   p;
        Ulong   diskBlock0, diskBlock1;

        p = (CPint)realAddr / VM_PAGE_LENGTH;
        LockPageFrameTable();
        if( pageFrameTable[p].type == PFT_TYPE_PAGE_TABLE ) {
                SubFreePageFrame( p+1 );
        }
        SubFreePageFrame( p );
```

```
	UnlockPageFrameTable();
}

/*=============================================================================*\
 *  Link a page frame table entry into the LRU list and make it available
 *  for paging.  Changes status to "in dirty" if the modified flag is true.
 *  ASSUMES PAGE FRAME IS IN "UNREADY" STATE.
\*=============================================================================*/

void ReadyPageFrame( realAddr, modified )
	Void	*realAddr;
	Boolean modified;
{
	Ulong p;
	PageFrameTableEntry *pe;

	/* if page frame modified, release disk block and mark dirty */
	p = (CPint) realAddr / VM_PAGE_LENGTH;
	pe = &(pageFrameTable[p]);
	if( modified ) {
		if( pe->diskBlock != PFT_DISK_BLOCK_NULL ) {
			ReleaseVmDiskBlock( pe->diskBlock );
		}
		pe->status = PFT_STATUS_IN_DIRTY;
	}

	LockPageFrameTable();
	if( pe->pagable ) {
		/* only put it into LRU list if it is pagable */
		InsertPageFrameTableEntry( p, PFT_MASTER_LRU_FRAME,
			&lruPageFrameCount);
		pe->busy = FALSE;
	}
	UnlockPageFrameTable();
}

/*=============================================================================*\
 *  Unlink a page frame table entry from the LRU list to for manipulation.
 *  ASSUMES THE PAGE FRAME TABLE IS ALREADY LOCKED.
\*=============================================================================*/

void UnreadyPageFrame( realAddr )
	Void	*realAddr;
{
	Ulong p;
	PageFrameTableEntry *pe;

	p = (CPint)realAddr / VM_PAGE_LENGTH;
	pe = &(pageFrameTable[p]);
	if( !pe->busy ) {
		UnlinkPageFrameTableEntry( p, PFT_MASTER_LRU_FRAME,
			&lruPageFrameCount );
		pe->busy = TRUE;
	}
}
```

```
/*=============================================================================*\
 *  Lock the page frame table.
\*=============================================================================*/

void LockPageFrameTable()
{
      SemWait( pageFrameSem );
}

/*=============================================================================*\
 *  Unlock the page frame table.
\*=============================================================================*/

void UnlockPageFrameTable()
{
      SemSignal( pageFrameSem );
}

/*=============================================================================*\
 *  For debugging, display the address of an important page frame table
 *  entry.
\*=============================================================================*/

#ifdef DEBUG_INITPFT
LOCAL void PrintPftAddr (str, entry)
      char   *str;
      Ulong   entry;
{
      Kprintf ("%s = [%d] = &0x%X for virt addr 0x%X.\n",str,entry,
              &pageFrameTable[entry], entry*VM_PAGE_LENGTH);
}
#endif

/*=============================================================================*\
 *  Reserve two correctly aligned page frame table entries for a page table.
\*=============================================================================*/

LOCAL Void *GetPageTableFrames( as, virtAddr, pageType )
      AddrSpace *as;
      Void   *virtAddr;
      Ulong   pageType;
{
      Ulong p, pOther;
      PageFrameTableEntry *poe;

      LockPageFrameTable();

#     ifdef DEBUG_VERIFY_MUCH
      /* for heavy verification, verify that the free list is correct */
      for(pOther=0,p=pageFrameTable[PFT_MASTER_FREE_FRAME].nextPage;
          p != PFT_MASTER_FREE_FRAME;
          p = pageFrameTable[p].nextPage) {
              pOther++;
      }
      Kprintf("PFT: Count=%d, check=%d\n", freePageFrameCount, pOther);
      if(freePageFrameCount != pOther) Panic("They don't match!");
```

```
#       endif

        /* try to search free list for acceptable pair of page frames */
        if( freePageFrameCount > 0 ) {
                p = pageFrameTable[PFT_MASTER_FREE_FRAME].nextPage;
                while( p != PFT_MASTER_FREE_FRAME ) {
                        /* figure whether the other page of the pair */
                        /* is the next or previous physical one */
                        if( p % 2 == 0 ) {
                                pOther = p+1;
                        } else {
                                pOther = p-1;
                        }
                        poe = &(pageFrameTable[pOther]);
                        if( !poe->busy && poe->type == PFT_TYPE_FREE ) {
                                /* we have found a good pair; get them */
                                UnlinkPageFrameTableEntry( p,
                                        PFT_MASTER_FREE_FRAME,
                                        &freePageFrameCount );
                                UnlinkPageFrameTableEntry( pOther,
                                        PFT_MASTER_FREE_FRAME,
                                        &freePageFrameCount );
                                InitPageFrame( p, as, virtAddr, pageType );
                                InitPageFrame( pOther, as, virtAddr, pageType);
                                if( pOther < p ) p = pOther;
                                UnlockPageFrameTable();
                                (as->memoryPageCount) += 2;
                                return( (Void*) (p * VM_PAGE_LENGTH) );
                        }
                        p = pageFrameTable[p].nextPage;
                }
        }

        /* should search LRU list for page frame pair */
        Panic("GetPageTableFrames: cannot find acceptable pair of frames");

        UnlockPageFrameTable();
}

/*=============================================================================*\
 *  Link the given page frame table entry into the free list.
\*=============================================================================*/

LOCAL void SubFreePageFrame( p )
        Ulong   p;
{
        PageFrameTableEntry *pe;

        pe = &(pageFrameTable[p]);
#       ifdef DEBUG_VERIFY
        if( pe->type == PFT_TYPE_FREE ) {
                Kprintf("SubFreePageFrame: p=%d, raddr=0x%X\n", p,
                        p*VM_PAGE_LENGTH);
                Panic("Attempt to free already free page frame");
        }
        if( !pe->busy ) {
```

```
                Kprintf("SubFreePageFrame: p=%d, raddr=0x%X\n", p,
                        p*VM_PAGE_LENGTH);
                Panic("Attempt to free page frame in LRU list");
        }
#       endif
        (pe->addrSpace->memoryPageCount)--;
        pe->busy = FALSE;
        pe->type = PFT_TYPE_FREE;
        if( pe->diskBlock != PFT_DISK_BLOCK_NULL ) {
                UnlockPageFrameTable();
                ReleaseVmDiskBlock( pe->diskBlock );
                LockPageFrameTable();
        }
        InsertPageFrameTableEntry( p, PFT_MASTER_FREE_FRAME,
                &freePageFrameCount );
}

/*=============================================================================*\
 *  Initialize fields of a page frame table entry.  Called by the
 *  GetPageFrame() functions.  Puts the page frame into "unready" state.
 *  ASSUMES THE PAGE FRAME TABLE IS ALREADY LOCKED.
\*=============================================================================*/

LOCAL void InitPageFrame( p, as, virtAddr, pageType )
        Ulong   p;
        AddrSpace *as;
        Void    *virtAddr;
        Ulong   pageType;
{
        PageFrameTableEntry *pe;

        pe = &(pageFrameTable[p]);
        pe->pagable     = as->swappable;
        pe->busy        = TRUE;
        pe->type        = pageType;
        pe->status      = PFT_STATUS_IN_CLEAN;
        pe->virtualPage = (CPint)virtAddr / VM_PAGE_LENGTH;
        pe->wsDelta     = PFT_WSDELTA_START;
        pe->addrSpace   = as;
        pe->diskBlock   = PFT_DISK_BLOCK_NULL;
}

/*=============================================================================*\
 *  Ulink a page frame table entry from the list that it is currently in.
\*=============================================================================*/

LOCAL void UnlinkPageFrameTableEntry( p, master, countPtr )
        Ulong   p;
        Ulong   master;
        Ulong  *countPtr;
{
        register PageFrameTableEntry *pe;

        pe = &(pageFrameTable[p]);
        pageFrameTable[pe->nextPage].prevPage = pe->prevPage;
        pageFrameTable[pe->prevPage].nextPage = pe->nextPage;
```

```
	(*countPtr)--;
/*	Kprintf("%s: m=%d, p=%d, raddr=0x%X, cnt=%d, vaddr=0x%X\n",
			"UnlinkPageFrameTableEntry", master, p, p*512, *countPtr,
			(pe->virtualPage)*512 );*/
}

/*============================================================================*\
 *  Link a page frame into the tail position of the specified list.
\*============================================================================*/

LOCAL void InsertPageFrameTableEntry( p, master, countPtr )
	Ulong	p;
	Ulong	master;
	Ulong  *countPtr;
{
	register PageFrameTableEntry *mpe, *pe;
	Ulong	oldLast;

	mpe = &(pageFrameTable[master]);
	oldLast = mpe->prevPage;
	pageFrameTable[oldLast].nextPage = p;
	mpe->prevPage = p;
	pe = &(pageFrameTable[p]);
	pe->nextPage = master;
	pe->prevPage = oldLast;
	(*countPtr)++;

/*	Kprintf("%s: m=%d, p=%d, raddr=0x%X, cnt=%d, vaddr=0x%X\n",
			"InsertPageFrameTableEntry", master,p, p*512, *countPtr,
			(pe->virtualPage)*512 );*/
}

/*----------------------------------------------------------------------------*\
 *  END OF MODULE:  pageframe.c
\*----------------------------------------------------------------------------*/
```

```
#/*-------------------------------------------------------------------------*\
# *  MODULE:  panic.s    1.9    92/08/30    22:37:10
# *
# *  This module implements the panic function which is called when a fatal
# *  internal error is detected.  This module is implemented in assembler
# *  so it can preserve the processor registers (as well as possible) for
# *  the return to the ROM monitor.
#\*-------------------------------------------------------------------------*/

#/*---------------------------------------------- Exported Functions -----*/

      .globl  _Panic

#/*---------------------------------------------- Local Declarations -----*/

      .set    MSRMASK,        0xfff8ffff

#/*------------------------------------------------ Static Variables ----*/

      .comm   PANIC_TEMP,4
      .text

#/*=========================================================================*\
# *  Function.
#\*=========================================================================*/

_Panic: #( panicMessage )
      enter   [r0,r1,r2,r3,r4,r5,r6,r7],0
      jsr     _DeviceShutdown         # shut down devices
      jsr     _IcuShutdown
      lprd    intbase, _romMonIntbase # restore the
      movd    8(fp),tos               # get panic message
      movd    4(fp),tos               # get caller's address
      movd    $_PanicTitle,tos        # printf the panic message
      jsr     _Kprintf
      adjspb  $-12
      movqd   0,r0                    # count to 10000 to wait for the
_rep: addqd   1,r0                    #   console to stop printing
      cmpd    r0,$10000
      blt     _rep
      exit    [r0,r1,r2,r3,r4,r5,r6,r7]
      smr     msr, PANIC_TEMP         # turn off virtual memory mapping
      andd    $MSRMASK, PANIC_TEMP
      lmr     msr, PANIC_TEMP
      bpt                             # re-enter the ROM monitor
      ret     0

_PanicTitle:
      .ascii  "PANIC (0x%X): %s\n\0"

#/*-------------------------------------------------------------------------*\
# *  END OF MODULE:  panic.s
#\*-------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  port.c    1.8    92/08/30    22:37:19
 *
 *  This module implements functions for manipulating port control blocks
 *  (except for message passing).
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <as.h>
#include <queue.h>
#include <sem.h>
#include <port.h>
#include <server.h>
#include <vm.h>
#include <pageframe.h>
#include <errcode.h>
#include <trap.h>

/*---------------------------------------------------------- Exported Functions ----*/

void  CreatePort();
void  RemovePort();
void  InitPortAliasTable();
void  SetPortAlias();
Errcode       VerifyPort();

/*---------------------------------------------------------- Imported Functions ----*/

extern MsgBuffer *FindMsgBufferSpot();
extern Ulong  GetPageCount();
extern Void    *Kmalloc();
extern Void    *GetPageFrame();
extern Ulong  GetRand();
extern AddrSpace *CreateAddrSpace();

/*---------------------------------------------------------- External Variables ----*/

extern Queue   *portDescList;
extern Queue   *addrSpaceList;
extern AddrSpace *kernelAddrSpace;
extern Port   portAliasTable[];

/*=============================================================================*\
 *  Dynamically allocate and initialize a port control block.  Returns the
 *  port name.
\*=============================================================================*/

void CreatePort( as, outPortName )
      AddrSpace      *as;
      Port           *outPortName;
{
      PortDesc       *port;
      AddrSpace      *storeAddrSpace;

      port = (PortDesc *) Kmalloc( sizeof(PortDesc) );
      port->priority          = 0;
```

```
        port->portMagicNumber   = PORT_MAGIC_NUMBER;
        port->ownerAddrSpace    = as;
        port->portCheck         = GetRand();
        port->lockingSem        = SemCreate( 1, SEM_PRIORITY_PORT );
        port->waitingThreadList = QueueCreate();
        port->storeAddrSpace    = CreateAddrSpace( port->priority );
        port->waitingMsgCount   = 0;
        port->firstMsgSlot      = 0;
        port->lastMsgSlot       = 0;
        SemWait( as->lockingSem );
        QueueInsert( as->ownedPortList, (QueueEntry *) port );
        SemSignal( as->lockingSem );
        outPortName->machineID = KERN_MACHINE_ID;
        outPortName->portDesc  = (Ulong) port;
        outPortName->portCheck = port->portCheck;
}

/*============================================================================*\
 *  Removes a port control block (and the address space it contains).  This
 *  routine is called as part of removing an address space (which makes the
 *  process recursive).
 *  ASSUMES THAT THE OWNER ADDRESS SPACE IS *NOT* ALREADY LOCKED.
\*============================================================================*/

void RemovePort( port )
        PortDesc *port;
{
        Thread *awaken;
        Boolean intsave;
        AddrSpace *pas;

#       ifdef DEBUG_PORT
        Kprintf("RemovePort: entering for port = 0x%X\n", port);
#       endif
        SemWait( port->lockingSem );

        /* remove port storage address space (recursive call) */
        RemoveAddrSpace( port->storeAddrSpace );

        /* wake up all waiting threads with an error */
        while( QueueCount(port->waitingThreadList) > 0 ) {
                awaken = (Thread*) QueueGetHead( port->waitingThreadList );
                intsave = DisableInterrupts();
                awaken->reg[SVC_REG_ERRCODE] = ERR_MSG_BAD_PORT_DESC;
                MakeReady( awaken );
                RestoreInterrupts( intsave );
        }

        /* finish up */
        QueueRemove( port->waitingThreadList );
        port->portMagicNumber = KERN_UNMAGIC_NUMBER;
        SemSignal( port->lockingSem );
        SemRemove( port->lockingSem );
        Kfree( (Void*) port, sizeof(PortDesc) );
}
```

```
/*=============================================================================*\
 *  Initialize the port alias table to all empty.  The port alias table
 *  translated between aliases and actual port names.  A port alias has a
 *  special machine number, and the "portDesc" field of the port name gives
 *  the index to look up in the port alias table.
\*=============================================================================*/

void InitPortAliasTable()
{
      Ulong i;
      for (i=0; i<SERV_PORT_ALIAS_COUNT; i++) {
            portAliasTable[i].machineID = SERV_ALIAS_MACHINE;
      }
}

/*=============================================================================*\
 *  Sets the given port alias number to the given port name.
\*=============================================================================*/

void SetPortAlias( alias, port )
      Port    port;
      Ulong   alias;
{
      if (alias >= SERV_PORT_ALIAS_COUNT)
            Panic("SetPortAlias: alias out of range");
      if (portAliasTable[alias].machineID != SERV_ALIAS_MACHINE)
            Panic("SetPortAlias: alias already taken");
      portAliasTable[alias] = port;
}

/*=============================================================================*\
 *  Verifies that a port name is valid.
\*=============================================================================*/

Errcode VerifyPort( portName )
      Port portName;
{
      PortDesc *port;

      port = (PortDesc *) portName.portDesc;
      /* if ( !Mapped( (Void *) port, sizeof(PortDesc) ) ) {
            return( ERR_MSG_BAD_PORT_DESC );
      } */

      /* should do a real memory peek here */
      if (port->portMagicNumber != PORT_MAGIC_NUMBER) {
            return( ERR_MSG_BAD_PORT_DESC );
      }
      if (portName.portCheck != port->portCheck) {
            return( ERR_MSG_BAD_PORT_CHECK );
      }
      return( ERR_OK );
}

