Miscellaneous System Services

Memory management support in the UNIX world allows a process to manipulate its address space dynamically under program control. The shared memory mechanism previously discussed is an example of this kind of service. Other mechanisms in this area include:

- The **brk()** call to adjust the size of a process global data segment to a fixed address boundary
- The **sbrk()** call to adjust the size of a process global data segment to a relative address boundary and return the new location
- The **mmap()** call to dynamically memory map an object from the file system. The object may be an ordinary file or the physical control registers of some system IO component if the device driver for the component supports the mmap() call

Miscellaneous System Services (Cont'd)

```
SYNOPSIS
       #include <unistd.h>

       int brk(void *endds);

   where:
       endds  The address of the first byte beyond
              the new end of the data area

   returns:   0 on success, or -1

SYNOPSIS
       #include <unistd.h>

       void *sbrk(int increment);

   where:
       increment   The signed increment by which
                   to change the data area size

   returns:        the previous break value on
                   success, or -1
}
```

Miscellaneous System Services (Cont'd)

brk() and sbrk()   (cont'd}

EXAMPLE:

```
   int addr_val;

   printf("current end of data partition is at %x\n",
                    addr_val = (int)sbrk(0));


   printf("expanding data partition by %d bytes\n",
                                    2000);


   if(brk(addr_val + 2000) == -1){
      perror("could not extend data partition");
   }else{
      printf("the new data partition limit is %x\n",
                                  (int)sbrk(0));
   }
```

Miscellaneous System Services (Cont'd)

SYNOPSIS

```
     #include <sys/types.h>
     #include <sys/mman.h>

     caddr_t mmap (addr, len, prot, flags, fd, off)
     caddr_t addr;
     size_t  len;
     int     prot;
     int     flags;
     int     fd;
     off_t   off;
```

where:

addr   is the optional starting address for the
new memory region to map.

len   is the length in bytes of the region to map.

prot   assigns the access attribute for the mapped
region: read, write, execute, a combination,
or no access; PROT_READ, PROT_READ, PROT_EXEC,
PROT_NONE

flags specify the mapping mode (shared or private),
and whether the requested address must be used
exactly; MAP_SHARED, MAP_PRIVATE, MAP_FIXED

fd   is the file descriptor of the memory object to
be mapped into the region.

off   is the offset into the file to be mapped into
the region.

Miscellaneous System Services (Cont’d)

EXAMPLE:

```
fd = open(...);
lseek(fd, off, SEEK_SET);
read(fd, buf, len);

/* change first byte to ascii ’a’ */

*buf = ’a’;
lseek(fd, off, SEEK_SET);
write(fd, buf, len);
fsync(fd);
```

Here is a rewrite using mmap(2):

```
fd = open(...);
pa = mmap((caddr_t)0, len, (PROT_READ|PROT_WRITE),
                            MAP_PRIVATE, fd, off);

/* change first byte to ascii ’a’ */

*pa = ’a’;
msync(pa, len, 0);
```

Miscellaneous System Services (Cont'd)

SYNOPSIS

```
#include <sys/types.h>
#include <sys/mman.h>

int msync(caddr_t addr, size_t len, int flags);
```

where:

addr   is the memory start region to update

len   is the number of bytes to update

flags is abit pattern built from the following values:

MS_ASYNC	perform asynchronous writes
MS_SYNC	perform synchronous writes
MS_INVALIDATE	invalidate mappings

returns: 0 on success, or -1