/*-----------------------------------------------------------------------------*\
 *  END OF MODULE:  port.c
```

----------/

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  queue.c    1.7    92/08/30    22:37:33
 *
 *  Implements all code for creating, destroying, and manipulating round-
 *  robin priority queues.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <queue.h>

/*------------------------------------------------- Exported Functions -----*/

Queue          *QueueCreate();                  /*( ) : queue */
void           QueueRemove();           /*( queue )*/
void           QueueInit();             /*( queue )*/
void           QueueInsert();           /*( queue, queueEnt )*/
void           QueueUnlink();           /*( queue, queueEnt )*/
Ulong          QueueCount();            /*( queue ) : count */
Ulong          QueueHeadPriority();     /*( queue ) : priority */
QueueEntry     *GetQueueHead();               /*( queue ) : queueEnt */
QueueEntry     *QueueHead();            /*( queue ) : queueEnt */

/*------------------------------------------------- Imported Functions -----*/

extern Void    *Kmalloc();
extern void    Kfree();

/*=============================================================================*\
 *  Allocate and initialize a queue structure.
\*=============================================================================*/

Queue *QueueCreate()
{
      Queue *queue;

      queue = (Queue *) Kmalloc( sizeof(Queue) );
      QueueInit( queue );
      return( queue );
}

/*=============================================================================*\
 *  Deallocate a queue structure.
\*=============================================================================*/

void QueueRemove( queue )
      Queue *queue;
{
      queue->queueMagicNumber = KERN_UNMAGIC_NUMBER;
      Kfree( (Void *) queue, sizeof(Queue) );
}

/*=============================================================================*\
 *  Initialize a queue structure to empty.
\*=============================================================================*/

void QueueInit( queue )
```

```
        Queue *queue;
{
        queue->head = (QueueEntry *) queue;
        queue->tail = (QueueEntry *) queue;
        queue->lowpriority = QUEUE_LOWEST_PRIORITY;
        queue->count = 0;
        queue->queueMagicNumber = QUEUE_MAGIC_NUMBER;
}

/*=============================================================================*\
 *  Insert a new entry into a queue, in round-robin priority order.
\*=============================================================================*/

void QueueInsert( queue, entry )
        Queue      *queue;
        QueueEntry *entry;
{
        register QueueEntry *p;
        register Ulong epri;

        epri = entry->priority;
        for (p = queue->head; p->priority <= epri; p = p->next) ;
        entry->next = p;
        entry->prev = p->prev;
        p->prev->next = entry;
        p->prev = entry;
        queue->count++;
}

/*=============================================================================*\
 *  Unlink a queue entry from the queue it is in.
\*=============================================================================*/

void QueueUnlink( queue, entry )
        Queue      *queue;
        QueueEntry *entry;
{
        entry->prev->next = entry->next;
        entry->next->prev = entry->prev;
        queue->count--;
}

/*=============================================================================*\
 *  Return the number of entries in a queue.
\*=============================================================================*/

Ulong QueueCount( queue )
        Queue *queue;
{
        return( queue->count );
}

/*=============================================================================*\
 *  Return the priority of the highest priority entry in a queue (the head
 *  entry) (priority is in reverse numerical order).
\*=============================================================================*/
```

```
Ulong QueueHeadPriority( queue )
      Queue *queue;
{
      return( QueueHead(queue)->priority );
}

/*=============================================================================*\
 *  Unlink and return the head entry of a queue.
\*=============================================================================*/

QueueEntry *QueueGetHead( queue )
      Queue *queue;
{
      QueueEntry *qe;

      qe = QueueHead( queue );
      QueueUnlink( queue, qe );
      return( qe );
}

/*=============================================================================*\
 *  Return pointer to the highest priority entry of a queue.  This routine
 *  implements the queuing discipline.
\*=============================================================================*/

QueueEntry *QueueHead( queue )
      Queue *queue;
{
      return( queue->head );
}

/*-----------------------------------------------------------------------------*\
 *  END OF MODULE:  queue.c
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  ready.c    1.6    92/08/30    22:37:44
 *
 *  This module implements functions for manipulating the ready list and the
 *  current thread.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <as.h>
#include <queue.h>
#include <errcode.h>
#include <debug.h>

/*------------------------------------------------------- Exported Functions ----*/

Errcode       Resume();
Errcode       Suspend();
void  MakeReady();
void  Reschedule();

/*------------------------------------------------------- Imported Functions ----*/

extern Boolean        DisableInterrupts();
extern void   RestoreInterrupts();

/*------------------------------------------------------- External Variables ----*/

extern Thread  *currentThread, *previousThread;
extern Queue   *readyList;

/*=============================================================================*\
 *  Resume the given thread.  The thread must be in suspended state.
\*=============================================================================*/

Errcode Resume( th )
      Thread *th;
{
      Boolean intsave;

      intsave = DisableInterrupts();
      if (th->state != AS_THREAD_STATE_SUSPENDED) {
            return( ERR_TH_RESUME );
      }
      /* check if thread has valid magic number ??? */
      MakeReady (th);
      RestoreInterrupts( intsave );
      return( ERR_OK );
}

/*=============================================================================*\
 *  Suspend the given thread.  The thread must either be current or ready.
 *  A better scheme would have been to use a "suspend count".
\*=============================================================================*/

Errcode Suspend( th )
      Thread *th;
```

```
{
	Boolean intsave;

	/* make sure the thread is current or ready */
	intsave = DisableInterrupts();
	if (th->state != AS_THREAD_STATE_CURRENT &&
	    th->state != AS_THREAD_STATE_READY) {
		return( ERR_TH_SUSPEND );
	}

	/* suspend the thread */
	th->state = AS_THREAD_STATE_SUSPENDED;
	if (th->state == AS_THREAD_STATE_CURRENT) {
		Reschedule();
	} else {
		QueueUnlink( readyList, (QueueEntry *) th );
	}
	RestoreInterrupts( intsave );
	return( ERR_OK );
}

/*=============================================================================*\
 *  Puts the given thread into the ready list and does a reschedule.
 *  ASSUMES THAT INTERRUPTS ARE DISABLED.
\*=============================================================================*/

void MakeReady( th )
	Thread *th;
{
	th->state = AS_THREAD_STATE_READY;
	QueueInsert( readyList, th );
	Reschedule();
}

/*=============================================================================*\
 *  Makes sure the highest priority ready thread is made current.
\*=============================================================================*/

void Reschedule ()
{
	Thread *oldcurr;

#	ifdef DEBUG_VERIFY
	/* if verifying, verify that interrupts are indeed disabled; */
	/* we get into deep trouble if interrupts are not disabled */
	Boolean intsave;
	intsave = DisableInterrupts();
	if( intsave ) {
		Panic("Reschedule: somebody called me with interrupts active!");
	}
#	endif

	/* if the current thread is the highest priority (and is indeed still */
	/* current) then exit immediately */
	previousThread = currentThread;
	if (currentThread->state == AS_THREAD_STATE_CURRENT &&
```

```
        currentThread->priority < QueueHeadPriority(readyList) ) {
            return;
    }

    /* put the current thread into the ready list (if it is still in */
    /* "current" state) */
    if (currentThread->state == AS_THREAD_STATE_CURRENT) {
            currentThread->state = AS_THREAD_STATE_READY;
            QueueInsert( readyList, currentThread );
    }

    /* get the new current thread and do a context switch to it */
    oldcurr = currentThread;
    currentThread = (Thread *) QueueGetHead( readyList );
    currentThread->state = AS_THREAD_STATE_CURRENT;
    ContextSwitch( &(oldcurr->reg[0]) );
}

/*-------------------------------------------------------------------------*\
 *  END OF MODULE:  ready.c
\*-------------------------------------------------------------------------*/
```

```
#/*-----------------------------------------------------------------------*\
# *  MODULE:  realmem.s    1.4    92/08/30    22:37:56
# *
# *  This module implements the assembler functions for accessing physical
# *  memory addresses (for changing memory mapping).
#\*-----------------------------------------------------------------------*/

#/*---------------------------------------------- Exported Functions ----*/

.globl        _RealMemPeek
.globl        _RealMemPoke
.globl        _RealMemFill

#/*---------------------------------------------- Local Declarations -----*/

.set  MAPON,  0x70000
.set  MAPOFF, 0x00000
.set  INTBIT, 0x800
.text

#/*=======================================================================*\
# *  Return the double word value at the given physical memory address
# *  (as opposed to virtual address).
#\*=======================================================================*/

_RealMemPeek: #( address) : value
      enter   [r5,r6,r7],0
      movd    8(fp),r5        # get the address parameter
      sprw    psr, r7         # save the interrupt status
      bicpsrw $INTBIT         # disable interrupts
      smr     msr, r6         # save the memory management status register
      lmr     msr, $MAPOFF    # turn off virtual memory mapping
      movd    0(r5),r0        # get the value from the address
      lmr     msr, r6         # restore virtual memory mapping
      lprw    psr, r7         # restore interrupts
      exit    [r5,r6,r7]
      ret     0               # return the peek value in register 0

#/*=======================================================================*\
# *  Store a double word value at a physical memory address.
#\*=======================================================================*/

_RealMemPoke: #( address, value )
      enter   [r0,r5,r6,r7],0
      movd    8(fp),r5        # get the address parameter
      movd    12(fp),r0       # get the poke value parameter
      sprw    psr, r7         # save the interrupt status
      bicpsrw $INTBIT         # disable interrupts
      smr     msr, r6         # save the memory status register
      lmr     msr, $MAPOFF    # turn virtual memory mapping off
      movd    r0,0(r5)        # poke the value
      lmr     msr, r6         # restore virtual memory mapping
      lprw    psr, r7         # restore interrupts
      exit    [r0,r5,r6,r7]
      ret     0
```

```
#/*=============================================================================*\
# *  Fill physical memory from the given address for the specified number of
# *  double words with the given double word value.
#\*=============================================================================*/

_RealMemFill: #( pageAddr, dwords, value )
      enter   [r0,r4,r5,r6,r7],0
      movd    8(fp),r4                  # get real address parameter
      movd    12(fp),r0                 # get fill count parameter
      movd    16(fp),r5                 # get fill value (double word) parameter
      sprw    psr, r6                   # save interrupt enable status
      bicpsrw $INTBIT                   # disable interrupts
      smr     msr, r7                   # save memory management status register
      lmr     msr, $MAPOFF              # turn off memory mapping

repeat:       movd    r5,0(r4)                  # fill memory
      addqd   $4,r4
      subd    $1,r0
      cmpqd   $0,r0
      bne     repeat

      lmr     msr, r7                   # restore memory mapping
      lprw    psr, r6                   # restore interrupts
      exit    [r0,r4,r5,r6,r7]
      ret     0

#/*-----------------------------------------------------------------------------*\
# *  END OF MODULE:  realmem.s
#\*-----------------------------------------------------------------------------*/
```

```
#/*-----------------------------------------------------------------------*\
# *  MODULE:  regwork.s    1.2    92/08/30    22:38:09
# *
# *  This module implements some assembler functions for accessing the
# *  processor and MMU registers.
#\*-----------------------------------------------------------------------*/

#/*------------------------------------------------ Exported Functions ----*/

.globl        _DisableInterrupts      #( ) : oldEnableFlag
.globl        _RestoreInterrupts      #( enableFlag )
.globl        _InvalidateKernelAddr   #( addr )

.text

#/*=======================================================================*\
# *  Disables interrupts and returns the old interrupt setting in a boolean
# *  variable.
#\*=======================================================================*/

_DisableInterrupts: #( ) : oldEnableFlag
      enter   [],0
      sprd    psr,r0          # get current psr value
      lshd    $-11,r0         # shift the interrupt mask to bit0 position
      andd    $1,r0           # zero the rest of the bits; got return value
      bicpsrw $0x800          # disable interrupts
      exit    []
      ret     0

#/*=======================================================================*\
# *  Sets the interrupts enable bit according to the boolean variable
# *  parameter.  Complement of DisableInterrupts.
#\*=======================================================================*/

_RestoreInterrupts: #( enableFlag )
      enter   [],0
      cmpd    $0,8(fp)                # check "enableFlag"
      beq     _RestoreIntOff
      bispsrw $0x800                  # if TRUE,  enable interrupts
      br      _RestoreIntExit
_RestoreIntOff:
      bicpsrw $0x800                  # if FALSE, disable interrupts
_RestoreIntExit:
      exit    []
      ret     0

#/*=======================================================================*\
# *  Invalidates the given address of the kernel address space by loading the
# *  address into the MMU's EIA register.  The address is invalidated for
# *  the PTB0 address space and also for the PTB1 address space if PTB1 also
# *  points to the kernel address space page table (like it will for kernel
# *  threads).  User address space addresses do not have to be invalidated
# *  since the loading of the page table address into PTB1 on a context
# *  switch will invalidate all cached addresses.
#\*=======================================================================*/
```

```
_InvalidateKernelAddr: #( addr )
      enter   [r0,r1],0
      movd    8(fp),r0                # get the addr parameter
      andd    $0x00ffffff,r0          # extract only the 24-bit address
      lmr     eia,r0                  # invalidate PTB0 address
      smr     ptb1,r1                 # check if PTB1 points to kernel a.s.
      cmpd    r1,$0x8000
      bne     _InvalidateKernelAddrExit
      ord     $0x80000000,r0          # if so, set address for PTB1 and
      lmr     eia,r0                  #   invalidate
_InvalidateKernelAddrExit:
      exit    [r0,r1]
      ret     0

#/*-------------------------------------------------------------------------*\
# *  END OF MODULE:  regwork.s
#\*-------------------------------------------------------------------------*/
```

```
#/*-------------------------------------------------------------------------*\
# *  MODULE:  saveregs.s    1.13    92/08/30    22:38:17
# *
# *  This module implements the functions for register saving and restoring
# *  during context switching.  It is suspected that this module houses the
# *  infamous "0x100 PSR" context switching bug.
#\*-------------------------------------------------------------------------*/

#/*---------------------------------------------------- Exported Functions ----*/

.globl          _ContextSwitch
.globl          _SaveThreadRegs
.globl          _RestoreThreadRegs

#/*----------------------------------------------------- Static Variables -----*/

.comm _userStackRestore,4
.comm _reg0save,4
.comm _psrwork,4

#/*----------------------------------------------------- Local Declarations ----*/

.set SPOFF_PSR,                 10      # offset from SP of interrupted thread's PSR
.set SPOFF_MOD,                 8       # offset from SP of MOD
.set SPOFF_PC,                  4       # offset from SP of PC
.set TRAP_STACK_BIT,    9       # PSR bit that selects SP0/SP1
.set SELECT_SP1,        0x200   # PSR pattern that selects SP0/SP1
.set THOFF_ADDRSP,      6*4     # offset from thread descriptor of addr. space
.set THOFF_USERPTR,     7*4     # offset from TD of "user pointer" (unused)
.set THOFF_REG_SET,     8*4     # offset from TD of register set array
.set ADOFF_PTB1,        4*4     # offset from addr.space descr. of page table
.set REGOFF_R0,                 0*4     # offset from TD register set of R0
.set REGOFF_R1,                 1*4     #   etc.
.set REGOFF_R2,                 2*4
.set REGOFF_R3,                 3*4
.set REGOFF_R4,                 4*4
.set REGOFF_R5,                 5*4
.set REGOFF_R6,                 6*4
.set REGOFF_R7,                 7*4
.set REGOFF_PC,                 8*4
.set REGOFF_PSR,        9*4
.set REGOFF_SP,                 10*4
.set REGOFF_FP,                 11*4
.set REGOFF_MOD,        12*4
.set ADDR_LAST_THREAD_RESTORED,         0xFFFC00        #last thread completely restored
.text

#/*=========================================================================*\
# *  Perform a demand context switch from one thread to another.
#\*=========================================================================*/

_ContextSwitch: #( &oldRegset )
      sprw    psr, _psrwork             #context switch only if persis. thread
      tbitw   $TRAP_STACK_BIT, _psrwork
      bfc     _exitContextSwitch
      sprw    psr, tos                  #save registers as if interrupt occured
```

```
        sprw    mod, tos
        addr    _exitContextSwitch, tos         #desired resume address
        movd    12(sp), r0                      #register save area addr
        movqd   $0,_reg0save                    #old thread return value
        jsr     _SaveThreadRegsGotAddr          #save old thread registers
        addr    8(sp), REGOFF_SP(12(sp))        #desired resume stack pointer
        jsr     _RestoreThreadRegs
        movd    r0, _reg0save
        sprd    sp, r0                  #save old SP1
        lprd    sp, _userStackRestore   #move to new SP1
        bicpsrw $SELECT_SP1             #switch to SP0 (trap stack)
        lprd    sp,$0x8dfc
        movw    6(r0), tos              #put rett info onto SP0
        movw    4(r0), tos
        movd    0(r0), tos
        movd    _reg0save, r0
        movd    _previousTrapMod,_previousTrapModSave
        movd    $0xCB1122CB,_previousTrapMod    #indicate cause of last rett

        tbitw   $9,6(sp)
        bfc     badCtxsw

        rett    0                       #pop rett information from SP0
_exitContextSwitch:
        ret     0

badCtxsw:
        addr    badCtxswMsg,tos
        jsr     _Panic
badCtxswMsg:
        .ascii  "ContextSwitch: cannot resume SP0 thread!\0"

#/*=============================================================================*\
# *  Save the processor registers into the current thread control block
# *  after an interrupt occurs and before processing the interrupt.
#\*=============================================================================*/

_SaveThreadRegs:
        tbitw   $TRAP_STACK_BIT, SP0FF_PSR(sp)  # check if we interrupted an
        bfc     _SaveTrapOnTrap                 #   interrupt
        movd    r0, _reg0save
        movd    _currentThread, r0      # get register set address within
        addd    $THOFF_REG_SET, r0      #   the current thread
_SaveThreadRegsGotAddr:
        movd    _reg0save,REGOFF_R0(r0) # save registers into currentThread
        movd    r1, REGOFF_R1(r0)
        movd    r2, REGOFF_R2(r0)
        movd    r3, REGOFF_R3(r0)
        movd    r4, REGOFF_R4(r0)
        movd    r5, REGOFF_R5(r0)
        movd    r6, REGOFF_R6(r0)
        movd    r7, REGOFF_R7(r0)
        sprd    fp, REGOFF_FP(r0)
        movd    SP0FF_PC(sp), REGOFF_PC(r0)     # take these ones off the
        movw    SP0FF_PSR(sp), REGOFF_PSR(r0)   #   trap stack
        movw    SP0FF_MOD(sp), REGOFF_MOD(r0)
```

```
        sprw    psr, r1                         # select SP1 and save its
        bispsrw $SELECT_SP1                     #   value into the current
        sprd    sp, REGOFF_SP(r0)               #   thread
        lprw    psr, r1
        ret     0

_SaveTrapOnTrap:
        addr    _SaveTotMsg,tos
        jsr     _Panic
_SaveTotMsg:
        .ascii  "SaveRegs: Invalid PSR value on interrupt stack!\0"

#/*=============================================================================*\
# *  Restore the register values in the current thread control block into
# *  the processor registers.
#\*=============================================================================*/

_RestoreThreadRegs:
        movd    _currentThread, r0      #restore registers from currentThread
        addd    $THOFF_REG_SET, r0
        movd    REGOFF_R2(r0), r2
        movd    REGOFF_R3(r0), r3
        movd    REGOFF_R4(r0), r4
        movd    REGOFF_R5(r0), r5
        movd    REGOFF_R6(r0), r6
        lprd    fp, REGOFF_FP(r0)
        movd    REGOFF_PC(r0), SPOFF_PC(sp)     # restore these onto the trap
        movw    REGOFF_PSR(r0), SPOFF_PSR(sp)   #   stack
        movw    REGOFF_MOD(r0), SPOFF_MOD(sp)

        #restore the stack pointer
        sprw    psr, r1                         # check if SP0 is selected
        tbitw   $TRAP_STACK_BIT,r1
        bfs     _RestoreSP1
        bispsrw $SELECT_SP1                     # if so, then select SP1 and
        lprd    sp, REGOFF_SP(r0)               #   restore the value from the
        bicpsrw $SELECT_SP1                     #   current thread into it, and
        br      _RestoreCont                    #   reselect SP0
_RestoreSP1:
        movd    REGOFF_SP(r0),_userStackRestore # otherwise, put into temp. var

_RestoreCont:
        movd    REGOFF_R1(r0), r1       #restore parameter registers
        movd    REGOFF_R7(r0), r7
        movd    REGOFF_R0(r0), _reg0save
        movd    _currentThread, r0      #set PTB1 register
        movd    THOFF_ADDRSP(r0), r0
        lmr     ptb1, ADOFF_PTB1(r0)
        movd    _reg0save, r0
        movd    _currentThread, ADDR_LAST_THREAD_RESTORED

        tbitw   $TRAP_STACK_BIT, SPOFF_PSR(sp)  # check if we interrupted an
        bfc     _RestoreTrapOnTrap              #   interrupt

        ret     0
```

```
_RestoreTrapOnTrap:
      addr    _RestoreTotMsg,tos
      jsr     _Panic
_RestoreTotMsg:
      .ascii  "RestoreRegs: Invalid PSR value on interrupt stack!\0"

#/*-------------------------------------------------------------------------*\
# *  END OF MODULE:  saveregs.s
#\*-------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  sem.c    1.9    92/08/30    22:38:32
 *
 *  This module implements the functions for manipulating counting semaphores.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <as.h>
#include <queue.h>
#include <sem.h>

/*------------------------------------------------------- Exported Functions ----*/

Semaphore *SemCreate();                 /*( count, priority ) : sem */
void  SemInit();                /*( sem, queue, count, semPriority )*/
void  SemRemove();              /*( sem )*/
void  SemWait();                /*( sem )*/
void  SemSignal();              /*( sem )*/
void  SemRestart();             /*( sem, count )*/

/*------------------------------------------------------- Imported Functions ----*/

extern Void    *Kmalloc();     /* kernlib.c */
extern void   Kfree();         /* kernlib.c */

/*------------------------------------------------------- External Variables ----*/

extern Thread  *currentThread;

/*=============================================================================*\
 *  Dynamically allocate and initialize a semaphore.
\*=============================================================================*/

Semaphore *SemCreate( count, semPriority )
      Ulong count;
      Ulong semPriority;
{
      Semaphore *sem;
      Queue *queue;

      sem = (Semaphore *) Kmalloc( sizeof(Semaphore) );
      queue = QueueCreate();
      SemInit( sem, queue, count, semPriority );
      return( sem );
}

/*=============================================================================*\
 *  Initialize an existing semaphore control block.
\*=============================================================================*/

void SemInit( sem, queue, count, semPriority )
      Semaphore *sem;
      Queue *queue;
      Ulong count;
      Ulong semPriority;
{
```

```
        sem->semMagicNumber = SEM_MAGIC_NUMBER;
        sem->count = count;
        sem->semPriority = semPriority;
        sem->waitingThreadList = queue;
}

/*=============================================================================*\
 *  Remove a semaphore control block.
\*=============================================================================*/

void SemRemove( sem )
        Semaphore *sem;
{
        Boolean intsave;
        Thread *awaken;

        intsave = DisableInterrupts();
        while (QueueCount(sem->waitingThreadList) > 0) {
                awaken = (Thread *) QueueGetHead( sem->waitingThreadList );
                MakeReady( awaken );
        }
        QueueRemove( sem->waitingThreadList );
        sem->semMagicNumber = KERN_UNMAGIC_NUMBER;
        RestoreInterrupts( intsave );
        Kfree( (Void *) sem, sizeof(Semaphore) );
}

/*=============================================================================*\
 *  Wait on a semaphore.
\*=============================================================================*/

void SemWait (sem)
        Semaphore *sem;
{
        Boolean intsave;

        intsave = DisableInterrupts();
        if (sem->semPriority != SEM_PRIORITY_NULL &&
            currentThread->semMaxPriority >= sem->semPriority) {
                Kprintf("SemWait: th=%x, sem=%x\n",currentThread,sem);
                Panic("SemWait: attempt to wait on lower priority semaphore");
        }
        currentThread->semMaxPriority != sem->semPriority;
        if (sem->count > 0) {
                (sem->count)--;
        } else {
                currentThread->state = AS_THREAD_STATE_WAIT_SEM;
                currentThread->semWaitingOn = (Void *) sem;
                QueueInsert(sem->waitingThreadList, (QueueEntry*)currentThread);
                Reschedule();
        }
        RestoreInterrupts( intsave );
}

/*=============================================================================*\
 *  Signal a semaphore.
```

```
\*=========================================================================*/

void SemSignal (sem)
      Semaphore *sem;
{
      Thread *awakened;
      Boolean intsave;

      intsave = DisableInterrupts();
      currentThread->semMaxPriority &= ~(sem->semPriority);
      if (QueueCount(sem->waitingThreadList) == 0) {
            (sem->count)++;
      } else {
            awakened = (Thread *) QueueGetHead( sem->waitingThreadList );
            MakeReady( awakened );
      }
      RestoreInterrupts( intsave );
}

/*=========================================================================*\
 *  Restarts a semaphore from a given non-negative count.
 *  All waiting threads are awakened.
\*=========================================================================*/

void SemRestart (sem, value)
      Semaphore *sem;
      Ulong value;
{
      Thread *awakened;
      Boolean intsave;

      intsave = DisableInterrupts();
      while (QueueCount(sem->waitingThreadList) > 0) {
            awakened = (Thread *) QueueGetHead( sem->waitingThreadList );
            MakeReady( awakened );
      }
      sem->count = value;
      RestoreInterrupts( intsave );
}

/*-------------------------------------------------------------------------*\
 *  END OF MODULE:  sem.c
\*-------------------------------------------------------------------------*/
```

```
#/*-----------------------------------------------------------------------*\
# *  MODULE:  startup.s    1.11    92/08/30    22:38:44
# *
# *  UNBOS main function.
#\*-----------------------------------------------------------------------*/

#/*---------------------------------------------------- Exported Functions ----*/

.globl _start

#/*---------------------------------------------------- Local Declarations ----*/

.set TOS,      0x8dfc          # startup/interrupt stack pointer
.set MAPON,    0x70000         # pattern for enabling virtual memory mapping
.set MSRRESET,         0x2A            # pattern for resetting MMU's MSR
.set INITPSR, 0x000            # initial PSR value
.set INTBIT,   0x800           # PSR interrupt bit
.set PTBO,     0x8000          # address of kernel address space page table
.set DISPATCH,         0x8600          # address of dipatch table
.set OFFPSR,    9 * 4          # offset to PSR register in thread control block
.set OFFSP,    10 * 4          #   stack pointer register
.set OFFFP,    11 * 4          #   frame pointer register
.set OFFMOD,   12 * 4          #   module table pointer register
.text

#/*=======================================================================*\
# *  Start.
#\*=======================================================================*/

_start:
      lprw    psr, $INITPSR           # set the initial PSR
      lprd    sb, 0                   # for GCC which assumes SB is 0
      sprd    intbase, _romMonIntbase # save the ROM monitor's intbase
      lprd    fp, $TOS                # use the startup stack
      lprd    sp, $TOS
      movqd   $0, _kernelIsReady      # set flag
      jsr     _InitKernelAddrSpace    # initialize page table
      lmr     ptb0, $PTBO             # enable virtual memory mapping
      lmr     msr, $MSRRESET
      lmr     msr, $MAPON
      jsr     _InitTraps              # initialize interrupts
      lprd    intbase, $DISPATCH      # use the dispatch table
      jsr     _InitKernelThreads      #r0 contains addr of boot thread reg set
      lprw    psr, OFFPSR(r0)         # get boot thread register settings
      lprd    sp, OFFSP(r0)           #   from boot thread control block
      lprd    fp, OFFFP(r0)
      lprw    mod, OFFMOD(r0)
      jsr     _IcuInit                # initialize ICU
     bispsrw $INTBIT                  # enable interrupts
      jsr     _IcuClockStart          # start ICU clock interrupts
      jsr     _InitDevices            # devices before device servers start
      jsr     _InitKernelSystemCalls
      jsr     _StartLocalServers
      jsr     _Banner                 # print the startup banner
      jsr     _Getty                  # start user processes; new main
```

```
        bicpsrw $INTBIT                 # shut down the system:
        jsr     _DeviceShutdown         # shut down interrup sources
        jsr     _IcuShutdown
        lprd    intbase, _romMonIntbase # restore ROM monitor intbase
        lmr     msr, $MSRRESET          # disable memory mapping
        movqd   0,r7                    #delay to let UART finish printing
_wait:  addqd   1,r7
        cmpd    r7,$10000
        blt     _wait
        bpt                             # exit to ROM monitor

#/*-------------------------------------------------------------------------*\
# *  END OF MODULE:  startup.s
#\*-------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  svchandler.c    1.6    92/08/30    22:38:55
 *
 *  This module implements the SVC trap handler and the SVC server.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <as.h>
#include <queue.h>
#include <sem.h>
#include <trap.h>
#include <server.h>
#include <errcode.h>

/*------------------------------------------------------- Exported Functions ----*/

void  SvcTrapProcessor();
void  SvcPickup();

/*------------------------------------------------------- Imported Functions ----*/

extern void   Reschedule();

/*------------------------------------------------------- External Variables ----*/

extern Thread  *currentThread;
extern Queue   *svcWaitingList;
extern Semaphore *svcPickupSem;

/*=============================================================================*\
 *  Put the current thread into the SVC server waiting line.
\*=============================================================================*/

void SvcTrapHandler()
{
      currentThread->state = AS_THREAD_STATE_WAIT_SVC;
      QueueInsert( svcWaitingList, (QueueEntry *) currentThread );
      Reschedule();
      SemSignal( svcPickupSem );
}

/*=============================================================================*\
 *  SVC server: get the next requesting thread and carry out its SendMessage
 *  or ReceiveMessage request.  Parameters are passed in the registers set
 *  of the current thread control block.
\*=============================================================================*/

void SvcPickup()
{
      Thread *th;
      Ulong   i;
      Port    port;
      Boolean wait, intsave;
      MsgBuffer *msgBuffer;
      Errcode err;
      Ulong   length;
```

```
        while( TRUE ) {
                /* wait for a request from a thread */
                SemWait( svcPickupSem );

                /* put the thread into "busy" state */
                intsave = DisableInterrupts();
                th = (Thread *) QueueGetHead( svcWaitingList );
                th->state = AS_THREAD_STATE_BUSY;
                RestoreInterrupts( intsave );

                /* carry out request */
                if (th->reg[SVC_REG_OPERATION] == SVC_OP_SEND) {
                        /* SendMessage: extract parameters & call */
                        port.machineID = th->reg[SVC_REG_PORT_MACHINE];
                        port.portDesc  = th->reg[SVC_REG_PORT_DESC];
                        port.portCheck = th->reg[SVC_REG_PORT_CHECK];
                        msgBuffer = (MsgBuffer *) th->reg[SVC_REG_MSG_BUFFER];
/*                      Kprintf("SvcSEND: MsgBuffer=0x%X Port=[%d 0x%X %d]\n",
                            msgBuffer, port.machineID, port.portDesc,
                            port.portCheck); */
                        err = KernSend(th->ownerAddrSpace, msgBuffer, port);
                        th->reg[SVC_REG_RETURN] = (Ulong) err;
                } else if (th->reg[SVC_REG_OPERATION] == SVC_OP_RECEIVE) {
                        port.machineID = th->reg[SVC_REG_PORT_MACHINE];
                        /* ReceiveMessage: extract parameters & call */
                        port.portDesc  = th->reg[SVC_REG_PORT_DESC];
                        port.portCheck = th->reg[SVC_REG_PORT_CHECK];
                        wait = th->reg[SVC_REG_WAIT_FLAG];
/*                      Kprintf("SvcRECEIVE: Port=[%x %x %x] Wait=%i\n",
                            port.machineID, port.portDesc, port.portCheck,wait);*/
                        err = KernReceive(th, port, wait, &msgBuffer, &length);
                        if (err != ERR_MSG_RECEIVE_WAIT) {
                                /* if the thread did not have to wait on an */
                                /* empty port, fill in the return parameters */
                                th->reg[SVC_REG_RETURN] = (Ulong) msgBuffer;
                                th->reg[SVC_REG_MSG_LENGTH] = length;
                                th->reg[SVC_REG_ERRCODE] = (Ulong) err;
                        }
                } else {
                        /* unknown request code */
                        Kprintf("SvcPickedup UNKNOWN (0x%X)\n",
                                th->reg[SVC_REG_OPERATION] );
                        err = ERR_OK;
                        th->reg[SVC_REG_RETURN] = NULL;
                }

                /* if the requesting thread did not have to wait on a port, */
                /* then awaken it */
                if (err != ERR_MSG_RECEIVE_WAIT) {
                        intsave = DisableInterrupts();
                        MakeReady( th );
                        RestoreInterrupts( intsave );
                }
        }
}
```

```
/*-------------------------------------------------------------------------*\
 *  END OF MODULE:  svchandler.c
\*-------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  thread.c    1.12    92/08/30    22:39:06
 *
 *  This module implements functions for manipulating thread control blocks
 *  and thread stacks.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <as.h>
#include <memmap.h>
#include <queue.h>
#include <sem.h>
#include <port.h>
#include <server.h>
#include <pageframe.h>
#include <errcode.h>
#include <kernserver.h>
#include <debug.h>

/*------------------------------------------------------- Exported Functions ----*/

void  InitThreadDesc();
void  InitThreadStack();
void  GetThreadCap();
void  SetCap();
Thread *CheckThreadCap();
Boolean       RemoveThread();
void  Suicide();
void  SubRemoveThread();

/*------------------------------------------------------- Imported Functions ----*/

extern Ulong  GetRand();
extern Void    *Kmalloc();
extern Void    *GetPageFrame();
extern void   Kfree();
extern MsgBuffer *CreateMsgBuffer();

/*------------------------------------------------------- External Variables ----*/

extern Queue   *addrSpaceList;
extern Port   kernelServerPort;
extern Queue *svcWaitingList;
extern Semaphore *svcPickupSem;
extern Thread *currentThread;
extern AddrSpace *kernelAddrSpace;
extern Queue *addrSpaceList;

/*=============================================================================*\
 *  Initialize an exisiting thread descriptor.
\*=============================================================================*/

void InitThreadDesc (th, as, initPC, initPSR, priority, stackAddr, stackLength,
                     name)
      Thread         *th;
      AddrSpace      *as;
```

```
        Ulong           initPC;
        Ulong           initPSR;
        Ulong           priority;
        char           *stackAddr;
        Ulong           stackLength;
        char            name[];
{
        Ulong           i;

        th->nextThread = NULL;
        th->prevThread = NULL;
        if (priority >= as->priority) {
                th->priority = priority;
        } else {
                th->priority = as->priority;
        }
        th->threadMagicNumber = AS_MAGIC_NUM_THREAD;
        th->state = AS_THREAD_STATE_SUSPENDED;
        th->threadCapabilityCheck = GetRand();
        th->ownerAddrSpace = as;
        th->userPtr = NULL;/*TODO- accept as parameter*/
        if (as->ownedThreadList != NULL) {
                th->nextThreadInAddrSpace = as->ownedThreadList;
        } else {
                th->nextThreadInAddrSpace = (Thread *) NULL;
        }
        as->ownedThreadList = th;
        th->semWaitingOn = NULL;
        th->portWaitingOn = NULL;
        th->semMaxPriority = 0;

        /* set the initial register values */
        for (i=0; i<AS_REG_GPR_COUNT; i++) {
                th->reg[AS_REG_R0+i] = 0;
        }
        th->reg[AS_REG_PC] = initPC;
        th->reg[AS_REG_FP] =
        th->reg[AS_REG_SP] = (Ulong)stackAddr + stackLength;
        th->reg[AS_REG_PSR] = initPSR;
        th->reg[AS_REG_MOD] = ADDR_MOD_TABLE;
        strcpy( th->name, name );
}

/*=============================================================================*\
 *  Initialize a kernel thread's initial stack contents.  It will contain
 *  a message buffer pointer, a port, and a variable number of other
 *  arguments (this routine is used for initializing kernel and device
 *  server threads).
\*=============================================================================*/

Ulong InitStack( base, length, port, msgBuffer, argc, args )
        Ulong   base;
        Ulong   length;
        Port    port;
        MsgBuffer *msgBuffer;
        Ulong   argc;
```

```
	Ulong	args;
{
	Ulong *sp, *inarg;
	Ulong i;
	void (*suptr)();

	sp = (Ulong *) (base + length);

	/* push port and message buffer onto stack */
	*(--sp) = (Ulong) msgBuffer;
	*(--sp) = port.portCheck;
	*(--sp) = port.portDesc;
	*(--sp) = port.machineID;

	/* push initial arguments onto stack and set up frame */
	inarg = (Ulong *) ((Ulong)&args + argc * sizeof(Ulong));
	for (i=0; i<argc; i++) {
		*(--sp) = *(--inarg);
	}

	/* push address of Suicide() as last return address */
	suptr = Suicide;
	*(--sp) = (Ulong) suptr;
	return( (Ulong) sp );
}

/*============================================================================*\
 *  Manufacture a capability for a thread.
\*============================================================================*/

void GetThreadCap( cap, th )
	Capability *cap;
	Thread     *th;
{
	SetCap( cap, kernelServerPort, (Ulong) th, SERV_CAP_RIGHTS_OWNER,
		th->threadCapabilityCheck );
}

/*============================================================================*\
 *  Verify a thread capability.
\*============================================================================*/

Thread *CheckThreadCap( cap )
	Capability cap;
{
	Thread *th;

	th = (Thread *) cap.object;
	if( /*Mapped(th)*/ TRUE ) {
		if( th->threadMagicNumber == AS_MAGIC_NUM_THREAD ) {
			if( cap.check == th->threadCapabilityCheck ) {
				return( th );
			}
		}
	}
	return( (Thread *) NULL );
```

```
}
/*=============================================================================*\
 *  This is the return address pushed onto the stack for a kernel or device
 *  server thread.  This routine should never be called as servers never
 *  exit.
\*=============================================================================*/

void Suicide()
{
      Panic("Suicide: The current thread has attempted to kill itself\n");
}

/*=============================================================================*\
 *  Remove a thread.
\*=============================================================================*/

Boolean RemoveThread( th )
      Thread *th;
{
      AddrSpace *as;

      /* check for thread killing itself */
      if (th == currentThread) {
            Suicide();
            return;
      }

      /* lock the owner address space & remove thread */
      as = th->ownerAddrSpace;
      LockAddrSpace( as );
      if (th->threadMagicNumber != AS_MAGIC_NUM_THREAD) {
            UnlockAddrSpace( as );
            return( ERR_INVALID_OBJECT );
      }
      SubRemoveThread( th );
      UnlockAddrSpace( as );

      /* if we just removed the last thread of the address space, */
      /* then remove that too */
      if( as->ownedThreadList == NULL ) {
            RemoveAddrSpace( as );
      }
}

/*=============================================================================*\
 *  Take the thread to be removed out of whatever state it is in and kill it.
 *  ASSUMES OWNER ADDRESS SPACE IS ALREADY LOCKED.
\*=============================================================================*/

void SubRemoveThread( th )
      Thread *th;
{
      Thread *p;
      AddrSpace *as;
      Boolean intsave;
```

```
#       ifdef DEBUG_THREAD
        StatMsg("SubRemoveThread( th=0x%X )\n", th);
#       endif
        intsave = DisableInterrupts();

        /* get thread out of whatever list it is in into suspended state */
        switch( th->state ) {
        case AS_THREAD_STATE_CURRENT:
        case AS_THREAD_STATE_READY:
                Suspend( th );
                break;
        case AS_THREAD_STATE_WAIT_PORT:
                /* remove from the port it is waiting on */
                QueueUnlink( (Queue *)
                        ((PortDesc *) th->portWaitingOn)->waitingThreadList,
                        (QueueEntry *) th );
                th->state = AS_THREAD_STATE_SUSPENDED;
                break;
        case AS_THREAD_STATE_WAIT_SVC:
                /* remove from svc list */
                QueueUnlink( svcWaitingList, (QueueEntry *) th );
                SemWait( svcPickupSem );
                th->state = AS_THREAD_STATE_SUSPENDED;
                break;
        case AS_THREAD_STATE_WAIT_SEM:
                /* remove from semaphore - cheat: manipulate sem */
                QueueUnlink( (Queue *)
                        ((Semaphore *) th->semWaitingOn)->waitingThreadList,
                        (QueueEntry *) th );
                th->state = AS_THREAD_STATE_SUSPENDED;
                break;
        case AS_THREAD_STATE_BUSY:
                /* wait for state to become un-busy */
                /* - wait-forever possibility if higher server priority */
                /* - should do something to eliminate this state */
                Panic("RemoveThread: not yet able to remove busy thread");
                break;
        case AS_THREAD_STATE_SUSPENDED:
                /* do nothing */
                break;
        }
        RestoreInterrupts( intsave );

        /* remove thread from address space thread list */
        as = th->ownerAddrSpace;
        if( as->ownedThreadList = th ) {
                as->ownedThreadList = th->nextThreadInAddrSpace;
        } else {
                for( p = as->ownedThreadList;
                     p->nextThreadInAddrSpace != th && p != NULL;
                     p = p->nextThreadInAddrSpace ) ;
                if( p == NULL ) {
                        Panic("RemoveThread: th not in owned list");
                }
                p->nextThreadInAddrSpace = th->nextThreadInAddrSpace;
        }
```

```
    /* remove thread descriptor */
    th->threadMagicNumber = KERN_UNMAGIC_NUMBER;
    Kfree( (Void *) th, sizeof(Thread) );
}

/*-----------------------------------------------------------------------------*\
 *  END OF MODULE:  thread.c
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  trap.c    1.8    92/08/30    22:39:26
 *
 *  This module implements functions for initializing trap and interrupts
 *  and handling invalid ones.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <memmap.h>
#include <as.h>
#include <trap.h>
#include <debug.h>

/*------------------------------------------------------- Exported Functions -----*/

void  InitTraps();            /*( )*/
void  DefaultTrapHandler();   /*( trapNumber )*/

/*------------------------------------------------------- Imported Functions -----*/

extern void   DefaultTrapCatcher();
extern void   ClockIntCatcher();
extern void   SvcTrapCatcher();

/*------------------------------------------------------- External Variables -----*/

extern Thread *currentThread;

/*=============================================================================*\
 *  Initialize the dispatch and module tables for interrupts and traps.
\*=============================================================================*/

void InitTraps()
{
      ModTable       *modTable;
      DispatchTable  *dispatchTable;
      Ulong           i;

#     ifdef DEBUG
      Kprintf ("Initializing mod & dispatch tables...\n");
#     endif
      modTable = (ModTable *) ADDR_MOD_TABLE;
      dispatchTable = (DispatchTable *) ADDR_DISPATCH_TABLE;

      /* initialize the dispatch and module tables */
      for (i=0; i < TRAP_TABLE_ENTRY_COUNT; i++) {
              modTable->entry[i].staticBase = TRAP_STATIC_BASE_DEFAULT_VALUE;
              /* set the link base value to the trap number */
              /* so we can tell what trap happened later */
              modTable->entry[i].linkBase = i;
              modTable->entry[i].programBase = (char *) DefaultTrapCatcher;
              modTable->entry[i].reserved = TRAP_RESERVED_DEFAULT_VALUE;

              dispatchTable->entry[i].offset = 0;
              dispatchTable->entry[i].modTablePointer =
                      (unsigned short)(&modTable->entry[i]);
```

```
    }

    /* set the traps and interrupts that the kernel knows how to handle */
    modTable->entry[TRAP_SVC].programBase = (Void *) SvcTrapCatcher;
    modTable->entry[TRAP_CLOCK].programBase = (Void *) ClockIntCatcher;
}

/*=============================================================================*\
 *  High-level routine for handling unknown traps.  Prints debugging
 *  information.
\*=============================================================================*/

void DefaultTrapHandler( trapNum )
    Ulong   trapNum;
{
    Thread *th;

    th = currentThread;
    Kprintf("\nTrap #%d (0x%X) captured by default trap handler.\n",
            trapNum, trapNum);
    Kprintf("CurrentThread=0x%X (\"%s\"), AddrSpace=0x%X (PageTable=0x%X)\n"
            ,th, th->name, th->ownerAddrSpace,
            th->ownerAddrSpace->pageTable );
    Kprintf("thread PC=0x%X, FP=0x%X, SP=0x%X, PSR=0x%X\n",
            th->reg[AS_REG_PC], th->reg[AS_REG_FP], th->reg[AS_REG_SP],
            th->reg[AS_REG_PSR] );
}

/*-----------------------------------------------------------------------------*\
 *  END OF MODULE:  trap.c
\*-----------------------------------------------------------------------------*/
```

```
#/*-------------------------------------------------------------------------*\
# *  MODULE:  trapcatch.s    1.10    92/08/30    22:39:37
# *
# *  This module implements the low-level routines which catch traps and
# *  interrupts for the kernel and then call the higher-level C functions
# *  (if any) after the C environment has been established.
#\*-------------------------------------------------------------------------*/

#/*------------------------------------------------- Exported Functions -----*/

.globl        _ClockIntCatcher
.globl        _SvcTrapCatcher
.globl        _DefaultTrapCatcher

#/*------------------------------------------------- Local Declarations -----*/

.set INTBIT,          0x800   #interrupt enable flag in PSR
.set LINK_BASE_OFF,   4       #link base field offset in mod table entry
.set PSR_STACK_OFF,   6       #offset of PSR on the stack at entry
.text

#/*=========================================================================*\
# *  Clock interrupt catcher.  Backs off if it is interrupting a trap.
#\*=========================================================================*/

_ClockIntCatcher:
      #first check if we should back off
      tbitw   $9,PSR_STACK_OFF(sp)    #check if SP0 was selected before int
      bfs     _ClockIntOk             #if not, then ok
      bicw    $INTBIT,PSR_STACK_OFF(sp)  #clear the interrupt bit for the trap
      reti                            #Acknowledge the interrupt; should find
                                      #  a better way around this
_ClockIntOk:
      #process clock interrupt
      sprd    mod,_previousTrapMod
      jsr     _SaveThreadRegs
      jsr     _Reschedule
      jsr     _RestoreThreadRegs

      movd    4(sp),_previousRetiMP   # get debugging information
      movd    0(sp),_previousRetiPC
      reti

#/*=========================================================================*\
# *  SuperVisor Call trap catcher.
#\*=========================================================================*/

_SvcTrapCatcher:
      bicpsrw $INTBIT                 # interrupting interrupts will back off
      sprd    mod,_previousTrapMod
      addqd   1, 0(sp)                # increment return pc
      jsr     _SaveThreadRegs
      jsr     _SvcTrapHandler
      jsr     _RestoreThreadRegs
      rett    0
```

```
#/*=============================================================================*\
# *  Unexpected trap and interrupt catcher - Panics
#\*=============================================================================*/

_DefaultTrapCatcher:
      bicpsrw $INTBIT                   # interrupting interrupts will back off
      jsr     _SaveThreadRegs
      sprd    mod, r0                   # pass trap number (in link base mod
      movd    LINK_BASE_OFF(r0), tos    # entry) as C parameter
      jsr     _DefaultTrapHandler       # print debugging information
      adjspb  $-4
      jsr     _RestoreThreadRegs
      movd    $_PanicString, tos        # panic
      jsr     _Panic
      rett    0

_PanicString: .ascii "Unable to handle exception!\007\0"

#/*-----------------------------------------------------------------------------*\
# *  END OF MODULE:  trapcatch.s
#\*-----------------------------------------------------------------------------*/
```

```
AR      = /bin/ar
CC      = /usr/bin/scc
CFLAGS          = -I../h
AS      = /bin/as
ASFLAGS         =
H       = ../h

.c.o:
        $(CC) $(CFLAGS) -c $<
.s.o:
        $(AS) $(ASFLAGS) -o $*.o $<

OBJFILES = devglobals.o ttyserver.o ttyinit.o ttyrcv.o ttyxmit.o artinit.o \
        artint.o arthandle.o bootserver.o diskserver.o diskops.o iocc.o \
        ioccint.o setvec.o

devlib.a: $(OBJFILES)
        rm -f devlib.a
        $(AR) cr devlib.a $(OBJFILES)

artint.o: artint.s
ioccint.o: ioccint.s
arthandle.o: arthandle.c $H/kernel.h $H/art.h $H/debug.h
artinit.o: artinit.c $H/kernel.h $H/art.h
bootserver.o: bootserver.c $H/kernel.h $H/server.h $H/fileserver.h \
        $H/errcode.h $H/debug.h
devglobals.o: devglobals.c $H/kernel.h $H/art.h $H/queue.h $H/sem.h $H/tty.h \
        $H/trap.h $H/iocc.h $H/xb1401.h $H/debug.h
diskops.o: diskops.c $H/kernel.h $H/server.h $H/errcode.h $H/iocc.h \
        $H/xb1401.h $H/as.h $H/queue.h $H/sem.h $H/fileserver.h $H/vm.h \
        $H/memmap.h
diskserver.o: diskserver.c $H/kernel.h $H/server.h $H/iocc.h $H/xb1401.h \
        $H/as.h $H/queue.h $H/sem.h $H/diskserver.h $H/errcode.h $H/debug.h
iocc.o: iocc.c $H/kernel.h $H/iocc.h $H/as.h $H/queue.h $H/sem.h $H/debug.h
setvec.o: setvec.c $H/kernel.h $H/memmap.h $H/trap.h
ttyinit.o: ttyinit.c $H/kernel.h $H/art.h $H/queue.h $H/sem.h $H/tty.h
ttyrcv.o: ttyrcv.c $H/kernel.h $H/art.h $H/queue.h $H/sem.h $H/tty.h \
        $H/debug.h
ttyserver.o: ttyserver.c $H/kernel.h $H/server.h $H/art.h $H/queue.h $H/sem.h \
        $H/tty.h $H/errcode.h $H/fileserver.h $H/debug.h
ttyxmit.o: ttyxmit.c $H/kernel.h $H/art.h $H/queue.h $H/sem.h $H/tty.h

clean:
        rm -f *.o core a.out devlib.a
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  %M%    %I%    %E%    %U%
 *
 *  Functions for handling ART interrupts.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <art.h>
#include <debug.h>

/*------------------------------------------------------- Exported Functions ---*/

void  ArtReceiveHandler();
void  ArtTransmitHandler();

/*=============================================================================*\
 *  Handle an ART receive interrupt, given the ART descriptor.
\*=============================================================================*/

void ArtReceiveHandler( art )
      ArtDesc *art;
{
      unsigned char isr;

#     ifdef DEBUG_TTY
      Kprintf("*** ArtReceiveHandler called (art 0x%X) ***\n", art);
#     endif
      isr = art->ioaddr[ARTISR];

      /* determine which channel cause the interrupt and dispatch */
      if( isr & ARTRIENABLEA & art->imrValue ) {
              (art->receiveVectorA) ( art->channelParmA );
      } else if( isr & ARTRIENABLEB & art->imrValue ) {
              (art->receiveVectorB) ( art->channelParmB );
      } else {
              /* we have to panic here rather than merely print a message */
              /* since the interrupt will still be active after exiting */
              /* and we would go into an endless loop */
              Panic("ArtReceiveHandler: unexpected interrupt!\n");
      }
}

/*=============================================================================*\
 *  Handle an ART transmit interrupt, given the ART descriptor.
\*=============================================================================*/

void ArtTransmitHandler( art )
      ArtDesc *art;
{
      unsigned char isr;

      isr = art->ioaddr[ARTISR];

      /* determine which channel cause the interrupt and dispatch */
      if( isr & ARTTIENABLEA & art->imrValue ) {
              (art->transmitVectorA) ( art->channelParmA );
```

```
        } else if( isr & ARTTIENABLEB & art->imrValue ) {
                (art->transmitVectorB) ( art->channelParmB );
        } else {
                Panic("ArtTransmitHandler: unexpected interrupt");
        }
}

/*-----------------------------------------------------------------------------*\
 *  END OF MODULE:  %M%
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  %M%    %I%    %E%    %U%
 *
 *  Functions for initializing ART control blocks and hardware.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <art.h>

/*------------------------------------------------------ Exported Functions ----*/

void  InitArt();
void  InitArtHardware();
void  SetArtVectors();
void  ArtEnable();
void  ArtDisable();
Uchar  *ArtGetChanRegs();

/*------------------------------------------------------ Imported Functions ----*/

extern void   DefaultTrapHandler();
extern void   SetVector();
extern void   ArtReceiveCatcher();
extern void   ArtTransmitCatcher();

/*=============================================================================*\
 *  Initialize an ART control block and hardware.
\*=============================================================================*/

void InitArt( art, ioaddr, receiveVecNum, transmitVecNum, imask )
      ArtDesc *art;
      unsigned char *ioaddr;
      Ulong   receiveVecNum;
      Ulong   transmitVecNum;
{
      /* set the dispatch vectors and parameters */
      art->receiveVectorA  = DefaultTrapHandler;
      art->receiveVectorB  = DefaultTrapHandler;
      art->transmitVectorA = DefaultTrapHandler;
      art->transmitVectorB = DefaultTrapHandler;
      art->channelParmA = 0;
      art->channelParmB = 0;

      /* set the hardware values */
      art->ioaddr = ioaddr;
      art->imrValue = ARTIMRCLEAR;
      InitArtHardware( ioaddr );

      /* set the interrupt vectors and enable the ART */
      SetVector( receiveVecNum, ArtReceiveCatcher, (Ulong) art );
      SetVector( transmitVecNum, ArtTransmitCatcher, (Ulong) art );
      IcuEnable( imask );
}

/*=============================================================================*\
 *  Initialize an ART's hardware.
```

```
\*=============================================================================*/

void InitArtHardware( ioaddr )
      unsigned char *ioaddr;
{
      register unsigned char *art;

      art = ioaddr;

      /* transmit the hardware configuration to the uart. */
      art[ARTIMR]   = ARTIMRCLEAR;
      art[ARTCRA]   = ARTRESETMR;
      art[ARTMRA]   = ARTMODE1;
      art[ARTMRA]   = ARTMODE2;
      art[ARTCSRA]  = ARTSPEED;
      art[ARTCRB]   = ARTRESETMR;
      art[ARTMRB]   = ARTMODE1;
      art[ARTMRB]   = ARTMODE2;
      art[ARTCSRB]  = ARTSPEED;
      art[ARTOPCR]  = ARTOPCONF;
      art[ARTACR]   = ARTAUXVAL;
      art[ARTSOPBC] = ARTSOPBVAL;
      art[ARTCRA]   = ARTTXENABLE | ARTRXENABLE | ARTCLEARERR;
      art[ARTCRB]   = ARTTXENABLE | ARTRXENABLE | ARTCLEARERR;
}

/*=============================================================================*\
 *  Set the dispatch vectors for an ART.  These dispatch routines will
 *  point to TTY interrupt handlers (or other serial line drivers).
\*=============================================================================*/

void SetArtVectors( art, channel, receiveVec, transmitVec, parmValue )
      ArtDesc *art;
      Ulong   channel;
      void    (*receiveVec)();
      void    (*transmitVec)();
      Ulong   parmValue;
{
      if (channel == ARTCHANA) {
            art->receiveVectorA = receiveVec;
            art->transmitVectorA = transmitVec;
            art->channelParmA = parmValue;
      } else {
            art->receiveVectorB = receiveVec;
            art->transmitVectorB = transmitVec;
            art->channelParmB = parmValue;
      }
}

/*=============================================================================*\
 *  Enable transmit and/or receive interrupts for an ART channel.
\*=============================================================================*/

void ArtEnable( art, channel, mask )
      ArtDesc *art;
      Ulong channel;
```

```
	Ulong mask;
{
	/* modify the imrValue pseudo-register as well */
	art->imrValue |= (channel==ARTCHANA ? mask : mask << 4);
	art->ioaddr[ARTIMR] = art->imrValue;
}

/*============================================================================*\
 *  Disable transmit and/or receive interrupts for an ART channel.
\*============================================================================*/

void ArtDisable( art, channel, mask )
	ArtDesc *art;
	Ulong channel;
	Ulong mask;
{
	/* modify the imrValue pseudo-register as well */
	art->imrValue &= ~(channel==ARTCHANA ? mask : mask << 4);
	art->ioaddr[ARTIMR] = art->imrValue;
}

/*============================================================================*\
 *  Return the address of the transmit/receive data register for the given
 *  ART and channel.
\*============================================================================*/

Uchar *ArtGetChanRegs( art, channel )
	ArtDesc *art;
	Ulong   channel;
{
	if (channel == ARTCHANA) {
		return( &(art->ioaddr[ARTMRA]) );
	} else {
		return( &(art->ioaddr[ARTMRB]) );
	}
}

/*----------------------------------------------------------------------------*\
 *  END OF MODULE:  %M%
\*----------------------------------------------------------------------------*/
```

```
#/*-------------------------------------------------------------------------*\
# *  MODULE:  %M%    %I%    %E%    %U%
# *
# *  Assembler functions for catching ART interrupts and calling the C
# *  language interrupt handlers.  This code is written in assembler since
# *  the contents of the stack must be carefully manipulated for saving
# *  and restoring the current thread's registers.
#\*-------------------------------------------------------------------------*/

#/*---------------------------------------------------- Exported Functions ----*/

.globl        _ArtReceiveCatcher
.globl        _ArtTransmitCatcher

#/*---------------------------------------------------- Local Declarations ----*/

.set  PSR_STACK_OFF,  6
.set  LINK_BASE_OFF,  4
.set  INTBIT,         0x800
.text

#/*=========================================================================*\
# *  Catch an ART receive interrupt.  This function will abort if it
# *  detects that it is interrupting a trap, and the ART interrupt will be
# *  caught again after the trap has completed.
#\*=========================================================================*/

_ArtReceiveCatcher:
      #first check if we should back off
      tbitw   $9,PSR_STACK_OFF(sp)    #check if SP0 was selected before int
      bfs     _ArtReceiveCatcherOk    #if not, then ok
      bicw    $INTBIT,PSR_STACK_OFF(sp)  #clear the interrupt bit for the trap
      reti                            #interrupt source will still be active

_ArtReceiveCatcherOk:
      jsr     _SaveThreadRegs
      sprd    mod,_previousTrapMod
      sprd    mod, r0                 #pass link base value of mod entry
      movd    LINK_BASE_OFF(r0), tos  #  - it contains the ART descr addr
      jsr     _ArtReceiveHandler
      adjspb  $-4
      jsr     _RestoreThreadRegs
      reti

#/*=========================================================================*\
# *  Catch an ART transmit interrupt.  Will back off if interrupting a trap.
#\*=========================================================================*/

_ArtTransmitCatcher:
      #first check if we should back off
      tbitw   $9,PSR_STACK_OFF(sp)    #check if SP0 was selected before int
      bfs     _ArtTransmitCatcherOk   #if not, then ok
      bicw    $INTBIT,PSR_STACK_OFF(sp)  #clear the interrupt bit for the trap
      reti                            #interrupt source will still be active

_ArtTransmitCatcherOk:
```

```
        jsr     _SaveThreadRegs
        sprd    mod,_previousTrapMod
        sprd    mod, r0                 #pass link base value of mod entry
        movd    LINK_BASE_OFF(r0), tos  #  - it contains the ART descr addr
        jsr     _ArtTransmitHandler
        adjspb  $-4
        jsr     _RestoreThreadRegs
        movd    4(sp),_previousRetiMP   # get debugging information
        movd    0(sp),_previousRetiPC
        reti

#/*-------------------------------------------------------------------------------------------------*\
# *  END OF MODULE:  %M%
#\*-------------------------------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  %M%    %I%    %E%    %U%
 *
 *  Functions implementing the Boot Disk Server.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <server.h>
#include <fileserver.h>
#include <errcode.h>
#include <debug.h>
#include <memmap.h>

/*------------------------------------------------------- Exported Functions -----*/

void          BootServer();

/*-------------------------------------------------------- Local Functions -----*/

LOCAL void    ReqOpen();
LOCAL void    ReqClose();
LOCAL void    ReqRead();
LOCAL void    ReqDup();

/*------------------------------------------------------- Imported Functions ----*/

extern int strcmp();

/*------------------------------------------------------- Local Declarations -----*/

typedef struct {
      char    filename[16];
      Ulong   startPos;
      Ulong   length;
} DirectoryEntry;

typedef struct {
      Ulong   bootDiskSize;
      Ulong   direntCount;
      DirectoryEntry direntTable[1];  /* variable sized array */
} BootDiskImage;

typedef struct {
      Boolean used;           /* flag */
      Ulong   dupCount;       /* duplicate open count */
      char   *dataPointer;    /* pointer to next byte of data to return */
      Ulong   dataCount;      /* number of data bytes remaining */
} FileControlBlock;

#define BOOT_FCB_COUNT        10
FileControlBlock fcbtable[BOOT_FCB_COUNT];

/*=============================================================================*\
 *  Boot Server main control routine.
\*=============================================================================*/
```

```
void BootServer( port )
    Port port;
{
    MsgBuffer *request;
    Ulong i;
    char *name;

    for (i=0; i<BOOT_FCB_COUNT; i++) {
        fcbtable[i].used = FALSE;
    }
#   ifdef DEBUG
    StatMsg("Boot Disk Server ready, %d FCBs free\n", BOOT_FCB_COUNT);
#   endif
    while( TRUE ) {
        request = Receive( port );
        request->destPort = request->replyPort;
        request->errorCode = ERR_OK;
        switch( request->operationCode ) {
        case FS_OPEN:
            name = (char *)request + request->msgStartOffset;
#           ifdef DEBUG_BOOT_DISK_SERVER
            StatMsg("BootDisk: open( filename=\"%s\" )\n",name);
#           endif
            ReqOpen( request, name );
#           ifdef DEBUG_BOOT_DISK_SERVER
            if (request->errorCode == ERR_OK) {
                StatMsg("BootDisk: open succeeded: FCB=%d\n",
                    request->capability.object);
            } else {
                StatMsg("BootDisk: open FAILED!!!\n");
            }
#           endif
            break;
        case FS_CLOSE:
#           ifdef DEBUG_BOOT_DISK_SERVER
            StatMsg("BootDisk: close( FCB=%d )\n",
                request->capability.object);
#           endif
            ReqClose( request );
            break;
        case FS_READ:
#           ifdef DEBUG_BOOT_DISK_SERVER_MUCH
            StatMsg("BootDisk: read( FCB=%d, bytes=%d )\n",
                request->capability.object, request->msgLength);
#           endif
            ReqRead( request );
            break;
        case FS_DUP:
#           ifdef DEBUG_BOOT_DISK_SERVER
            StatMsg("BootDisk: dup( FCB=%d )\n",
                request->capability.object);
#           endif
            ReqDup( request );
            break;
        case FS_GET_DEVICE_TYPE:
#           ifdef DEBUG_BOOT_DISK_SERVER
```

```
                        StatMsg("BootDisk: GetDevType( FCB=%d )\n",
                                request->capability.object);
#                       endif
                        request->data[0] = FS_DEVICE_TYPE_CHAR;
                        request->data[1] = 0;
                        request->data[2] = 0;
                        break;
                default:
#                       ifdef DEBUG_BOOT_DISK_SERVER
                        StatMsg("BootDisk: unknown request opcode\n");
#                       endif
                        request->errorCode = ERR_SERVER_UNKNOWN_OPCODE;
                        break;
                }
#               ifdef DEBUG_BOOT_DISK_SERVER_MUCH
                StatMsg("BootDisk: operation complete; errcode=%d\n",
                        request->errorCode);
#               endif
                Send( request );
        }
}

/*=============================================================================*\
 *  Handle a request for opening a file.
\*=============================================================================*/

LOCAL void ReqOpen( request, searchFilename )
        MsgBuffer *request;
        char *searchFilename;
{
        Ulong   fcb, i;
        BootDiskImage *im;
        char *bufptr, *fileptr;

        /* find free file control block */
        for (fcb=0; fcb<BOOT_FCB_COUNT && fcbtable[fcb].used; fcb++);
        if (fcb >= BOOT_FCB_COUNT) {
                request->errorCode = ERR_SERVER_TOO_MANY_OPEN_FILES;
                return;
        }
        request->capability.object = fcb;

        /* search for file name */
        im = (BootDiskImage*) ADDR_BOOT_DISK_IMAGE;
        for( i=0;
             i<im->direntCount &&
                 strcmp(searchFilename, im->direntTable[i].filename) != 0;
             i++ ) ;

        /* get start, length parameters */
        if (i<im->direntCount) {
                fcbtable[fcb].used = TRUE;
                fcbtable[fcb].dupCount = 1;
                fcbtable[fcb].dataPointer = (char *) ADDR_BOOT_DISK_IMAGE
                        + im->direntTable[i].startPos;
                fcbtable[fcb].dataCount = im->direntTable[i].length;
```

```
	} else {
		request->errorCode = ERR_SERVER_FILE_NOT_FOUND;
	}
}

/*=============================================================================*\
 *  Handle request to close a file.
\*=============================================================================*/

LOCAL void ReqClose( request )
	MsgBuffer *request;
{
	Ulong fcb;

	fcb = request->capability.object;
	if (fcb >= BOOT_FCB_COUNT) {
		request->errorCode = ERR_SERVER_BAD_CAPABILITY;
	} else {
		/* decrement the dup count and really close only */
		/* if dup count reaches zero */
		if ( (--fcbtable[fcb].dupCount) <= 0 ) {
			fcbtable[fcb].used = FALSE;
		}
	}
}

/*=============================================================================*\
 *  Handle request to duplicate a file descriptor.
\*=============================================================================*/

LOCAL void ReqDup( request )
	MsgBuffer *request;
{
	Ulong fcb;

	fcb = request->capability.object;
	if (fcb >= BOOT_FCB_COUNT) {
		request->errorCode = ERR_SERVER_BAD_CAPABILITY;
	} else {
		/* simply increment the dup count */
		(fcbtable[fcb].dupCount)++;
	}
}

/*=============================================================================*\
 *  Handle request to read an open file's file data.
\*=============================================================================*/

LOCAL void ReqRead( request )
	MsgBuffer *request;
{
	Ulong fcb;
	char *bufPtr;
	char *filePtr;
	Ulong readCount, fileCount, charCount, copyCount;
```

```
        /* get the copying parameters */
        fcb       = request->capability.object;
        filePtr   = fcbtable[fcb].dataPointer;
        bufPtr    = (char *) &(request->data[0]);
        readCount = request->msgLength;
        fileCount = fcbtable[fcb].dataCount;
        request->msgStartOffset = (CPint)bufPtr - (CPint)request;
        if( readCount > request->physicalLength - request->msgStartOffset ) {
                readCount = request->physicalLength - request->msgStartOffset;
        }
        copyCount = (readCount < fileCount) ? readCount : fileCount;

        /* copy the data */
        memcpy( (Void*)bufPtr, (Void*)filePtr, copyCount );

        /* modify the fcb and message buffer data variables */
        fcbtable[fcb].dataPointer = filePtr + copyCount;
        fcbtable[fcb].dataCount -= copyCount;
        request->msgLength = copyCount;
}

/*-------------------------------------------------------------------------------*\
 *  END OF MODULE:  %M%
\*-------------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  %M%    %I%    %E%    %U%
 *
 *  Functions for initializing device hardware and interrupt routines and
 *  declaring global device variables.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <art.h>
#include <queue.h>
#include <sem.h>
#include <tty.h>
#include <trap.h>
#include <iocc.h>
#include <xbi401.h>
#include <debug.h>

/*------------------------------------------------------- Exported Functions ----*/

void          InitDevices();
void          DeviceShutdown();

/*------------------------------------------------------- Local Declarations ----*/

#define  ART_COUNT    2
#define  TTY_COUNT    3
#define  FLOPPY_COUNT 1

/*------------------------------------------------------- Exported Variables ----*/

ArtDesc               artTable [ART_COUNT];
TtyDesc               ttyTable [TTY_COUNT];
Semaphore      *ioccSubchanSem [IOCC_NSC];
IoccCommand    *ioccCmdPtrTable [IOCC_NSC];
IoccCommand   ioccCommand[IOCC_NSC];
XebecAllocMap xebecAllocMap[ FLOPPY_COUNT ];

/*=============================================================================*\
 *  Initialize devices.
\*=============================================================================*/

void InitDevices ()
{
      register int i;
      Boolean intsave;

      intsave = DisableInterrupts();
#     ifdef DEBUG
      Kprintf("Initializing ARTs and TTYs...\n");
#     endif
      for (i=0; i<10000; i++) ; /* wait for ART to finish printing */

      /* initialize ARTs */
      InitArt( &artTable[0], 0xA00020, TRAP_ART0_RECEIVE,
               TRAP_ART0_TRANSMIT, 0x0404 ); /* ports 1 & 2 */
      InitArt( &artTable[1], 0xA00040, TRAP_ART1_RECEIVE,
```

```
                TRAP_ART1_TRANSMIT, 0x1010 ); /* ports 3 & 4 */

        /* initialize TTYs */
        InitTty( &ttyTable[0], &artTable[1], 1 ); /* console = port 4 */
        InitTty( &ttyTable[1], &artTable[1], 0 ); /* term port 3 */
        InitTty( &ttyTable[2], &artTable[0], 1 ); /* term port 2 */

        /* initialize IOCC */
        IoccInit( 0xA000A0, TRAP_IOCC, 0x2000 );
        RestoreInterrupts( intsave );
}

/*============================================================================*\
 *  Shutdown hardware devices.
\*============================================================================*/

void DeviceShutdown()
{
        (&artTable[0])->ioaddr[ARTIMR] = 0;
        (&artTable[1])->ioaddr[ARTIMR] = 0;
}

/*----------------------------------------------------------------------------*\
 *  END OF MODULE:  %M%
\*----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------*\
 *  MODULE:  %M%    %I%    %E%    %U%
 *
 *  Functions for carrying out floppy disk server requests.
\*-----------------------------------------------------------------------*/

#include <kernel.h>
#include <server.h>
#include <errcode.h>
#include <iocc.h>
#include <xbi401.h>
#include <as.h>
#include <queue.h>
#include <sem.h>
#include <fileserver.h>
#include <vm.h>
#include <memmap.h>

/*---------------------------------------------------- Exported Functions ----*/

void  ReqDiskReadOrWrite();
void  ReqDiskAllocate();
void  ReqDiskFree();
Errcode       DiskSync();
Errcode       DiskFormat();
Errcode       DiskMount();

/*---------------------------------------------------- Imported Functions ----*/

extern void   DiskServer();
extern void   InitDrive();
extern void   InitIocb();
extern void   SetTpta();
extern Ulong  ExecCommand();

/*---------------------------------------------------- External Variables ----*/

extern Semaphore *ioccSubchanSem [IOCC_NSC];
extern IoccCommand ioccCommand[IOCC_NSC];

/*---------------------------------------------------- Local Static Variables ----*/

Uchar bitval[8] = { 1, 2, 4, 8, 16, 32, 64, 128 };

/*=======================================================================*\
 *  Handle floppy disk server request for reading or writing blocks.  Read
 *  and WriteBlocks can be combined since the commands are so similar.
\*=======================================================================*/

void ReqDiskReadOrWrite( request, channel, subchannel, operation )
      MsgBuffer *request;
      Uchar  *channel;
      Uchar   subchannel;
      Uchar   operation;
{
      Ulong currentAddr, startBlock;
```

```
        register Ulong blockCount, curBlock, chunkSize, blockIndex;
        Ulong ptable, ptableOffset, pgte, pageValue;
        PageTable *kernPgTable;
        IoccCommand *iocb;

        /* read/write the requested blocks into the given message buffer */
        blockCount = request->data[0];
        blockIndex = 1;
        currentAddr = (Ulong)request + request->msgStartOffset;

        /* set the iocb for the page table and offset */
        pgte = currentAddr / VM_POINTER_TABLE_ENTRY_COUNT / VM_PAGE_LENGTH;
        ptableOffset = currentAddr
                - pgte * VM_POINTER_TABLE_ENTRY_COUNT * VM_PAGE_LENGTH;
        kernPgTable = (PageTable *) ADDR_PTBO_TABLE;
        pageValue = RealMemPeek( &(kernPgTable->entry[pgte]) );
        ptable = (pageValue & VM_PAGE_MASK);
        SetTpta( channel, subchannel, (Ulong) ptable );

        /* read/write the blocks */
        while (blockCount > 0) {
                /* read/write a contiguous-sectors chunk */
                startBlock = curBlock = request->data[blockIndex++];
                chunkSize = 1;
                while (chunkSize < blockCount
                                && request->data[blockIndex] == curBlock+1) {
                        /* increase the chunk size */
                        chunkSize++;
                        curBlock = request->data[blockIndex++];
                }
                /* we must make sure right here that the memory for all */
                /* of the pages to be read/written are paged in */
                ExecCommand( channel, subchannel, operation, startBlock,
                                chunkSize, ptableOffset );
                ptableOffset += chunkSize * XB_BLOCK_LENGTH;
                blockCount -= chunkSize;
        }
}

/*============================================================================*\
 *  Handle floppy disk server request for allocating blocks.  Only the
 *  in-core block allocation map is updated.
\*============================================================================*/

void ReqDiskAllocate( request, blockmap )
        MsgBuffer *request;
        XebecAllocMap *blockmap;
{
        register Ulong getcount, gotcount, block, freeCount, byte, bit;

        getcount = request->data[0];
        gotcount = 0;
        block = blockmap->firstFreeBlock;
        freeCount = blockmap->freeBlockCount;

        /* get the blocks one at a time */
```

```
	while (getcount > 0) {
		if (freeCount == 0) {
			request->errorCode = ERR_DISK_FULL;
			/* should free the gotten blocks here */
			getcount = 0;
		} else {
			freeCount--;

			/* search for a byte in the bitmap with free bit flags*/
			while((blockmap->bitmap[block/8] & bitval[block%8])==0){
				block++;
			}

			/* mark that bit as allocated and get it */
			blockmap->bitmap[block/8] &= ~(bitval[block%8]);
			request->data[++getcount] = block++;
			getcount--;
		}
	}
	blockmap->firstFreeBlock = block;
	blockmap->freeBlockCount = freeCount;
}

/*============================================================================*\
 *  Handle floppy disk server request for freeing blocks.
\*============================================================================*/

void ReqDiskFree( request, blockmap )
	MsgBuffer *request;
	XebecAllocMap *blockmap;
{
	register Ulong block, index, limit;
	register Uchar *bp;

	/* could have it range-check the blocks */
	/* and whether they are really allocated */
	limit = request->data[0];
	/* loop for all of the blocks to free */
	for( index=1; index <= limit; index++ ) {
		/* locate the bit corresponding to the current block */
		block = request->data[index];
		bp = &(blockmap->bitmap[block/8]);
		if (*bp & bitval[block%8]) {
			request->errorCode = ERR_DISK_BLOCK_ALREADY_FREE;
		}
		/* mark the block as free */
		*bp |= bitval[block%8];
		/* update the first block free pointer */
		if (block < blockmap->firstFreeBlock) {
			blockmap->firstFreeBlock = block;
		}
	}
}

/*============================================================================*\
 *  Handle floppy disk server request for synchronizing the block.  Write
```

```
 *  the block allocation map block out to disk.
\*=============================================================================*/

Errcode DiskSync( channel, subchannel, blockmap )
      Uchar  *channel;
      Uchar   subchannel;
      XebecAllocMap *blockmap;
{
      Errcode err;

      SetTpta( channel, subchannel, 0 );
      err = ExecCommand( channel, subchannel, XOWRITE,
                         XB_BITMAP_BLOCK, 1, blockmap);
      return( err );
}

/*=============================================================================*\
 *  Handle floppy disk server request for formatting a disk.  The floppy disk
 *  controller does all the hard work.
\*=============================================================================*/

Errcode DiskFormat( channel, subchannel, blockmap, interleave )
      Uchar  *channel;
      Uchar   subchannel;
      XebecAllocMap *blockmap;
      Ulong interleave;
{
      Errcode err;
      register Ulong i;

      /* physically format the disk */
      err =ExecCommand( channel, subchannel, XOFMTD, 0, interleave, blockmap);
      if (err == ERR_OK) {
              /* fill in and write the block-free bitmap */
              blockmap->magicNumber = XB_MAGIC_NUMBER;
              blockmap->firstFreeBlock = XB_FIRST_BLOCK;
              blockmap->freeBlockCount = XB_BLOCK_COUNT - 1;
              for( i=0; i<XB_BITMAP_LEN; i++ ) {
                      blockmap->bitmap[i] = 0xFF;
              }
              for( i=0; i<XB_FILLER_LEN; i++ ) {
                      blockmap->filler[i] = 0x00;
              }
              blockmap->bitmap[XB_BITMAP_BLOCK/8] &=
                      ~(bitval[XB_BITMAP_BLOCK/8]);
              err = DiskSync( channel, subchannel, blockmap );
      }
      return( err );
}

/*=============================================================================*\
 *  Handle floppy disk server request for mounting a disk.  Read the block
 *  allocation map block into memory.
\*=============================================================================*/

Errcode DiskMount( channel, subchannel, blockmap )
```

```
        Uchar  *channel;
        Uchar   subchannel;
        XebecAllocMap *blockmap;
{
        Errcode err;

        SetTpta( channel, subchannel, 0 );
        err = ExecCommand( channel, subchannel, XOREAD,
                           XB_BITMAP_BLOCK, 1, blockmap);
        if (err == 0 && blockmap->magicNumber != XB_MAGIC_NUMBER) {
                err = 0;        /* put appropriate error code here */
        }
        return( err );
}

/*-----------------------------------------------------------------------------*\
 *  END OF MODULE:  %M%
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  %M%    %I%    %E%    %U%
 *
 *  Contains the floppy disk server main control fuction and some
 *  hardware initialization and utility routines for the floppy disk drive.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <server.h>
#include <iocc.h>
#include <xbi401.h>
#include <as.h>
#include <queue.h>
#include <sem.h>
#include <diskserver.h>
#include <errcode.h>
#include <debug.h>

/*------------------------------------------------------- Exported Functions ----*/

void  DiskServer();
void  InitDrive();
void  InitIocb();
void  SetTpta();
Ulong ExecCommand();

/*------------------------------------------------------- Imported Functions ----*/

extern void   ReqDiskReadOrWrite();
extern void   ReqDiskAllocate();
extern void   ReqDiskFree();
extern Errcode        DiskSync();
extern Errcode        DiskFormat();
extern Errcode        DiskMount();

/*------------------------------------------------------- External Variables ----*/

extern Semaphore *ioccSubchanSem [IOCC_NSC];
extern IoccCommand ioccCommand[IOCC_NSC];

/*------------------------------------------------------- Static Local Variables ----*/

Uchar xebecDriveParms[8] =
      { XFCYL, XFST, XFMT, XFDH, XFSIZE, XFUT, XFSECT, XFDEN };

/*=============================================================================*\
 *  Floppy disk server main control routine.  A "blockmap dirty" flag is
 *  maintained for knowing when the block allocation map needs to be
 *  flushed to disk.
\*=============================================================================*/

void DiskServer( channel, subchannel, target, drive, blockmap, port )
      Uchar *channel;
      Uchar subchannel;
      Uchar target;
      Uchar drive;
```

```
        Port port;
        XebecAllocMap *blockmap;
{
        MsgBuffer *request;
        Boolean blockmapDirty, diskMounted;
        Errcode err;
        Ulong floppy;

        /* initialize the drive */

        floppy = subchannel;
#       ifdef DEBUG_DISK_BLOCK_SERVER
        StatMsg(
        "Floppy#%d Disk Block Server: chan=%x, subch=%d, target=%d, drive=%d\n",
                floppy, channel, subchannel, target, drive);
#       endif
        InitIocb( subchannel, target, drive );
        InitDrive( channel, subchannel );

        diskMounted = FALSE;
        blockmapDirty = FALSE;

#       ifdef DEBUG
        StatMsg("Floppy#%d Disk Block Server ready\n", floppy);
#       endif

        /* start accepting requests */
        while( TRUE ) {
                request = Receive( port );
                request->destPort = request->replyPort;
                request->errorCode = ERR_OK;
                /* make sure the disk is mounted for some operations */
                if ( !diskMounted && request->operationCode != DS_MOUNT_DISK
                     && request->operationCode != DS_FORMAT_DISK ) {
                        request->errorCode = ERR_DISK_NOT_MOUNTED;
#                       ifdef DEBUG
                        StatMsg("Floppy#%d: Disk is not mounted error\n");
#                       endif
                } else {
                switch( request->operationCode ) {
                case DS_READ_BLOCKS:
#                       ifdef DEBUG_DISK_BLOCK_SERVER
                        StatMsg("Floppy#%d: ReadBlocks(count=%d, block1=%d)\n",
                                floppy, request->data[0], request->data[1]);
#                       endif
                        ReqDiskReadOrWrite(request,channel,subchannel,XOREAD);
                        break;
                case DS_WRITE_BLOCKS:
#                       ifdef DEBUG_DISK_BLOCK_SERVER
                        StatMsg("Floppy#%d: WriteBlocks(count=%d, block1=%d)\n",
                                floppy, request->data[0], request->data[1]);
#                       endif
                        ReqDiskReadOrWrite(request,channel,subchannel,XOWRITE);
                        break;
                case DS_ALLOCATE_BLOCKS:
#                       ifdef DEBUG_DISK_BLOCK_SERVER
```

```
                StatMsg("Floppy#%d: %s( count=%d, closeTo=%d )\n",
                        floppy, "AllocateBlocks",request->data[0],
                        request->data[1] );
#               endif
                ReqDiskAllocate( request, blockmap );
                blockmapDirty = TRUE;
                break;
        case DS_FREE_BLOCKS:
#               ifdef DEBUG_DISK_BLOCK_SERVER
                StatMsg("Floppy#%d: FreeBlocks(count=%d, block1=%d)\n",
                        floppy, request->data[0], request->data[1]);
#               endif
                ReqDiskFree( request, blockmap );
                blockmapDirty = TRUE;
                break;
        case DS_MOUNT_DISK:
#               ifdef DEBUG_DISK_BLOCK_SERVER
                StatMsg("Floppy#%d: MountDisk()\n", floppy);
#               endif
                if (diskMounted) {
                        request->errorCode = ERR_DISK_MOUNTED;
                } else {
                        err = DiskMount( channel, subchannel, blockmap);
                        diskMounted = (err == ERR_OK);
                        blockmapDirty = FALSE;
                        request->errorCode = err;
                }
                request->data[0] = XB_BLOCK_LENGTH;
                request->data[1] = XB_FIRST_BLOCK;
                break;
        case DS_UNMOUNT_DISK:
#               ifdef DEBUG_DISK_BLOCK_SERVER
                StatMsg("Floppy#%d: UnmountDisk()\n", floppy);
#               endif
                err = ERR_OK;
                if (blockmapDirty) {
                        err = DiskSync( channel, subchannel, blockmap );
                        blockmapDirty = (err != ERR_OK);
                }
                if (err == ERR_OK) {
                        diskMounted = FALSE;
                }
                request->errorCode = err;
                break;
        case DS_SYNC_DISK:
#               ifdef DEBUG_DISK_BLOCK_SERVER
                StatMsg("Floppy#%d: SyncDisk()\n", floppy);
#               endif
                err = ERR_OK;
                if (blockmapDirty) {
                        err = DiskSync( channel, subchannel, blockmap );
                        blockmapDirty = (err != ERR_OK);
                }
                request->errorCode = err;
                break;
        case DS_FORMAT_DISK:
```

```
#                       ifdef DEBUG_DISK_BLOCK_SERVER
                        StatMsg("Floppy#%d: FormatDisk(interleave=%d)\n",
                                floppy, request->data[0]);
#                       endif
                        if (diskMounted) {
                                request->errorCode = ERR_DISK_MOUNTED;
                        } else {
                                err=DiskFormat( channel, subchannel, blockmap,
                                        request->data[0] );
                                blockmapDirty = FALSE;
                                request->errorCode = err;
                        }
                        break;
                default:
#                       ifdef DEBUG_DISK_BLOCK_SERVER
                        StatMsg("Floppy#%d: Received unknown request opcode\n",
                                floppy);
#                       endif
                        request->errorCode = ERR_SERVER_UNKNOWN_OPCODE;
                        break;
                }
                }
                Send( request );
        }
}

/*=============================================================================*\
 *  Initialize the floppy disk controller hardware and set the disk format
 *  parameters (standard 360K 5-1/4" floppy disk format).
\*=============================================================================*/

void InitDrive( channel, subchannel )
        Uchar *channel;
        Uchar subchannel;
{
#       ifdef DEBUG_DISK_BLOCK_SERVER_MUCH
        Ulong floppy;
        floppy = subchannel;
        StatMsg("Floppy#%d: Initializing subchannel=%i, parms=%x\n",
                floppy, subchannel, xebecDriveParms);
#       endif
        ExecCommand( channel, subchannel, XOINIT, 0, 1, xebecDriveParms );
}

/*=============================================================================*\
 *  Initialize the IOCB that is used for all floppy disk commands for the
 *  server.
\*=============================================================================*/

void InitIocb( subchannel, target, drive )
        Uchar subchannel;
        Uchar target;
        Uchar drive;
{
        IoccCommand *iocb;
```

```
        iocb = &(ioccCommand[subchannel]);
        iocb->ioccode = 0x00;
        iocb->xunit = drive;
        iocb->xcntl = XRETRY | XLOGAD;
        iocb->iocdest = (target << 4) | subchannel;
        iocb->ioctarstat = 0;
        iocb->iocerrstat = 0;
        iocb->iocdata = (char *) 0;
        iocb->ioctpta = (char *) 0;     /* use real addresses */
        iocb->ioclimit = 0;
        iocb->ioclink = (char *) 0;
}

/*============================================================================*\
 *  Set the translation pointer table address (pointer table address when
 *  dealing with message buffers) for the IOCB.
\*============================================================================*/

void SetTpta( channel, subchannel, tpta )
        Uchar *channel;
        Uchar   subchannel;
        Ulong   tpta;
{
        IoccCommand *iocb;

        iocb = &(ioccCommand[subchannel]);
        iocb->ioctpta = (char *) tpta;
}

/*============================================================================*\
 *  Execute a floppy disk controller command.  Uses semaphores for
 *  communication with the IOCC.
\*============================================================================*/

Ulong ExecCommand( channel, subchannel, operation, sector, sectorCount, buf )
        Uchar *channel;
        Uchar subchannel;
        char operation;
        Ulong sector;
        Ulong sectorCount;
        char *buf;
{
        IoccCommand *iocb;
        Ulong errcode;

#       ifdef DEBUG_DISK_BLOCK_SERVER_MUCH
        Ulong floppy;
        floppy = subchannel;
        StatMsg("Floppy#%d: IOCC command: op=0x%X, sector=%d, secCount=%d\n",
                floppy, operation, sector, sectorCount);
#       endif

        /* set the floppy disk controller command parameters in the IOCB */
        iocb = &(ioccCommand[subchannel]);
        iocb->xop = operation;
        iocb->xmaddr = (sector >> 8) & 0xFF;
```

```
        iocb->xladdr = (sector & 0xFF);
        iocb->xcount = sectorCount;
        iocb->iocdata = buf;
        iocb->ioclimit = sectorCount * XB_BLOCK_LENGTH;

        /* wait for the floppy disk channel to be free */
        SemWait( ioccSubchanSem[subchannel] );
        /* execute the command */
        IoccExec( channel, IOCC_SIO + subchannel );
        /* wait for the command to complete */
        SemWait( ioccSubchanSem[subchannel] );
        /* set the channel to free again */
        SemSignal( ioccSubchanSem[subchannel] );

        /* check for errors */
        errcode = iocb->iocerrstat;
        if (errcode == 0) {
                errcode = iocb->ioctarstat;
        }
        /* could translate the error into some unbos code */
        return( errcode );
}

/*-------------------------------------------------------------------------*\
 *  END OF MODULE:  %M%
\*-------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  %M%    %I%    %E%    %U%
 *
 *  Functions for using the I/O Channel Controller.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <iocc.h>
#include <as.h>
#include <queue.h>
#include <sem.h>
#include <debug.h>

/*------------------------------------------------------- Exported Functions ----*/

void  IoccInit();
void  IoccInitHardware();
void  IoccIntHandler();
Uchar IoccStatus();
void  IoccExec();

/*------------------------------------------------------- Imported Functions ----*/

extern void   IoccIntCatcher();

/*------------------------------------------------------- External Variables ----*/

extern Semaphore       *ioccSubchanSem[IOCC_NSC];
extern IoccCommand     *ioccCmdPtrTable[IOCC_NSC];
extern IoccCommand    ioccCommand[IOCC_NSC];

/*=============================================================================*\
 *  Initialize the IOCC hardware and interrupt handling mechanism.
\*=============================================================================*/

void IoccInit( ioaddr, intvecnum, imask )
      char *ioaddr;
      Ulong intvecnum;
      Ulong imask;
{
      Ulong i;

#     ifdef DEBUG
      Kprintf("Initializing IOCC hardware, interrupts, and semaphores\n");
#     endif

      /* initialize the hardware */
      IoccInitHardware( ioaddr );

      /* initialize the interrupt handler */
      SetVector( intvecnum, IoccIntCatcher, (Ulong) ioaddr );
      IcuEnable( imask );

      /* initialize the Sub-channel Ready semaphores and cmd ptrs*/
      for (i=0; i<IOCC_NSC; i++) {
            ioccSubchanSem[i] = SemCreate( 1, SEM_PRIORITY_NULL );
```

```
                ioccCmdPtrTable[i] = &(ioccCommand[i]);
        }
}

/*=============================================================================*\
 *  Initialize the IOCC hardware.  This initialization will cause a
 *  spurrious IOCC interrupt to happen later; ignore it.
\*=============================================================================*/

void IoccInitHardware( ioaddr )
        char *ioaddr;
{
        register Ulong i;
        Ulong cmdtable;

        /* reset I/O channel controller */
        IoccExec( ioaddr, IOCC_RSTIOC );
        for (i=0; ((IoccStatus(ioaddr, IOCC_BT2) & IOCC_SRRST) == 0)
                        && (i<IOCC_BT2); i++) ;
        if (i >= IOCC_BT2) {
                Panic("IoccInitHardware: reset I/O channel timeout");
        }

        /* acknowledge I/O channel controller reset */
        IoccExec( ioaddr, IOCC_RSTA );
        for (i=0; ((IoccStatus(ioaddr, IOCC_BT1) & IOCC_SRRST) != 0)
                        && (i<IOCC_BT1); i++) ;
        if (i >= IOCC_BT1) {
                Panic("IoccInitHardware: acknowledge reset IOCC timeout");
        }

        /* set command pointer table address */
        IoccExec( ioaddr, IOCC_WCTPA );
        for (i=0, cmdtable=(Ulong) ioccCmdPtrTable; i<3; i++, cmdtable>>=8) {
#               ifdef DEBUG_IOCC
                Kprintf("cmdtable = 0x%X / %d\n", cmdtable, cmdtable);
#               endif
                IoccExec( ioaddr, (unsigned char) (cmdtable & 0xFF) );
        }
}

/*=============================================================================*\
 *  Handle an IOCC interrupt.  Signal the subchannel's semaphore.
\*=============================================================================*/

void IoccIntHandler( ioaddr )
        char *ioaddr;
{
        unsigned char status;
        Ulong subch;

        /* determine which subchannel caused the interrupt */
        status = IoccStatus( ioaddr, IOCC_BT1 );
        if ((status & IOCC_SRINT) != 0) {
                /* signal the subchannel semaphore */
                subch = (status & IOCC_SRCHMASK);
```

```
                SemSignal( ioccSubchanSem[subch] );
                IoccExec( ioaddr, IOCC_IA );
        } else {
                Kprintf("IoccIntHandler: sperrious interrupt!\n");
        }
}

/*============================================================================*\
 *  Polled wait for the IOCC to become ready and return the IOCC status code.
\*============================================================================*/

Uchar IoccStatus( ioaddr, maxwait )
        char *ioaddr;
        Ulong maxwait;
{
        register int i;

        for (i=0; ((*ioaddr & IOCC_SRBSY) != 0) && i<maxwait; i++) ;
        if (i>=maxwait) {
                Panic("IoccStatus: busy timeout");
        }
        return( *ioaddr );
}

/*============================================================================*\
 *  Polled wait for the IOCC to become ready and command it to execute a
 *  command code.
\*============================================================================*/

void IoccExec( ioaddr, command )
        char *ioaddr;
        unsigned char command;
{
        register int i;

        for (i=0; ((*ioaddr & IOCC_SRBSY) != 0) && i<IOCC_BTI; i++) ;
        if (i>=IOCC_BTI) {
                Panic("IoccExec: busy timeout");
        }
        *ioaddr = command;
}

/*----------------------------------------------------------------------------*\
 *  END OF MODULE:  %M%
\*----------------------------------------------------------------------------*/
```

```
#/*-------------------------------------------------------------------------*\
# *  MODULE:  %M%    %I%    %E%    %U%
# *
# *  Assembler code to catch an IOCC interrupt and call the higher-level
# *  C code interrupt handler routine.
#\*-------------------------------------------------------------------------*/

#/*---------------------------------------------- Exported Functions -----*/

.globl        _IoccIntCatcher

#/*---------------------------------------------- Local Declarations -----*/

.set  INTBIT,          0x800
.set  LINK_BASE_OFF,   4
.set  PSR_STACK_OFF,   6
.text

#/*=========================================================================*\
# *  Catch an IOCC interrupt; back off if interrupting a trap.
#\*=========================================================================*/

_IoccIntCatcher:
      #first check if we should back off
      tbitw   $9,PSR_STACK_OFF(sp)    #check if SPO was selected before int
      bfs     _IoccIntCatcherOk       #if not, then ok
      bicw    $INTBIT,PSR_STACK_OFF(sp)  #clear the interrupt bit for the trap
      reti                            #interrupt source will still be active

_IoccIntCatcherOk:
      jsr     _SaveThreadRegs
      sprd    mod,_previousTrapMod
      sprd    mod, r0                 #get link base value from mod descriptor
      movd    LINK_BASE_OFF(r0), tos  #  and pass it as the iocc address
      jsr     _IoccIntHandler
      adjspb  $-4
      jsr     _RestoreThreadRegs
      reti

#/*-------------------------------------------------------------------------*\
# *  END OF MODULE:  %M%
#\*-------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  %M%    %I%    %E%    %U%
 *
 *  Function to set the address of an interrupt handler.  This function would
 *  probably have been better put into "trap.c" in the kernel directory.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <memmap.h>
#include <trap.h>

/*------------------------------------------------------ Exported Functions ---*/

void  SetVector();

/*=============================================================================*\
 *  Set the interrupt catcher address for a given interrupt vector number.
 *  Also allows you to set a parameter to be passed to the interrupt hander
 *  code via the link base entry of a module table entry.
\*=============================================================================*/

void SetVector( vecnum, codeaddr, parmValue )
      Ulong   vecnum;
      Void    (*codeaddr)();
      Ulong   parmValue;
{
      ModTable *modtab;

      modtab = (ModTable *) ADDR_MOD_TABLE;
      modtab->entry[vecnum].programBase = (char *) codeaddr;
      /* the link base entry holds the parameter to pass to the ISR */
      modtab->entry[vecnum].linkBase = parmValue;
}

/*-----------------------------------------------------------------------------*\
 *  END OF MODULE:  %M%
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  %M%    %I%    %E%    %U%
 *
 *  Function to initialize a TTY.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <art.h>
#include  <queue.h>
#include <sem.h>
#include <tty.h>

/*------------------------------------------------------- Exported Functions ---*/

void InitTty();

/*------------------------------------------------------- Imported Functions ---*/

extern void   TtyReceive();
extern void   TtyTransmit();
extern unsigned char *ArtGetChanRegs();

/*=============================================================================*\
 *  Initialize a TTY control block.
\*=============================================================================*/

void InitTty( tty, art, channel )
      TtyDesc *tty;
      ArtDesc *art;
      Ulong   channel;
{
      tty->myart = art;
      tty->artChannel = channel;
      tty->emitLinefeed = FALSE;
      tty->outbufEmptySem = SemCreate( 1, SEM_PRIORITY_NULL );
      tty->inbufHead = 0;
      tty->inbufTail = 0;
      tty->inbufCount = 0;
      tty->inbufReadySem = SemCreate( 0, SEM_PRIORITY_NULL );
      tty->echobufHead = 0;    /* wrap-around buffer */
      tty->echobufTail = 0;
      tty->echobufCount = 0;
      tty->cookedMode = TRUE /*TRUE*/;
      tty->flowStopped = FALSE;
      tty->returnEof = FALSE;

      /* set the ART dispatch vectors for the TTY and enable interrupts */
      SetArtVectors( art, channel, TtyReceive, TtyTransmit, (Ulong) tty );
      ArtEnable( art, channel, ARTRIENABLE );
      ArtDisable( art, channel, ARTTIENABLE );
}

/*-----------------------------------------------------------------------------*\
 *  END OF MODULE:  %M%
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  %M%    %I%    %E%    %U%
 *
 *  Functions to implement the TTY receive interrupt handling, including
 *  handling cooked mode input.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <art.h>
#include <queue.h>
#include <sem.h>
#include <tty.h>
#include <debug.h>

/*------------------------------------------------------------ Exported Functions ----*/

void            TtyReceive();

/*------------------------------------------------------------- Local Functions ----*/

LOCAL void      TtyReceiveCooked();
LOCAL void      TtyReceivePut();
LOCAL void      TtyEcho();
LOCAL void      TtyEchoDelete();
LOCAL void      TtyEchoPut();

/*------------------------------------------------------------ Imported Functions ----*/

extern unsigned char *ArtGetChanRegs();

/*=============================================================================*\
 *  TTY Receive interrupt handler (called by the ART interrupt handler).
\*=============================================================================*/

void TtyReceive( tty )
      TtyDesc *tty;
{
      unsigned char c;
      unsigned char *artch;

      /* get the character from the ART */
      artch = ArtGetChanRegs( tty->myart, tty->artChannel );
      c = artch[ARTRHR];
      c &= TTY_CH_MASK;

      /* halt the system on PANIC key */
      if (c == TTY_CH_PANIC) {
            Panic("TtyReceive: user hit the PANIC button");
      }

      /* dispatch to cooked or raw mode handler */
      if (tty->cookedMode) {
            TtyReceiveCooked( tty, c );
      } else {
            TtyReceivePut( tty, c );
            SemSignal( tty->inbufReadySem );
```

```
	}

	/* allow further interrupts if not in flow stoped mode */
	if (!tty->flowStopped) {
		ArtEnable( tty->myart, tty->artChannel, ARTTIENABLE );
	}
}

/*============================================================================*\
 *  Process a received character for cooked mode.
\*============================================================================*/

LOCAL void TtyReceiveCooked( tty, c )
	TtyDesc *tty;
	unsigned char c;
{
	Ulong   prevptr;
	char    prevc;
	Boolean more;

	switch( c ) {
	case TTY_CH_FLOW_STOP:
		ArtDisable( tty->myart, tty->artChannel, ARTTIENABLE );
		tty->flowStopped = TRUE;
		c = TTY_CH_NONE;
		break;
	case TTY_CH_FLOW_RESTART:
		tty->flowStopped = FALSE;
		c = TTY_CH_NONE;
		break;
	case TTY_CH_BACKSPACE:
		prevptr = (tty->inbufTail - 1) % TTY_INBUF_LENGTH;
		prevc = tty->inbuf[prevptr];
		if (tty->inbufCount > 0 && prevc != '\n') {
			tty->inbufTail = prevptr;
			(tty->inbufCount)--;
			TtyEchoDelete( tty, prevc );
		}
		c = TTY_CH_NONE;
		break;
	case TTY_CH_KILL_LINE:
		more = TRUE;
		while (more) {
			/* move back to the beginning of the current line */
			prevptr = (tty->inbufTail - 1) % TTY_INBUF_LENGTH;
			prevc = tty->inbuf[prevptr];
			if (tty->inbufCount>0 && prevc != '\n') {
				tty->inbufTail = prevptr;
				(tty->inbufCount)--;
			} else {
				more = FALSE;
			}
		}
		/* move cursor to a new line */
		TtyEchoPut( tty, TTY_CH_RETURN );
		TtyEchoPut( tty, TTY_CH_LINEFEED );
```

```
                c = TTY_CH_NONE;
                break;
        case TTY_CH_RETURN:
        case TTY_CH_LINEFEED:
                c = '\n';
                break;
        }

        if (c != TTY_CH_NONE) {
                /* put character into receive buffer */
                if (tty->inbufCount >= TTY_INBUF_LENGTH
                   || (tty->inbufCount == TTY_INBUF_LENGTH-1 && c != '\n')) {
                        /* receive buffer full - ring bell */
                        TtyEchoPut( tty, TTY_CH_BELL );
                } else {
                        /* echo character */
                        TtyEcho( tty, c );
                        TtyReceivePut( tty, c );
                        if (c == TTY_CH_EOF) {
                                TtyEcho( tty, '\n' );
                        }
                        /* signal "input line ready" semaphore on end of line */
                        if (c == '\n' || c == TTY_CH_EOF) {
                                SemSignal( tty->inbufReadySem );
                        }
                }
        }
}

/*=============================================================================*\
 *  Put a cooked mode character into the echo buffer.  Control characters
 *  are prefixed with a "^" and new lines are translated to CrLf.
\*=============================================================================*/

LOCAL void TtyEcho( tty, c )
        TtyDesc *tty;
        unsigned char c;
{
        if (c == '\n') {
                TtyEchoPut( tty, TTY_CH_RETURN );
                TtyEchoPut( tty, TTY_CH_LINEFEED );
        } else if (c < TTY_CH_SPACE || c == TTY_CH_RUBOUT) {
                TtyEchoPut( tty, '^' );
                if (c != TTY_CH_RUBOUT) {
                        TtyEchoPut( tty, (c + '@') );
                } else {
                        TtyEchoPut( tty, '?' );
                }
        } else {
                TtyEchoPut( tty, c );
        }
}

/*=============================================================================*\
 *  Put characters into echo buffer to rub out the given cooked mode
 *  character.  Cooked mode characters prefixed with a "^" cause the
```

```
 *  "^" character to be rubbed out too.
\*=============================================================================*/

LOCAL void TtyEchoDelete( tty, c )
        TtyDesc *tty;
        unsigned char c;
{
        Ulong limit, i;

        if (c < TTY_CH_SPACE) {
                limit = 2;
        } else {
                limit = 1;
        }
        for (i=0; i<limit; i++) {
                TtyEchoPut( tty, TTY_CH_BACKSPACE );
                TtyEchoPut( tty, TTY_CH_SPACE );
                TtyEchoPut( tty, TTY_CH_BACKSPACE );
        }
}

/*=============================================================================*\
 *  Physically put a character into the echo buffer.
\*=============================================================================*/

LOCAL void TtyEchoPut( tty, c )
        TtyDesc *tty;
        unsigned char c;
{
        if (tty->echobufCount < TTY_ECHOBUF_LENGTH) {
                tty->echobuf[tty->echobufTail] = c;
                tty->echobufTail = (tty->echobufTail + 1) % TTY_ECHOBUF_LENGTH;
                (tty->echobufCount)++;
        }
}

/*=============================================================================*\
 *  Put a received character into the circular receive buffer.
\*=============================================================================*/

LOCAL void TtyReceivePut( tty, c )
        TtyDesc *tty;
        unsigned char c;
{
        tty->inbuf[tty->inbufTail] = c;
        tty->inbufTail = (tty->inbufTail + 1) % TTY_INBUF_LENGTH;
        (tty->inbufCount)++;
#       ifdef   DEBUG_TTY
        Kprintf("TtyRcv: c=%c, xc=0x%X, head=%d, tail=%d, count=%d\n",c,c,
                tty->inbufHead, tty->inbufTail, tty->inbufCount);
#       endif
}

/*-----------------------------------------------------------------------------*\
 *  END OF MODULE:  %M%
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  %M%    %I%    %E%    %U%
 *
 *  Functions implementing the TTY server proper.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <server.h>
#include <art.h>
#include <queue.h>
#include <sem.h>
#include <tty.h>
#include <errcode.h>
#include <fileserver.h>
#include <debug.h>

/*------------------------------------------------- Exported Functions ----*/

void          TtyServer();

/*-------------------------------------------------- Local Functions ----*/

LOCAL void    ReqWrite();
LOCAL void    ReqRead();
LOCAL void    ReqIoctl();
LOCAL Ulong   TtyGetCooked();
LOCAL char    TtyInbufGet();

/*=============================================================================*\
 *  TTY server main control routine.
\*=============================================================================*/

void TtyServer( tty, port )
      TtyDesc *tty;
      Port port;
{
      MsgBuffer *request;

#     ifdef DEBUG
      Kprintf("Terminal Server ready for TTY=0x%X\n",tty);
#     endif
      while( TRUE ) {
              request = Receive( port );
              request->destPort = request->replyPort;
              request->errorCode = ERR_OK;
              switch (request->operationCode) {
              case FS_READ:
                      ReqRead( tty, request );
                      break;
              case FS_WRITE:
                      ReqWrite( tty, request );
                      break;
              case FS_OPEN:
              case FS_CLOSE:
              case FS_DUP:
                      /* do nothing in these cases */
```

```
                break;
        case FS_IOCTL:
                ReqIoctl( tty, request );
                break;
        case FS_GET_DEVICE_TYPE:
                request->data[0] = FS_DEVICE_TYPE_TTY;
                request->data[1] = 80;
                request->data[2] = 24;
                break;
        default:
                request->errorCode = ERR_SERVER_UNKNOWN_OPCODE;
        }
        Send( request );
    }
}

/*=============================================================================*\
 *  Handle a TTY server write request.
\*=============================================================================*/

LOCAL void ReqWrite( tty, request )
    TtyDesc *tty;
    MsgBuffer *request;
{
    char *cpt;

    /* should load and lock message buffer "request" */
    SemWait( tty->outbufEmptySem );

    /* set data pointer and count to message buffer data */
    cpt = (char *)request + request->msgStartOffset;
    tty->outbufPtr = cpt;
    tty->outbufCount = request->msgLength;
    if (request->msgLength > 0) {
        /* send data and wait for completion */
        if (!tty->flowStopped) {
            ArtEnable( tty->myart, tty->artChannel, ARTTIENABLE );
        }
        SemWait( tty->outbufEmptySem );
    }
    SemSignal( tty->outbufEmptySem );
}

/*=============================================================================*\
 *  Handle TTY server read request.
\*=============================================================================*/

LOCAL void ReqRead( tty, request )
    TtyDesc *tty;
    MsgBuffer *request;
{
    Ulong inLength;
    char c;
    char *msgPtr;

#   ifdef DEBUG_TTY
```

```
        Kprintf("TTY: received READ request...waiting for data...\n");
#       endif
        if (tty->returnEof) {
#               ifdef DEBUG_TTY
                Kprintf("TTY: Picked up EOF and returning 0 length message\n");
#               endif
                tty->returnEof = FALSE;
                request->msgLength = 0;
                return;
        }
        SemWait( tty->inbufReadySem );
#       ifdef DEBUG_TTY
        Kprintf("TTY: input line ready - getting it...\n");
#       endif
        msgPtr = (char *)request + request->msgStartOffset;
        if (tty->cookedMode) {
                inLength = TtyGetCooked( tty, msgPtr, request->msgLength );
        } else {
                c = TtyInbufGet( tty );
                *(msgPtr) = c;
                inLength = 1;
        }
        request->msgLength = inLength;
}

/*=============================================================================*\
 *  Get a character from the TTY receive buffer for cooked mode.  Handle
 *  end of file.
\*=============================================================================*/

LOCAL Ulong TtyGetCooked( tty, msgPtr, maxlen )
        TtyDesc *tty;
        char *msgPtr;
        Ulong maxlen;
{
        Ulong inLength;
        char c;

        inLength = 0;
        do {
                c = TtyInbufGet( tty );
                *(msgPtr++) = c;
                inLength++;
        } while( inLength < maxlen && c != '\n' && c != TTY_CH_EOF );
        if (c == TTY_CH_EOF) {
#               ifdef DEBUG_TTY
                Kprintf("TTY: Got EOF, adjusting message\n");
#               endif
                inLength--;
                if (inLength != 0) {
#                       ifdef DEBUG_TTY
                        Kprintf("TTY: Returned msg before EOF\n");
#                       endif
                        tty->returnEof = TRUE;
                }
        }
```

```
    return( inLength );
}

/*=============================================================================*\
 *  Raw get a character from the TTY receive buffer.
\*=============================================================================*/

LOCAL char TtyInbufGet( tty )
    TtyDesc *tty;
{
    char c;

    c = tty->inbuf[ tty->inbufHead ];
    (tty->inbufCount)--;    /* indivisible operation */
    tty->inbufHead = (tty->inbufHead + 1) % TTY_INBUF_LENGTH;
    return( c );
}

/*=============================================================================*\
 *  Handle a TTY server ioctl request.  Only SetMode is implemented.
\*=============================================================================*/

LOCAL void ReqIoctl( tty, request )
    TtyDesc *tty;
    MsgBuffer *request;
{
    Ulong newmode;
    Boolean intsave;

    switch( request->data[0] ) {
    case TTY_IOCTL_SET_MODE:
        newmode = request->data[0];
        request->data[0] = tty->cookedMode;
        if (newmode != TTY_MODE_SAME) {
            intsave = DisableInterrupts();
            /* change the TTY mode, clear received characters */
            tty->cookedMode = newmode;
            tty->inbufCount = 0;
            tty->inbufHead = tty->inbufTail;
            SemRestart( tty->inbufReadySem, 0 );
            RestoreInterrupts( intsave );
        }
        break;
    default:
        request->errorCode = ERR_SERVER_UNKNOWN_OPCODE;
    }
}

/*-----------------------------------------------------------------------------*\
 *  END OF MODULE:  %M%
\*-----------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------*\
 *  MODULE:  %M%    %I%    %E%    %U%
 *
 *  TTY transmit interrupt handler.
\*-----------------------------------------------------------------------------*/

#include <kernel.h>
#include <art.h>
#include <queue.h>
#include <sem.h>
#include <tty.h>

/*------------------------------------------------------ Exported Functions ----*/

void TtyTransmit();

/*------------------------------------------------------ Imported Functions ----*/

extern unsigned char *ArtGetChanRegs();

/*=============================================================================*\
 *  Transmit the next character in the TTY echo or transmit buffer.  The
 *  echo buffer has priority.
\*=============================================================================*/

void TtyTransmit( tty )
      TtyDesc *tty;
{
      unsigned char *artch;
      char c;

      artch = ArtGetChanRegs( tty->myart, tty->artChannel );
      if (tty->echobufCount) {
            /* output an echo buffer character */
            c = tty->echobuf[tty->echobufHead];
            tty->echobufHead = (tty->echobufHead + 1) % TTY_ECHOBUF_LENGTH;
            (tty->echobufCount)--;
            if (tty->echobufCount == 0 && tty->outbufCount == 0) {
                  ArtDisable( tty->myart, tty->artChannel, ARTTIENABLE );
            }
            artch[ARTTHR] = c;
      } else if (tty->outbufCount) {
            /* output a transmit buffer character */
            c = *( (tty->outbufPtr)++ );
            if (c == '\n') {
                  /* for newline, send Cr and Lf next time */
                  if (!tty->emitLinefeed) {
                        c = TTY_CH_RETURN;
                        (tty->outbufPtr)--;
                        (tty->outbufCount)++;
                        tty->emitLinefeed = TRUE;
                  } else {
                        c = TTY_CH_LINEFEED;
                        tty->emitLinefeed = FALSE;
                  }
            }
```

```
                /* disable transmit interrupts if this is the last character */
                if ( (--(tty->outbufCount)) == 0) {
                        ArtDisable( tty->myart, tty->artChannel, ARTTIENABLE );
                        SemSignal( tty->outbufEmptySem );
                }
                artch[ARTTHR] = c;
        } else {
                /* nothing to transmit - shut off transmitter */
                ArtDisable( tty->myart, tty->artChannel, ARTTIENABLE );
        }
}

/*-------------------------------------------------------------------------------*\
 *  END OF MODULE:  %M%
\*-------------------------------------------------------------------------------*/
```

```
CC    = /usr/bin/scc
CFLAGS        = -I../h
AS    = /bin/as
ASFLAGS       =
H     = ../h
COREBLD       = /unb1/cs4605/cmd/corebld
LD    = /bin/ld
XLINK = xlink
XLINKFILE = xlink
LIBS  = ../syslib/syslib.a ../strlib/strlib.a

OBJFILES = bootentry.o userstart.o loadprog.o

userboot.im: $(XLINKFILE) userboot.oo $(LIBS)
      $(LD) userboot.oo $(LIBS) $(XLINK) -e _UserBootEntryPoint -o userboot.co
      $(COREBLD) <userboot.co >userboot.im
userboot.oo: $(OBJFILES)
      $(LD) $(OBJFILES) -r -o userboot.oo

bootentry.o: bootentry.s
loadprog.o: loadprog.c $H/kernel.h $H/server.h $H/kernserver.h $H/fileserver.h \
      $H/errcode.h
userstart.o: userstart.c $H/kernel.h $H/server.h $H/kernserver.h \
      $H/fileserver.h

clean:
      rm -rf *.o core a.out userboot.im userboot.co userboot.oo
```

```
.globl          _UserBootEntryPoint
.text
        .double _end                # length of boot code

_UserBootEntryPoint:
        jump    _UserStartup        # 0xFF0004
```

```
#include <kernel.h>
#include <server.h>
#include <kernserver.h>
#include <fileserver.h>
#include <errcode.h>

char   *LoadProgram();
Ulong BinaryRead();
void  CloseProgFile();

extern void   SetKernMsg();
extern MsgBuffer *SendKernMsg();
extern char    *strcpy();

/*****************************************************************************/
char *LoadProgram()
{
      MsgBuffer      *kernmsg;
      ThreadEntry    *the;
      Capability      filecap;
      char           *entryPoint;
      Ulong           textlen, datalen, bsslen, imagelen, magic;
      Ulong           readlen;
      Ulong           progParms[5];
      char           *loadAddr;

      /* open the program file */
      the = &(KS_AMSG_PTR->threadtab[0]);
      kernmsg = the->threadMsgBuffer;
      kernmsg->operationCode = FS_OPEN;
      kernmsg->replyPort = the->threadPort;
      kernmsg->capability = KS_AMSG_PTR->currentSearchCap;
      kernmsg->destPort = kernmsg->capability.port;
      kernmsg->data[0] = O_RDONLY;
      kernmsg->msgStartOffset = (CPint)&(kernmsg->data[2]) - (CPint)kernmsg;
      strcpy( (char *) &(kernmsg->data[2]), KS_AMSG_PTR->programFilename );
      Send( kernmsg );
      kernmsg = Receive( the->threadPort );
      if( kernmsg->errorCode ) {
            the->threadMsgBuffer = kernmsg;
            /* error code is returned in the errorCode field */
            return( NULL );
      }
      filecap = kernmsg->capability;

      /* get parameters from program file */
      readlen = BinaryRead( filecap, (char *)progParms, 5*sizeof(Ulong),
            &kernmsg, the->threadPort );
      magic      = progParms[0];
      entryPoint = (char *) progParms[1];
      textlen    = progParms[2];
      datalen    = progParms[3];
      bsslen     = progParms[4];
      imagelen   = textlen + datalen;
      if ( kernmsg->errorCode || readlen != 5*sizeof(Ulong)
                  || magic != KS_EXEC_PROG_MAGIC_NUM) {
```

```
            CloseProgFile( filecap, &kernmsg, the->threadPort );
            the->threadMsgBuffer = kernmsg;
            return( NULL );
      }

      /* allocate memory for program file */
      kernmsg->capability = KS_AMSG_PTR->addrSpaceCap;
      kernmsg->destPort = kernmsg->capability.port;
      kernmsg->operationCode = KS_CREATE_MEMORY;
      kernmsg->data[0] = textlen + datalen + bsslen;
      Send( kernmsg);
      kernmsg = Receive( the->threadPort );
      loadAddr = (char *) kernmsg->data[0];

      /* load program file */
      readlen = BinaryRead( filecap, loadAddr, imagelen, &kernmsg,
            the->threadPort );
      if( kernmsg->errorCode || readlen != imagelen ) {
            entryPoint = NULL;
      }

      /* close program file and exit */
      CloseProgFile( filecap, &kernmsg, the->threadPort );
      the->threadMsgBuffer = kernmsg;
      return( entryPoint );
}
/*****************************************************************************/
void CloseProgFile( filecap, msgptr, replyPort )
      Capability      filecap;
      MsgBuffer     **msgptr;
      Port            replyPort;
{
      register MsgBuffer *msg;

      msg = *msgptr;
      msg->capability = filecap;
      msg->destPort = msg->capability.port;
      msg->operationCode = FS_CLOSE;
      Send( msg );
      msg = Receive( replyPort );
      *msgptr = msg;
}

/*****************************************************************************/
Ulong BinaryRead( filecap, buf, bytes, msgptr, replyPort )
      Capability      filecap;
      register char  *buf;
      Ulong           bytes;
      MsgBuffer     **msgptr;
      Port            replyPort;
{
      Ulong           maxChunkSize;
      register MsgBuffer *msg;
      register char  *fileptr;
      register Ulong  dataCount;
      Ulong           bytesRead;
```

```
        msg = *msgptr;
        maxChunkSize = KS_BOOT_MSG_BUFFER_LENGTH
                        - ( (CPint)(msg->data) - (CPint)msg );
        msg->capability = filecap;
        msg->operationCode = FS_READ;
        bytesRead = 0;
        while (bytes > 0) {
                msg->destPort = msg->capability.port;
                msg->msgLength = (bytes < maxChunkSize) ? bytes : maxChunkSize;
                Send( msg );
                msg = Receive( replyPort );
                dataCount = msg->msgLength;
                if( dataCount == 0 || msg->errorCode ) {
                        bytes = dataCount = 0;
                }
                bytes -= dataCount;
                bytesRead += dataCount;
                fileptr = (char *)msg + msg->msgStartOffset;
                memcpy( (Void*) buf, (Void*) fileptr, dataCount );
                buf += dataCount;
        }
        *msgptr = msg;
        return( bytesRead );
}
```

```
#include <kernel.h>
#include <server.h>
#include <kernserver.h>
#include <fileserver.h>

void  UserStartup();

extern char    *LoadProgram();
extern void   SetKernMsg();
extern MsgBuffer *SendKernMsg();

/************************************************************************/
void UserStartup()
{
      int     argc;
      char  **argv;
      char   *envp;
      Port    port;
      MsgBuffer *kernmsg;
      int     (*entryPoint)();
      ThreadEntry *the;
      int     err;

      /* set up parameters and environment */
      argc = KS_AMSG_PTR->argc;
      argv = KS_AMSG_PTR->argv;
      envp = NULL;

      /* load user program */
      entryPoint = (int (*)()) LoadProgram();

      /* call user program if loaded properly */
      if ( entryPoint != NULL ) {
            (entryPoint) ( argc, argv, envp );
      } else {
            /* should exit() here with error code */
            /* error code will be in initial message buffer */
      }

      /* kill initial thread */
      the = &(KS_AMSG_PTR->threadtab[0]);
      kernmsg = the->threadMsgBuffer;
      kernmsg->operationCode = KS_REMOVE_THREAD;
      kernmsg->replyPort.machineID = SERV_ALIAS_MACHINE;
      kernmsg->replyPort.portDesc = SERV_ALIAS_MSG_RECYCLER;
      kernmsg->capability = the->threadCap;
      kernmsg->destPort = kernmsg->capability.port;
      Send( kernmsg );
      kernmsg = the->threadMsgBuffer = Receive( the->threadPort );
      /* should never reach this point */
      asm("bpt");
}
```

```
/*   xlink    1.2    92/08/31.12:03:31 */

MEMORY {
      bootmem : o=0xFF0000 l=1536
}

SECTIONS {
      .text : { userboot.oo(.text) }
      .data : { userboot.oo(.data) }
      .bss  : { userboot.oo(.bss)  }
}
```

```
AR     = /bin/ar
CC     = /usr/bin/scc
CFLAGS         = -I../h
H      = ../h

.c.o:
       $(CC) $(CFLAGS) -c $<

OBJFILES = doprint.o strfuncs.o convert.o memcpy.o memfuncs.o charfuncs.o atou.o

strlib.a: $(OBJFILES)
       rm -rf strlib.a
       $(AR) cr strlib.a $(OBJFILES)
clean:
       rm -rf *.o core a.out strlib.a

atou.o: atou.c
charfuncs.o: charfuncs.c
convert.o: convert.c
doprint.o: doprint.c
memcpy.o: memcpy.c
memfuncs.o: memfuncs.c $H/stdio.h
strfuncs.o: strfuncs.c $H/stdio.h
```

```
/*    atou.c    1.3    92/08/31    12:09:38    */

unsigned long int atou();

/*****************************************************************************/
unsigned long int atou( s )
      register char *s;
{
      register unsigned long val;
      register unsigned char c;

      val = 0;
      while ( (c = *s++) != '\0' ) {
              val = val * 10 + c - '0';
      }
      return( val );
}
```

```
/*    charfuncs.c    1.3    92/08/31    12:09:44    */

char tolower();

char _ctype[256] = { 0 };

/*****************************************************************************/
char tolower( c )
     char c;
{
     return( c - 'A' + 'a' );
}
```

```
/*    convert.c    1.5    92/08/31    12:09:50    */
/*  written 03-JUN-91 by Craig Bruce  */

void  utoa();

/*****************************************************************************/
void utoa (n, a)
      unsigned long   n;
      char            a[];
{
      register unsigned long  i, num;
      register long           j;
      char                    trans[20];

      if (n==0) {
              a[0] = '0';
              a[1] = '\0';
      } else {
              trans[0] = '\0';
              num = n;
              for (i=1; num != 0; i++) {
                      trans[i] = '0' + (char)(num % 10);
                      num /= 10;
              }
              for (j=i-1, i=0; j>=0; j--, i++) {
                      a[i] = trans[j];
              }
      }
}

/*****************************************************************************/
void htoa (n, a)
      unsigned long n;
      char a[];
{
      static char hexdigits [] =
          {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
      register unsigned long  i, num;
      register long           j;
      char                    trans[20];

      if (n==0) {
              a[0] = '0';
              a[1] = '\0';
      } else {
              trans[0] = '\0';
              num = n;
              for (i=1; num != 0; i++) {
                      trans[i] = hexdigits[num % 16];
                      num /= 16;
              }
              for (j=i-1, i=0; j>=0; j--, i++) {
                      a[i] = trans[j];
              }
      }
}
```

```
/*    doprint.c    1.5    92/08/31    12:09:57    */

int           _DoPrint();
static int    _DoPuts();

extern void utoa();
extern void htoa();

/*****************************************************************************/
int _DoPrint (func, stream, f, args)
      int     (*func)();
      unsigned long stream;
      char    f[];
      unsigned long *args;
{
      unsigned long *ip;
      long i;
      char c;
      char a[20];
      int count;

      count = 0;
      ip = args;
      for (i=0; (c=f[i]) != '\0'; i++) {
              switch (c) {
              case '%':
                      c=f[++i];
                      switch (c) {
                      case 'i': case 'd':
                              utoa( *(ip++), a );
                              count += _DoPuts( func, stream, a );
                              break;
                      case 'x': case 'X':
                              htoa (*(ip++), a);
                              count += _DoPuts( func, stream, a );
                              break;
                      case 's':
                              count += _DoPuts( func, stream, *(ip++) );
                              break;
                      case 'c':
                              (*func)( (char) *(ip++), stream );
                              count++;
                              break;
                      case '%':
                              (*func)( '%', stream );
                              count++;
                              break;
                      case '\0':
                              i--;
                              break;
                      }
                      break;
              default:
                      (*func)( c, stream );
                      count++;
                      break;
```

```
                }
        }
        return( count );
}

/*****************************************************************************/
static int _DoPuts (func, stream, s)
        int     (*func)();
        unsigned long stream;
        char    s[];
{
        register long i;
        register char c;

        for (i=0; (c=s[i]) != '\0'; i++) {
                (*func)( c, stream );
        }
        return( i );
}
```

```
/*    memcpy.c    1.2    92/08/31    12:10:05    */

#include <stdio.h>
#define  NS32000

char *memcpy();

/*****************************************************************************/
char *memcpy (dest, source, count)
      register char   *dest;
      register char   *source;
      register size_t count;
{
      long    longWordCount;
      char *ret;

      ret = dest;
#ifdef NS32000
      /* we don't have to preserve r0,r1,r2 for ns32k compiler or GCC */
      if( (unsigned)dest%4==0 && (unsigned)source%4==0 && count%4==0 ) {
              /* move all data as long words - more efficient */
              longWordCount = count / 4;
              asm("movd    -4(fp),r0");       /* r0 = longWordCount */
              asm("movd    12(fp),r1");       /* r1 = source */
              asm("movd    8(fp), r2");       /* r2 = dest   */
              asm("movsd");                   /* move long words */
      } else {
              asm("movd    16(fp),r0");       /* r0 = count  */
              asm("movd    12(fp),r1");       /* r1 = source */
              asm("movd    8(fp), r2");       /* r2 = dest   */
              asm("movsb");                   /* move bytes  */
      }
#else
      while( count-- ) {
              *dest++ = *source++;
      }
#endif
      return( ret );
}
```

```
/*    memfuncs.c    1.4    92/08/31    12:10:24    */

#include <stdio.h>

char *memset();
char *memchr();

/*****************************************************************************/
char *memset (dest, c, count)
      register char  *dest;
      register char   c;
      register size_t count;
{
      char *ret;

      ret = dest;
      while( count-- ) {
              *dest++ = c;
      }
      return( ret );
}

/*****************************************************************************/
char *memchr (dest, c, count)
      register char  *dest;
      register char   c;
      register size_t count;
{
      while( count-- && *dest != c && *dest ) {
              dest++;
      }
      if( ++count ) {
              return( dest );
      } else {
              return( (char *) NULL );
      }
}
```

```
/*    strfuncs.c    1.4    92/08/31    12:10:31    */

#include <stdio.h>

char   *strcpy();
char   *strcat();
size_t        strlen();
char   *strchr();
int    strcmp();

/*****************************************************************************/
char *strcpy (dest, source)
      register char *dest;
      register char *source;
{
      char *ret;

      ret = dest;
      while( *dest++ = *source++ ) ;
      return( ret );
}

/*****************************************************************************/
char *strcat (dest, source)
      register char *dest;
      register char *source;
{
      char *ret;

      ret = dest;
      dest--;
      while( *++dest );
      while( *dest++ = *source++ ) ;
      return( dest );
}

/*****************************************************************************/
size_t strlen (string)
      register char *string;
{
      char *original;

      original = string;
      while( *string++ ) ;
      return( (size_t)string - (size_t)original );
}

/*****************************************************************************/
char *strchr (string, c)
      register char *string;
      register char c;
{
      while( *string != c && *string ) {
            string++;
      }
      if( *string ) {
```

```
                return( string );
        } else {
                return( (char *) NULL );
        }
}

/*****************************************************************************/
int strcmp(s1, s2)
        register char *s1;
        register char *s2;
{
        s1--;
        s2--;
        while (*++s1 == *++s2 && *s1) ;
        if (*s1 < *s2) {
                /* string1 is less than string2 */
                return( -1 );
        } else if (*s1 > *s2) {
                /* string1 is greater than string2 */
                return( 1 );
        } else {
                /* strings are equal */
                return( 0 );
        }
}
```

```
AR    = /bin/ar
CC    = /usr/bin/scc
CFLAGS        = -I../h
AS    = /bin/as
ASFLAGS       =
H     = ../h

.c.o:
      $(CC) $(CFLAGS) -c $<
.s.o:
      $(AS) $(ASFLAGS) -o $*.o $<

OBJFILES = callsoc.o callsrw.o fexec.o callutil.o threadlib.o msglib.o \
      threadtab.o abort.o userpanic.o statmsg.o svc.o malloc.o

syslib.a: $(OBJFILES)
      rm -f syslib.a
      $(AR) cr syslib.a $(OBJFILES)
clean:
      rm -f *.o core a.out syslib.a

svc.o: svc.s
abort.o: abort.c
callsoc.o: callsoc.c $H/kernel.h $H/server.h $H/kernserver.h $H/fileserver.h \
      $H/errcode.h $H/debug.h
callsrw.o: callsrw.c $H/kernel.h $H/server.h $H/kernserver.h $H/fileserver.h \
      $H/errcode.h
callutil.o: callutil.c $H/kernel.h $H/server.h $H/kernserver.h $H/fileserver.h
fexec.o: fexec.c $H/kernel.h $H/server.h $H/kernserver.h $H/errcode.h
malloc.o: malloc.c $H/kernel.h $H/syslib.h $H/server.h $H/kernserver.h
msglib.o: msglib.c $H/kernel.h $H/server.h $H/kernserver.h
statmsg.o: statmsg.c $H/kernel.h $H/server.h $H/kernserver.h $H/fileserver.h
threadlib.o: threadlib.c $H/kernel.h $H/server.h $H/kernserver.h
threadtab.o: threadtab.c $H/kernel.h $H/server.h $H/kernserver.h
userpanic.o: userpanic.c $H/kernel.h
```

```
/*    abort.c    1.4    92/08/31    12:13:49    */

void  abort();
void  exit();

/*****************************************************************************/
void abort()
{
      UserPanic("library call: abort");
}

/*****************************************************************************/
void exit( status )
      int status;
{
      StatMsg("*** UserExit: status=%d\n", status);
      asm("bpt");
}
```

```
/*    callsoc.c    1.2    92/08/31    12:13:53    */

#include <kernel.h>
#include <server.h>
#include <kernserver.h>
#include <fileserver.h>
#include <errcode.h>
#include <debug.h>

int open();
int creat();
int close();

extern void   SetFileBuffer();
extern MsgBuffer *GetFileBuffer();
extern void   SetKernMsg();
extern MsgBuffer *SendKernMsg();

/*****************************************************************************/
int open( name, mode, perms )
      char   *name;
      Ulong   mode;
      Ulong   perms;  /* optional */
{
      MsgBuffer *msg;
      Port replyPort;
      Ulong fd;
      FileEntry *fent;

      /* search for free file descriptor */
      for (fd=0; fd<KS_MAX_FILES && KS_AMSG_PTR->filetab[fd].used; fd++) ;
      if (fd >= KS_MAX_FILES) {
            SetErrno( EMFILE );
            return( -1 );
      }
      fent = &(KS_AMSG_PTR->filetab[fd]);
      fent->used = TRUE;
      fent->fileMsgBuffer = NULL;
#     ifdef DEBUG_SYSCALL_OPEN
      StatMsg("SCLopen: got file descriptor %d, addr 0x%X\n", fd, fent);
#     endif

      /* set up open call to server */
      SetKernMsg( &msg, &replyPort );
      msg->data[0] = mode;
      if (mode & O_CREAT) {
            msg->data[1] = perms;
      }
      msg->capability = KS_AMSG_PTR->currentSearchCap;
      msg->destPort = msg->capability.port;
      msg->replyPort = replyPort;
      msg->msgStartOffset = (CPint)&(msg->data[2]) - (CPint)msg;
      msg->operationCode = FS_OPEN;
      strcpy( (char *) &(msg->data[2]), name );

      /* send message and handle reply */
```

```
	msg = SendKernMsg( msg, replyPort );
	if (msg->errorCode == ERR_OK) {
		fent->fileCap = msg->capability;
	} else {
		fent->used = FALSE;
		SetErrno( msg->errorCode );
		return( -1 );
	}

	/* exit */
#	ifdef DEBUG_SYSCALL_OPEN
	StatMsg("SCLopen: open ok, returning fd=%d\n", fd);
#	endif
	return( fd );
}

/*****************************************************************************/
int creat( name, perms )
	char	*name;
	Ulong	perms;
{
	return( open( name, O_WRONLY | O_CREAT | O_TRUNC, perms ) );
}

/*****************************************************************************/
int close( fd )
	int	fd;
{
	FileEntry *fent;
	MsgBuffer *msg;
	Port replyPort;

#	ifdef DEBUG_SYSCALL_OPEN
	StatMsg("SCLclose: fd=%d\n", fd);
#	endif
	fent = &(KS_AMSG_PTR->filetab[fd]);
	if (! fent->used) {
		SetErrno( EBADF );
		return( -1 );
	}
	if ((msg = fent->fileMsgBuffer) != NULL) {
		/* dispose of file message buffer */
		GetPortAlias( SERV_ALIAS_MSG_RECYCLER, &(msg->destPort) );
		Send( msg );
		fent->fileMsgBuffer = NULL;
	}

	/* set up close call to server */
	SetKernMsg( &msg, &replyPort );
	msg->capability = fent->fileCap;
	msg->destPort = msg->capability.port;
	msg->replyPort = replyPort;
	msg->operationCode = FS_CLOSE;

	/* send message and handle reply */
	msg = SendKernMsg( msg, replyPort );
```

```
        fent->used = FALSE;
        if( msg->errorCode != ERR_OK ) {
                SetErrno( msg->errorCode );
                return( -1 );
        }
        return( 0 );
}

/*****************************************************************************/
int dup( fd )
        int     fd;
{
        UserPanic("Syscall dup called");
}
```

```
/*    callsrw.c    1.4    92/08/31    12:14:00    */

#include <kernel.h>
#include <server.h>
#include <kernserver.h>
#include <fileserver.h>
#include <errcode.h>

int   write();
int   read();
int   isatty();
off_t lseek();

Ulong errno = 0;

extern void   SetFileBuffer();
extern MsgBuffer *GetFileBuffer();
extern void   GetThreadPort();

/*****************************************************************************/
int read( fd, buf, n)
      int     fd;
      char   *buf;
      int     n;
{
      Port    replyPort;
      MsgBuffer *msg;
      long    bytes;

/*    StatMsg("read( fd=%i, buf=%x, n=%i )\n", fd,buf,n);*/
      bytes = (n <= KS_FILE_DATA_LENGTH) ? n : KS_FILE_DATA_LENGTH;
      msg = GetFileBuffer( fd );
      GetThreadPort( &replyPort );
      msg->msgLength = bytes;
      msg->destPort = msg->capability.port;
      msg->operationCode = FS_READ;
      msg->replyPort = replyPort;
      Send( msg );
      msg = Receive( replyPort );
      SetFileBuffer( fd, msg );
      bytes = msg->msgLength;
      memcpy( buf, (Void *) ((CPint)msg + msg->msgStartOffset), bytes );
      if (msg->errorCode) {
              SetErrno( msg->errorCode );
              bytes = -1;
      }
      return( bytes );
}

/*****************************************************************************/
int write( fd, buf, n)
      int     fd;
      char   *buf;
      int     n;
{
      Port    replyPort;
```

```
	MsgBuffer *msg;
	Ulong	bytes, written;

/*	StatMsg("write( fd=%i, buf=%x, n=%i )... \n", fd,buf,n);*/
	GetThreadPort( &replyPort );
	written = 0;
	while (written < n) {
/*		StatMsg("written=%i, buf=%x... ", written, buf);*/
		bytes = (n - written) <= KS_FILE_DATA_LENGTH
				? (n - written) : KS_FILE_DATA_LENGTH;
		msg = GetFileBuffer( fd );
/*		StatMsg("msg=%x, port=%x\n",msg,port.portDesc);*/
		msg->msgStartOffset = KS_FILE_DATA_OFFSET;
		msg->msgLength = bytes;
		msg->destPort = msg->capability.port;
		msg->operationCode = FS_WRITE;
		msg->replyPort = replyPort;
		memcpy( (Void *) ((Ulong)msg + msg->msgStartOffset),
			buf, bytes );
		Send( msg );
		msg = Receive( replyPort );
		SetFileBuffer( fd, msg );
		written += msg->msgLength;
		buf += bytes;
		if ((errno = msg->errorCode) != ERR_OK) {
			SetErrno( msg->errorCode );
			n = 0;
		}
	}
	return( written );
}

/*****************************************************************************/
int isatty( fd )
	int	fd;
{
	Port port;
	MsgBuffer *msg;

	if (fd < KS_MAX_FILES) {
		msg = GetFileBuffer( fd );
		GetThreadPort( &port );
		msg->operationCode = FS_GET_DEVICE_TYPE;
		msg->destPort = msg->capability.port;
		msg->replyPort = port;
		Send( msg );
		msg = Receive( port );
		SetFileBuffer( fd, msg );
		return( msg->data[0] == FS_DEVICE_TYPE_TTY );
	} else {
		return( 0 );
	}
}

/*****************************************************************************/
off_t lseek( fd, offset, origin )
```

```
        int     fd;
        off_t   offset;
        int     origin;
{
        UserPanic("Syscall: lseek called");
}
```

```
/*    callutil.c    1.3    92/08/31    12:14:06    */

#include <kernel.h>
#include <server.h>
#include <kernserver.h>
#include <fileserver.h>

MsgBuffer *GetFileBuffer();
void  SetFileBuffer();

extern void   SetKernMsg();
extern MsgBuffer *SendKernMsg();

/*****************************************************************************/
MsgBuffer *GetFileBuffer( fd )
      int     fd;
{
      MsgBuffer *msg, *kernmsg;
      FileEntry *file;
      Port port;

      file = &(KS_AMSG_PTR->filetab[ fd ]);
      msg = file->fileMsgBuffer;
      if (msg == (MsgBuffer *) NULL) {
              /*Kputf("forced to create file message buffer\n");*/
              /* create a new message buffer */
              SetKernMsg( &kernmsg, &port );
              kernmsg->operationCode = KS_CREATE_MSG_BUFFER;
              kernmsg->data[0] = KS_FILE_BUFFER_LENGTH;
              kernmsg = SendKernMsg( kernmsg, port );
              /* should check error code */
              msg = (MsgBuffer *) kernmsg->data[0];
              msg->capability = file->fileCap;
              msg->msgStartOffset = (Ulong)&msg->data[0] - (Ulong)msg;
              file->fileMsgBuffer = msg;
      }
      return( msg );
}

/*****************************************************************************/
void SetFileBuffer( fd, msg )
      Ulong   fd;
      MsgBuffer *msg;
{
       (KS_AMSG_PTR->filetab[fd]).fileMsgBuffer = msg;
}
```

```
#include <kernel.h>
#include <server.h>
#include <kernserver.h>
#include <errcode.h>

int   fexec();

char   *_PlaceFexecString();

extern void   SetKernMsg();
extern MsgBuffer *SendKernMsg();

/*****************************************************************************/
int fexec(progfile, argc, argv, envp, stdinCap, stdoutCap, stderrCap,
            priority, async)
      char *progfile;
      int argc;
      char *argv[];
      char *envp[];
      Capability stdinCap, stdoutCap, stderrCap;  /*change to fd's*/
      Ulong priority;
      Ulong async;
{
      CreateAddrSpaceMsg *msg;
      MsgBuffer *replymsg;
      Port port;
      MsgBuffer *kernmsg;
      Ulong i;
      Errcode retcode;
      char *cptr;

      /* get create addr space message buffer */
      SetKernMsg( &kernmsg, &port );
      kernmsg->operationCode = KS_CREATE_MSG_BUFFER;
      kernmsg->data[0] = sizeof(CreateAddrSpaceMsg) + 1000;
      kernmsg = SendKernMsg( kernmsg, port );
      msg = (CreateAddrSpaceMsg *) kernmsg->data[0];

      /* fill in creation request message buffer */
      *msg = *KS_AMSG_PTR;
      GetPortAlias( SERV_ALIAS_GLOBAL_AS_SERVER, &(msg->destPort) );
      msg->operationCode = KS_CREATE_ADDR_SPACE;
      msg->replyPort = port;
      if (async) {
            GetPortAlias(SERV_ALIAS_MSG_RECYCLER,&(msg->notificationPort) );
      } else {
            msg->notificationPort = port;
      }
      msg->requestedPriority = priority;

      /* handle std files - change to use file numbers */
      msg->filetab[0].fileCap = stdinCap;
      msg->filetab[1].fileCap = stdoutCap;
      msg->filetab[2].fileCap = stderrCap;

      /* copy parameters and environment */
```

```
	msg->argc = argc;
	cptr = (char *) &(msg->argv[argc]);
	for (i=0; i<argc; i++) {
		msg->argv[i] = _PlaceFexecString( argv[i], msg, &cptr );
	}

	/* copy program filename */
	msg->programFilename = _PlaceFexecString( progfile, msg, &cptr );

	/* send create addr space message buffer */
	Send( msg );
	replymsg = Receive( port );

	/* check out reply */
	retcode = (Errcode) replymsg->errorCode;

	/* get rid of creation reply message */
	GetPortAlias( SERV_ALIAS_MSG_RECYCLER, &(replymsg->destPort) );
	Send( replymsg );

	/* if synchronous, wait for notification of death message */
	if (retcode == ERR_OK && !async) {
		replymsg = Receive( port );
		retcode = (Errcode) replymsg->errorCode;
		GetPortAlias( SERV_ALIAS_MSG_RECYCLER, &(replymsg->destPort) );
		Send( replymsg );
	}
	return( retcode );
}

/*****************************************************************************/
char *_PlaceFexecString( str, msg, inCptr )
	char	*str;
	MsgBuffer *msg;
	char	**inCptr;
{
	register char *cptr, *ret;

	/* check for problems? */
	cptr = *inCptr;
	ret = cptr - (Ulong)msg + KS_ADDR_ADDR_SPACE_MSG;
	strcpy( cptr, str );
	cptr += strlen( str ) + 1;
	*inCptr = cptr;
	return( ret );
}
```

```
/* Written 91/12/09 by Craig Bruce */

#include <kernel.h>
#include <syslib.h>
#include <server.h>
#include <kernserver.h>

Void   *malloc();
void  free();
Void   *_rawMalloc();
void  _rawFree();

LOCAL Void *_GetMoreCore();
MemBlock mallocMasterBlock = { NULL, 0 };

extern void   SetKernMsg();
extern MsgBuffer *SendKernMsg();

/*****************************************************************************/
Void *malloc( inputSize )
      Ulong   inputSize;
{
      Ulong  *p;
      Ulong   size, getmemSize;

      size = inputSize + 2 * sizeof(Ulong);
      p = (Ulong*) _rawMalloc( size );
      if( p == NULL ) {
              /* get pages from kernel server */
              getmemSize = size>MALLOC_GETMEM_SIZE ? size: MALLOC_GETMEM_SIZE;
              p = (Ulong*) _GetMoreCore( &getmemSize );
              _rawFree( (Void*)p, getmemSize );
              p = (Ulong*) _rawMalloc( size );
      }

      /* put in the block header */
      if( p != NULL ) {
              *p++ = MALLOC_MAGIC_NUMBER;
              *p++ = size;
      }
      return( (Void*) p );
}

/*****************************************************************************/
void free( inp )
      Void   *inp;
{
      Ulong  *p;
      Ulong   size;

      p = (Ulong*)inp - 2;
      if( *p != MALLOC_MAGIC_NUMBER ) return;
      size = *(p+1);
      _rawFree( (Void*)p, size );
}
```

```
/*****************************************************************************/
Void *_rawMalloc( inputSize )
      Ulong inputSize;
{
      register Ulong size;
      register MemBlock *p, *q;
      MemBlock *leftover;

      /* align the request size to malloc-block-size multiple */
      size = ( (inputSize + sizeof(MemBlock) - 1) / sizeof(MemBlock) )
              * sizeof(MemBlock);

      /* search for a large enough block */
      q = &mallocMasterBlock;
      p = mallocMasterBlock.nextBlock;
      while (p != NULL && p->blockLength < size) {
              q = p;
              p = p->nextBlock;
      }

      if (p == NULL) {
              /* we are SOL (out of luck) */
              return( NULL );
      }

      if( p->blockLength == size ) {
              /* unlink the entire block */
              q->nextBlock = p->nextBlock;
      } else {
              /* link in leftover amount of block */
              leftover = (MemBlock *) ((CPint)p + size);
              q->nextBlock = leftover;
              leftover->nextBlock = p->nextBlock;
              leftover->blockLength = p->blockLength - size;
      }

      return( (Void *) p );
}

/*****************************************************************************/
void _rawFree( inputNew, inputSize )
      MemBlock *inputNew;
      Ulong   inputSize;
{
      register MemBlock *q, *new, *p;
      register Ulong size;

      size = ( (inputSize + sizeof(MemBlock) - 1 ) / sizeof(MemBlock))
              * sizeof(MemBlock);
      new = inputNew;

      /* search for correct spot to put new block */
      q = &mallocMasterBlock;
      p = mallocMasterBlock.nextBlock;

      while( p != NULL && p < new ) {
```

```
                q = p;
                p = p->nextBlock;
        }

        /* at this point, q and p straddle the new block */

        /* try to coalesce q with new */
        if( (CPint)q + q->blockLength == (CPint)new ) {
                /* coalesce */
                new = q;
                size += q->blockLength;
        } else {
                /* link in as new block */
                q->nextBlock = new;
        }

        /* try to coalesce new with p */
        if (p != NULL && (CPint)new + size == (CPint)p) {
                /* coalesce */
                size += p->blockLength;
                new->nextBlock = p->nextBlock;
        } else {
                /* will not coalesce */
                new->nextBlock = p;
        }
        new->blockLength = size;
}

/*****************************************************************************/
LOCAL Void *_GetMoreCore( inoutGetmemSize )
        Ulong   *inoutGetmemSize;
{
        MsgBuffer *msg;
        Port replyPort;

        SetKernMsg( &msg, &replyPort );
        msg->operationCode = KS_CREATE_MEMORY;
        msg->data[0] = *inoutGetmemSize;
        msg = SendKernMsg( msg, replyPort );
        *inoutGetmemSize = msg->data[1];
        return( (Void*) msg->data[0] );
}
```

```
#include <kernel.h>
#include <server.h>
#include <kernserver.h>

void  SetKernMsg();
MsgBuffer *SendKernMsg();
void  GetPortAlias();

extern void   GetThreadPort();
extern ThreadEntry *GetThreadEntry();
extern MsgBuffer *GetThreadMsgBuffer();

/*****************************************************************************/
void SetKernMsg( outkernmsg, outport )
      MsgBuffer **outkernmsg;
      Port *outport;
{
      MsgBuffer *kernmsg;
      Port port;

      GetThreadPort( &port );
      kernmsg = GetThreadMsgBuffer();
      kernmsg->capability = KS_AMSG_PTR->addrSpaceCap;
      kernmsg->destPort = kernmsg->capability.port;
      kernmsg->msgStartOffset = 0;
      kernmsg->msgLength = 0;
      kernmsg->operationCode = 0;
      kernmsg->replyPort = port;
      kernmsg->requestNumber = 0;
      kernmsg->errorCode = 0;
      *outport = port;
      *outkernmsg = kernmsg;
}

/*****************************************************************************/
MsgBuffer *SendKernMsg( kernmsg, port )
      MsgBuffer *kernmsg;
      Port port;
{
      Send( kernmsg );
      kernmsg = Receive( port );
      GetThreadEntry()->threadMsgBuffer = kernmsg;
      return( kernmsg );
}

/*****************************************************************************/
void GetPortAlias( aliasid, outPortName )
      Ulong aliasid;
      Port *outPortName;
{
      outPortName->machineID = SERV_ALIAS_MACHINE;
      outPortName->portDesc = aliasid;
      outPortName->portCheck = 0;
}
```

```
#include <kernel.h>
#include <server.h>
#include <kernserver.h>
#include <fileserver.h>

void  StatMsg();
int   _StatMsgPutc();

extern void   GetThreadPort();
extern MsgBuffer *GetThreadMsgBuffer();
extern MsgBuffer *SendKernMsg();

/*****************************************************************************/
void StatMsg( format, args )
      char    format[];
      CPint   args;
{
      MsgBuffer *msg;
      Port port;

      /* get msg and port from the thread table entry */
      GetThreadPort( &port );
      msg = GetThreadMsgBuffer();

      /* set up the message buffer */
      GetPortAlias( SERV_ALIAS_KERN_STATUS, &(msg->destPort) );
      msg->msgStartOffset = (Ulong) &(msg->data[0]) - (Ulong)msg;
      msg->msgLength = 0;
      msg->operationCode = FS_WRITE;
      msg->replyPort = port;

      /* put in string and send */
      _DoPrint( _StatMsgPutc, (CPint)msg, format, &args );
      SendKernMsg( msg, port );
}

/*****************************************************************************/
int _StatMsgPutc( c, msg )
      char c;
      MsgBuffer *msg;
{
      char *cptr;

      cptr = (char *) ( (CPint)msg + msg->msgStartOffset + msg->msgLength );
      *cptr = c;
      (msg->msgLength)++;
      return( 0 );
}
```

```
.globl        _Send
.globl        _Receive
.globl        _ReceiveOpt
.set  NULLMSG,      0
.set  OPSEND,       0x53454E44
.set  OPRECEIVE,    0x52454356
.set  WAITTRUE,     1

# register usage:
#    r0 = operation (call)          r4 = message buffer pointer (call)
#    r1 = port.machine id           r5 = wait flag (call)
#    r2 = port.port descriptor             r6 = physical message length (return)
#    r3 = port.port check           r7 = error code (return)
#
# return r0:
#    send: r0 = error code
#    receive: r0 = message buffer pointer

.text

#*****************************************************************************
_Send:        # ( msgBuffer )
      enter   [r1,r2,r3,r4],0 # preserve resgisters (not r0)
      movd    8(fp),r4                # get msgBuffer (parm 1)
      movd    0(r4),r1                # get port.machine from msgBuffer
      movd    4(r4),r2                # get port.portDesc from msgBuffer
      movd    8(r4),r3                # get port.check from msgBuffer
      movd    $OPSEND,r0              # set operation to Send
      svc                             # call kernel
      exit    [r1,r2,r3,r4]   # restore registers (not r0)
      ret     0

#*****************************************************************************
_Receive:  # ( port )
      enter   [r1,r2,r3,r5,r6,r7],0
      movd    $WAITTRUE,r5            # set wait flag to true
      movd    8(fp),r1                # get port.machine from parm 1
      movd    12(fp),r2               # get port.port desc from parm 1
      movd    16(fp),r3               # get port.check from parm 1
      movd    $OPRECEIVE,r0           # set Receive operation
      svc                             # call kernel
      cmpd    $NULLMSG,r0             # check if message returned
      beq     _nomsg                  #
      movd    r6,12(r0)               # set physical length of msgBuffer
_nomsg:
      exit    [r1,r2,r3,r5,r6,r7]
      ret     0

#*****************************************************************************
_ReceiveOpt:  # ( port, waitFlag, *outErrcode )
      enter   [r1,r2,r3,r5,r6,r7],0
      movd    20(fp),r5               # get wait flag from parm 2
      movd    8(fp),r1                # get port.machine id from parm 1
      movd    12(fp),r2               # get port.port desc from parm 1
      movd    16(fp),r3               # get port.check from parm 1
      movd    $OPRECEIVE,r0           # set Receive operation
```

```
        svc                             # call kernel
        cmpd    $NULLMSG,r0             # check if message actually received
        beq     _nomsgopt
        movd    r6,12(r0)               # store physical length of msgBuffer
_nomsgopt:
        movd    r7,0(24(fp))            # store error code into parm 3
        exit    [r1,r2,r3,r5,r6,r7]
        ret     0
```

```
#include <kernel.h>
#include <server.h>
#include <kernserver.h>

void  _RegisterThread();

extern void   GetThreadPort();
extern MsgBuffer *GetThreadMsgBuffer();
extern MsgBuffer *SendKernMsg();

/*****************************************************************************/
void _RegisterThread( threadID, msgbuf, port, threadCap )
      Ulong   threadID;
      MsgBuffer *msgbuf;
      Port    port;
      Capability threadCap;
{
      ThreadEntry *the;
      Ulong i;

      /* should lock the thread table at this point */

      /* search for free thread table entry */
      for (i=0, the=KS_AMSG_PTR->threadtab;
           i<KS_MAX_THREADS && the->used;
           i++, the++) {
      }

      if (i<KS_MAX_THREADS) {
              /* fill in thread info */
              the->used = TRUE;
              the->threadID = threadID;
              the->threadCap = threadCap;
              the->threadMsgBuffer = msgbuf;
              the->threadPort = port;
              the->threadErrno = 0;
      } else {
              UserPanic( "_RegisterThread: no free thread table entries" );
      }
}
```

```
#include <kernel.h>
#include <server.h>
#include <kernserver.h>

ThreadEntry    *GetThreadEntry();
MsgBuffer      *GetThreadMsgBuffer();
void  GetThreadPort();
void  GetThreadEntryCap();
int   GetErrno();
void  SetErrno();

extern int    errno;

/*****************************************************************************/
ThreadEntry *GetThreadEntry()
{
      register Ulong i;
      register ThreadEntry *the;

      the = &(KS_AMSG_PTR->threadtab[0]);

      /* see if threadtab[0] is for current thread */
      if (the->threadID == KS_CURRENT_THREAD) {
            return( the );
      }

      /* look for current thread entry */
      for (i=0;
           i<KS_MAX_THREADS && the->threadID != KS_CURRENT_THREAD;
           i++, the++) ;
      if (i<KS_MAX_THREADS) {
            return( the );
      } else {
            UserPanic("GetThreadTableEntry: can't find current thread");
      }
}

/*****************************************************************************/
void GetThreadPort( outPort )
      Port *outPort;
{
      *outPort = GetThreadEntry()->threadPort;
}

/*****************************************************************************/
MsgBuffer *GetThreadMsgBuffer()
{
      return( GetThreadEntry()->threadMsgBuffer );
}

/*****************************************************************************/
void GetThreadEntryCap( outCap )
      Capability *outCap;
{
      *outCap = GetThreadEntry()->threadCap;
}
```

```
/*****************************************************************************/
int GetErrno()
{
      return( GetThreadEntry()->threadErrno );
}

/*****************************************************************************/
void SetErrno( errnoValue )
      int errnoValue;
{
      GetThreadEntry()->threadErrno = errnoValue;
      errno = errnoValue;
}
```

```
/*    userpanic.c    1.2    92/08/31    12:15:02    */

#include <kernel.h>

void  UserPanic();

extern void   exit();

/*****************************************************************************/
void UserPanic( s )
      char *s;
{
      StatMsg( "*** UserPanic: %s ***\n", s );
      exit( 1 );
}
```

```
#    Makefile    1.2    92/08/31    12:18:57

USERFILES = shell hex
USERDEPS = shelldep hexdep
USERPROGS = shell/Userprog.unbos hex/Userprog.unbos

deps: $(USERDEPS) bootdisk.im
bootdisk.im: $(USERPROGS)
	mkboot $(USERFILES)

shelldep:
	-cd shell; make;
catdep:
	-cd cat; make;
datadep:
hexdep:
	-cd hex; make;

allclean: clean
	-cd shell; make clean;
	-cd cat; make clean;
	-cd hex; make clean;

clean:
	rm -f bootdisk.im
```

```
/*   Userlink    1.2    92/08/31.12:19:02 */

MEMORY {
      user : o=0x400 l=16000000
}

SECTIONS {
      .text : { Userprog.oo(.text) }
      .data : { Userprog.oo(.data) }
      .bss  : { Userprog.oo(.bss)  }
}
```

```
#include <stdio.h>
#define BUFLEN        512
#define MAXFILES 100

int filelen[MAXFILES];

/*****************************************************************************/
main( argc, argv )
      int argc;
      char *argv[];
{
      FILE *in, *boot;
      char name[100];
      char buf[BUFLEN];
      char direntName[16];
      int bytes, i, imagebytes, pos;

      if (argc > MAXFILES) {
            printf("Too many files for boot disk.\n");
            exit(1);
      }
      if (argc < 2) {
            printf("At least one file must be specified");
            exit(1);
      }
      imagebytes = 0;
      for (i=1; i<argc; i++) {
            strcpy( name, argv[i] );
            strcat( name, "/Userprog.unbos" );
            if( (in = fopen( name, "r")) == NULL) {
                  printf("cannot open \"%s\".\n", name);
                  exit(1);
            }
            fseek( in, 0, 2 );
            filelen[i] = ftell( in );
            fclose( in );
            imagebytes += filelen[i];
      }
      if( (boot = fopen( "bootdisk.im", "w")) == NULL) {
            printf("cannot open \"%s\".\n", "bootdisk.im");
            exit(1);
      }
      pos = 8 + (argc-1)*24;
      bytes = pos + imagebytes;
      printf("Bootdisk total length = %d, ", bytes);
      putlong( boot, bytes );
      printf("Number of directory entries = %d\n", (argc-1));
      putlong( boot, (argc-1) );
      for (i=1; i<argc; i++) {
            printf("  Dirent=%d, name=%s, start=%d, length=%d\n", i,
                  argv[i], pos, filelen[i]);
            strcpy( direntName, argv[i] );
            fwrite( direntName, 1, 16, boot );
            putlong( boot, pos );
            putlong( boot, filelen[i] );
            pos += filelen[i];
```

```
        }
        for (i=1; i<argc; i++) {
                strcpy( name, argv[i] );
                strcat( name, "/Userprog.unbos" );
                in = fopen( name, "r" );
                while( (bytes=fread(buf, 1, BUFLEN, in)) > 0) {
                        fwrite( buf, 1, bytes, boot );
                }
                fclose( in );
        }
        fclose( boot );
}

/*****************************************************************************/
putlong( file, value )
        FILE *file;
        int value;
{
        unsigned char c;

        c = (value) % 256;
        fputc( c, file );
        c = (value / 256) % 256;
        fputc( c, file );
        c = (value / 65536) % 256;
        fputc( c, file );
        c = (value / 16777216) % 256;
        fputc( c, file );
}
```

```
#include <stdio.h>
#define BUFLEN        512

/*****************************************************************************/
main( argc, argv )
      int argc;
      char *argv[];
{
      FILE *in, *out;
      char name[100];
      char buf[BUFLEN];
      int entryPoint, textlen, datalen, bsslen, imagelen;
      int bytes;

      if (argc != 2) {
              printf("wrong number of parameters.\n");
              exit(1);
      }

      /* get this information from the .co file */
      strcpy( name, argv[1] );
      strcat( name, "/Userprog.co" );
      if( (in = fopen( name, "r")) == NULL) {
              printf("cannot open \"%s\".\n", name);
              exit(1);
      }
      fseek( in, 20+4, 0 );
      textlen = getlong( in );
      datalen = getlong( in );
      bsslen  = getlong( in );
      fseek( in, 20+24, 0 );
      entryPoint = getlong( in );
      fclose( in );
      printf("EntryPoint=0x%x, TextLen=%d, DataLen=%d, BssLen=%d\n",
              entryPoint, textlen, datalen, bsslen);

      /* copy the .im file */
      strcpy( name, argv[1] );
      strcat( name, "/Userprog.im" );
      if( (in = fopen( name, "r")) == NULL) {
              printf("cannot open \"%s\".\n", name);
              exit(1);
      }
      fseek( in, 0, 2 );
      imagelen = ftell( in );
      fseek( in, 0, 0 );
      strcpy( name, argv[1] );
      strcat( name, "/Userprog.unbos" );
      if( (out = fopen( name, "w")) == NULL) {
              printf("cannot open \"%s\".\n", name);
              exit(1);
      }

      putlong( out, 666 );
      putlong( out, entryPoint );
      putlong( out, textlen );
```

```
        putlong( out, datalen );
        putlong( out, bsslen );
        while( (bytes=fread(buf, 1, BUFLEN, in)) > 0) {
                fwrite( buf, 1, bytes, out );
        }
        fclose( out );
        fclose( in );
}

/*****************************************************************************/
putlong( file, value )
        FILE *file;
        int value;
{
        unsigned char c;

        c = (value) % 256;
        fputc( c, file );
        c = (value / 256) % 256;
        fputc( c, file );
        c = (value / 65536) % 256;
        fputc( c, file );
        c = (value / 16777216) % 256;
        fputc( c, file );
}

/*****************************************************************************/
int getlong( file )
        FILE *file;
{
        unsigned char c1, c2, c3, c4;

        c1 = fgetc( file );
        c2 = fgetc( file );
        c3 = fgetc( file );
        c4 = fgetc( file );
        return( c4*16777216 + c3*65536 + c2*256 + c1 );
}
```

```
CC    = /usr/bin/scc
CFLAGS        = -I../../h
H     = ../../h
COREBLD       = /unb1/cs4605/cmd/corebld
LD    = /bin/ld
XLINK = ../Userlink
XLINKDEP= ../Userlink
LIBS  = ../../estdio/stdio.a ../../syslib/syslib.a ../../strlib/strlib.a

OBJFILES = cat.o

Userprog.unbos: Userprog.im Userprog.co
      -cd ..; mkexec cat;
Userprog.im: $(XLINKDEP) $(LIBS) Userprog.oo
      $(LD) Userprog.oo $(LIBS) $(XLINK) -e _main -o Userprog.co
      $(COREBLD) <Userprog.co >Userprog.im
Userprog.oo: $(OBJFILES)
      $(LD) $(OBJFILES) -r -o Userprog.oo

cat.o: cat.c $H/stdio.h

clean:
      rm -rf *.o core a.out *.co *.oo *.im Userprog.unbos
```

```
/*    cat.c    1.2    92/08/31    12:21:36    */

#include <stdio.h>

#define DATABUF_LEN   1024
char databuf[DATABUF_LEN];

/*****************************************************************************/
int main( argc, argv )
      int argc;
      char *argv[];
{
      FILE *f;
      int i;
      int fd, bytes;

      if (argc < 2) {
              CopyFile( 0, 1 );
      } else {
              for (i=1; i<argc; i++) {
                      fd = open( argv[i], 0 );
                      if( fd >= 0) {
                              CopyFile( fd, 1 );
                              close( fd );
                      } else {
                              fprintf(stderr, "cannot open file \"%s\"\n",
                                  argv[i]);
                      }
              }
      }
}

/*****************************************************************************/
int CopyFile( fdin, fdout )
      int fdin, fdout;
{
      int bytes;
      while( (bytes=read( fdin, databuf, DATABUF_LEN )) >0 ) {
              write( fdout, databuf, bytes );
      }
}
```

```
Hello there
This is some user data.
Have a nice day.
```

```
CC    = /usr/bin/scc
CFLAGS        = -I,./../h
H     = ../../h
COREBLD       = /unb1/cs4605/cmd/corebld
LD    = /bin/ld
XLINK = ../Userlink
XLINKDEP= ../Userlink
LIBS  = ../../estdio/stdio.a ../../syslib/syslib.a ../../strlib/strlib.a

OBJFILES = hex.o

Userprog.unbos: Userprog.im Userprog.co
      -cd ..; mkexec hex;
Userprog.im: $(XLINKDEP) $(LIBS) Userprog.oo
      $(LD) Userprog.oo $(LIBS) $(XLINK) -e _main -o Userprog.co
      $(COREBLD) <Userprog.co >Userprog.im
Userprog.oo: $(OBJFILES)
      $(LD) $(OBJFILES) -r -o Userprog.oo

clean:
      rm -rf *.o core a.out *.co *.oo *.im Userprog.unbos

hex.o: hex.c $H/stdio.h
```

```
#include <stdio.h>

void  main();
void  hexify();
void  puthex();

char  hexits[16] = {'0', '1', '2', '3', '4', '5', '6', '7',
                    '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'};

void main( argc, argv )
      int     argc;
      char   *argv[];
{
      int     i;
      FILE   *fin;

      if( argc == 1 ) {
              hexify( stdin );
      } else {
              for( i=1; i<argc; i++ ) {
                      if( (fin = fopen(argv[i], "r")) != NULL ) {
                              hexify( fin );
                              fclose( fin );
                      }
              }
      }
}

void hexify( fin )
      FILE   *fin;
{
      int     buf[80];
      int     buflen, filepos, c, i, chksum;

      filepos = 0;
      c = 1;
      while (c >= 0) {
              buflen = 0;
              while (buflen < 32 && c >= 0) {
                      c = getc( fin );
                      if (c >= 0) {
                              buf[buflen] = c;
                              buflen ++;
                      }
              }
              if (buflen > 0) {
                      puthex( (filepos / 65536) % 256 );
                      puthex( (filepos / 256) % 256 );
                      puthex( filepos % 256 );
                      putchar(':');
                      chksum = 0;
                      for (i=0; i<buflen; i++) {
                              chksum += buf[i];
                              puthex( buf[i] );
                      }
                      putchar(':');
```

```
                        puthex( chksum % 256 );
                        putchar('\n');
                        filepos += buflen;
                }
        }
}

void puthex( byte )
        int byte;
{
        putchar( hexits[byte/16] );
        putchar( hexits[byte%16] );
}
```

```
CC     = /usr/bin/scc
CFLAGS         = -I../../h
H      = ../../h
COREBLD        = /unb1/cs4605/cmd/corebld
LD     = /bin/ld
XLINK = ../Userlink
XLINKDEP= ../Userlink
LIBS  = ../../estdio/stdio.a ../../syslib/syslib.a ../../strlib/strlib.a

.c.o:
	$(CC) $(CFLAGS) -c $<

OBJFILES = shell.o builtin.o disk.o external.o

Userprog.unbos: Userprog.im
	-cd ..; mkexec shell;
Userprog.im: $(XLINKDEP) $(LIBS) Userprog.oo
	$(LD) Userprog.oo $(LIBS) $(XLINK) -e _main -o Userprog.co
	$(COREBLD) <Userprog.co >Userprog.im
Userprog.oo: $(OBJFILES)
	$(LD) $(OBJFILES) -r -o Userprog.oo

builtin.o: builtin.c $H/stdio.h shell.h
disk.o: disk.c $H/stdio.h $H/kernel.h $H/server.h $H/kernserver.h \
	$H/diskserver.h
external.o: external.c $H/stdio.h shell.h $H/kernel.h $H/server.h \
	$H/kernserver.h
shell.o: shell.c $H/stdio.h shell.h

clean:
	rm -f *.o core a.out Userprog.oo Userprog.co Userprog.im Userprog.unbos
```

```
/*    builtin.c    1.2    92/08/31    12:23:21    */

#include <stdio.h>
#include "shell.h"

/*****************************************************************************/
int CmdEcho(argc, argv)
      int argc;
      char *argv[];
{
      int i;

      for (i=1; i<argc; i++) {
            printf(argv[i]);
            putchar(' ');
      }
      putchar('\n');
      return( ERR_OK );
}

/*****************************************************************************/
int CmdArgs(argc, argv)
      int argc;
      char *argv[];
{
      int i;

      printf("Argument Count = %d\n", argc);
      for (i=0; i<argc; i++) {
            printf("*argv[%d] = \"%s\".\n", i, argv[i]);
      }
      return( ERR_OK );
}

/*****************************************************************************/
int CmdExit(argc, argv)
      int argc;
      char *argv[];
{
      return( ERR_EXIT );
}

/*****************************************************************************/
int CmdBomb(argc, argv)
      int argc;
      char *argv[];
{
      StatMsg("shell: I am about to bomb on a NULL pointer fault!");
      * (char*) NULL = 'x';
      return( ERR_OK );
}

/*****************************************************************************/
int CmdCount(argc, argv)
      int argc;
      char *argv[];
```

```
{
        register int i;

        printf("Commensing to count to five million...\n");
        for( i=0; i<5000000; i++) ;
        printf("Finished counting to five million.\n");
        return( ERR_OK );
}
```

```
/*    disk.c    1.2    92/08/31    12:23:31    */

#include <stdio.h>
#undef NULL
#define ERR_OK        0

#include <kernel.h>
#include <server.h>
#include <kernserver.h>
#include <diskserver.h>

extern void SetKernMsg();
extern MsgBuffer *SendKernMsg();
extern Ulong atou();

MsgBuffer *msg = (MsgBuffer *) 0;
Port msgport;

/*****************************************************************************/
int CmdDisk(argc, argv)
      int argc;
      char *argv[];
{
      MsgBuffer *kernmsg;
      Port port;
      Ulong block;

      if (msg == (MsgBuffer *) 0) {
            SetKernMsg( &kernmsg, &port );
            kernmsg->operationCode = KS_CREATE_MSG_BUFFER;
            kernmsg->data[0] = 1024;
            kernmsg = SendKernMsg( kernmsg, port );
            msg = (MsgBuffer *) kernmsg->data[0];
            msgport = port;
            msg->replyPort = port;
            GetPortAlias( SERV_ALIAS_DISK_BLOCK, &(msg->capability.port) );
      }

      msg->destPort = msg->capability.port;
      msg->msgStartOffset = 512;
      msg->msgLength = 512;

      if (argc < 2) {
            printf("usage: disk cmd ...\n");
            return( ERR_OK );
      } else if (argc >= 3) {
            block = atou( argv[2] );
      } else {
            block = 0;
      }

      if (strcmp(argv[1], "format")==0) {
            msg->data[0] = (block==0 ? 1 : block);
            Doop( DS_FORMAT_DISK, "Disk format... " );
      } else if (strcmp(argv[1], "mount")==0) {
            Doop( DS_MOUNT_DISK, "Disk mount..." );
```

```
                printf("Disk block size=%d, 'starter' block=%d\n",
                        msg->data[0], msg->data[1]);
        } else if (strcmp(argv[1], "unmount")==0) {
                Doop( DS_UNMOUNT_DISK, "Disk unmount... ");
        } else if (strcmp(argv[1], "read")==0) {
                printf("Disk read block %d... ", block);
                msg->data[0] = 1;
                msg->data[1] = block;
                Doop( DS_READ_BLOCKS, "");
                DumpMsg();
        } else if (strcmp(argv[1], "write")==0) {
                printf("Disk write block %d with last read data... ", block);
                msg->data[0] = 1;
                msg->data[1] = block;
                Doop( DS_WRITE_BLOCKS, "" );
        } else if (strcmp(argv[1], "sync")==0) {
                Doop( DS_SYNC_DISK, "Disk sync... " );
        } else if (strcmp(argv[1], "allocate")==0) {
                printf("Disk allocate close to block %d... ", block);
                msg->data[0] = 1;
                msg->data[1] = block;
                Doop( DS_ALLOCATE_BLOCKS, "" );
                printf("got block %d\n", msg->data[1]);
        } else if (strcmp(argv[1], "free")==0) {
                printf("Disk free block %d... ", block);
                msg->data[0] = 1;
                msg->data[1] = block;
                Doop( DS_FREE_BLOCKS, "" );
        } else {
                printf("unrecognized disk command \"%s\"\n", argv[1]);
        }
        return( ERR_OK );
}

/*****************************************************************************/
Doop( op, comment )
        Ulong op;
        char *comment;
{
        printf(comment);
        fflush( stdout );
        msg->operationCode = op;
        Send( msg );
        msg = Receive( msgport );
        printf("Done - err=%i\n", msg->errorCode);
}

/*****************************************************************************/
DumpMsg()
{
        register int i, j;
        register Uchar c, *cp;

        for (i=0; i<512; i+=32) {
                printf("%3x: ", i);
                for (j=0; j<32; j++) {
```

```
                        cp=(Uchar *) ((Ulong)msg+msg->msgStartOffset+i+j);
                        c = *cp;
                        if (c>=32 && c<127) {
                                putchar(c);
                                putchar(' ');
                        } else {
                                printf("%2x", c);
                        }
                }
                putchar('\n');
        }
}
```

```
/*    external.c    1.2    92/08/31    12:23:41    */

#include <stdio.h>
#include "shell.h"

#undef NULL
#undef TRUE
#undef FALSE
#include <kernel.h>
#include <server.h>
#include <kernserver.h>
Capability fcap;

/*****************************************************************************/
ExecuteExternal(argc, argv)
      int argc;
      char *argv[];
{
      int err;

      fcap = KS_AMSG_PTR->filetab[0].fileCap;

      err = fexec(argv[0], argc, argv, NULL, fcap, fcap, fcap,
            /*priority*/20, /*async*/0);
      return( ERR_OK /*err*/ );
}
```

```
/*    shell.c    1.2    92/08/31    12:23:48    */

#include <stdio.h>
#include "shell.h"

#define PROMPT          "Yes Sir> "
#define MAX_CMD_LEN     256
#define CMD_BUF_LEN     1024
#define MAX_CMD_ARGS    32

char command[MAX_CMD_LEN];
char cmdbuf[CMD_BUF_LEN];
char *cmdargv[MAX_CMD_ARGS];
int cmdargc;

extern int      CmdExit();
extern int      CmdEcho();
extern int      CmdArgs();
extern int      CmdDisk();
extern int      CmdBomb();
extern int      CmdCount();

typedef struct {
        char    *name;
        int      (*func)();
        char    *desc;
} BuiltinCmd;

BuiltinCmd builtinCmds[] = {
        "exit", CmdExit, "exit from shell",
        "echo", CmdEcho, "display the given arguments on a line",
        "args", CmdArgs, "display the given arguments",
        "disk", CmdDisk, "execute a disk command",
        "bomb", CmdBomb, "bomb on a page fault",
        "count",CmdCount,"count to five million"
};

#define BUILTIN_COUNT ( sizeof(builtinCmds) / sizeof(BuiltinCmd) )

int     main();

/*****************************************************************************/
int main()
{
        int c;
        long cmdlen;
        int err;

        printf("\n======== UNBOS Mini Command Shell 1.00.00 ========\n");
        while( 1 ) {
                printf( PROMPT );
                fflush( stdout );
                cmdlen = 0;
                while( (c=getchar()) != '\n' && c != EOF ) {
                        command[cmdlen] = c;
                        cmdlen++;
```

```
                }
                command[cmdlen] = '\0';
                err = Execute( command );
                if (err == ERR_EXIT || c == EOF) break;
                if (err != ERR_OK) {
                        printf("*** Error code %d ***\n");
                }
        }
        printf("======== Exit from UNBOS Mini Command Shell ========\n");
}

/*****************************************************************************/
int Execute(command)
        char *command;
{
        int cmdno, argc, err;
        char *cmdGiven;

        err = Parse( command );
        if (err != ERR_OK) return( err );
        if (cmdargc == 0) return( ERR_OK );
        cmdGiven = command;
        for (cmdno=0; cmdno<BUILTIN_COUNT; cmdno++) {
                if( strcmp(command, builtinCmds[cmdno].name)==0 ) break;
        }
        if (cmdno < BUILTIN_COUNT) {
                err = (builtinCmds[cmdno].func)( cmdargc, cmdargv );
        } else {
                err = ExecuteExternal( cmdargc, cmdargv );
        }
        return( err );
}

/*****************************************************************************/
int Parse( command )
        char *command;
{
        int i;
        char c;
        BOOLEAN InChars, InQuotes, WildcardArg;

        cmdargc = 0;
        InChars = WildcardArg = InQuotes = FALSE;

        /* scan */
        for (i=0; (c=command[i]) != '\0'; i++) {
                if (InChars) {
                        if( ((c==' ' || c=='\t') && !InQuotes) ||
                            (c=='"' && InQuotes) ) {
                                command[i] = '\0';
                                if (WildcardArg) {
                                        /* match with filenames */
                                }
                                InChars = WildcardArg = InQuotes = FALSE;
                        } else {
                                if (c=='*' || c=='?' || c=='[' || c==']') {
```

```
                                        WildcardArg = TRUE;
                                }
                        }
                } else {
                        if (c=='"') {
                                cmdargv[cmdargc++] = &command[i+1];
                                InChars = InQuotes = TRUE;
                        } else if (c!=' ' && c!='\t') {
                                cmdargv[cmdargc++] = &command[i];
                                InChars = TRUE;
                        }
                }
        }

        /* check last argument */
        if (InChars && WildcardArg) {
                /* match with filenames */
        }
        return( ERR_OK );
}
```

```
/*    shell.h    1.2    92/08/31    12:24:07    */

#define ERR_OK                0
#define ERR_EXIT             -1

typedef unsigned long int BOOLEAN;
#define FALSE         0
#define TRUE          1
```