MMAP AND PIPE

UNIX Programming 2014 Fall by Euiseong Seo

Memory Mapping

- mmap(2) system call allows mapping of a file into process address space
- Instead of using read() and write(), just write to memory to transfer data
- You cannot extend a mapped file without use of other calls such as truncate(2)

mmap Call

- Prototype

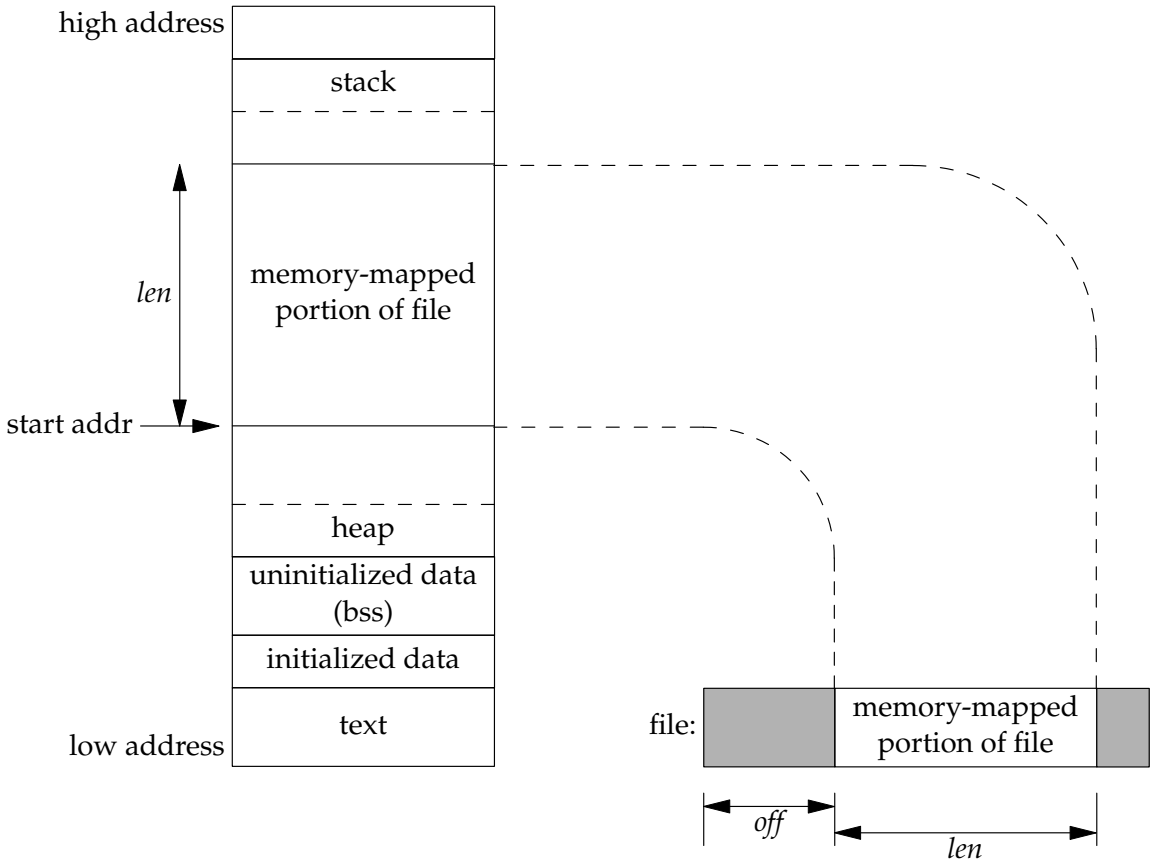
```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
int munmap(void *addr, size_t length);
```

  - addr - requested address, NULL to let system choose
  - len - length of mapped area
  - prot- PROT_READ for read, PROT_WRITE for write access
  - flags - MAP_SHARED save changes to disk, MAP_PRIVATE does not
  - fd - file descriptor of opened file to be mapped
  - off - offset from beginning of file to start mapping

Memory Mapping

- Address space after memory mapping of a file



Memory Mapping Example

```
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>


main(int argc, char *argv[])
{
    int fd;
    caddr_t addr;
    struct stat statbuf;
    if(argc != 2) {
        fprintf(stderr, "Usage:
mycat2 filename\n");
        exit(1);
    }
```

```
if(stat(argv[1], &statbuf) == -1)
    { perror("open"); exit(1); }

fd = open(argv[1], O_RDONLY);
if(fd == -1) {perror("open"); exit(1); }

    addr = mmap((caddr_t)NULL,
statbuf.st_size, PROT_READ, MAP_PRIVATE,
fd, (off_t) 0);

    if(addr == (caddr_t)-1)
        { perror("mmap"); exit(1); }
    /* no longer need fd */
    close(fd);
    write(1, addr, statbuf.st_size);
    return 0;
}
```

Extension of a Mapped File

- You cannot extend a mapped file
- If you want to extend a mapped file, increase file size by truncate(2) or ftruncate(3), and mmap() the newly increased region

Extension of a Mapped File

```
main()
{
    int fd;
    int pagesize =
sysconf(_SC_PAGESIZE);
    caddr_t addr;

    printf("Page size is %d bytes.
\n", pagesize);
    fd=open("mapfile", O_RDWR|
O_CREAT|O_TRUNC, 0666);
    if(fd == -1) {
        perror("open");
        exit(1);
    }
    if(ftruncate(fd,(off_t)
(6*pagesize)) == -1) {
        perror("ftruncate");
        exit(1);
    }
```

```
 system("ls -l mapfile");
    addr = mmap((caddr_t) NULL,
pagesize, PROT_READ|PROT_WRITE,
MAP_SHARED, fd, (off_t)0);
    if(addr == (caddr_t)-1) {
        perror("mmap");
        exit(1);
    }
    close(fd);
    (void)strcpy(addr, "Test
string.\n");
    system("head -1 mapfile");
    return 0;
}
```

Advising System on Access Patterns

- If you know how program will be accessing mmaped data, you may be able to improve performance by informing VM system with madvise(3)
- Prototype

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

int madvise(void *addr, size_t length, int advice);
```

Advising System on Access Patterns

- advice is one of followings
  - MADV_NORMAL
  - MADV_RANDOM
    - No read-ahead
  - MADV_SEQUENTIAL
    - Read-ahead and drop-behind
  - MADV_WILLNEED
    - Read in now
  - MADV_DONTNEED
    - Free immediately

File Locking

- You can lock a portion of a file or an entire file for exclusive access by using lockf(3)
  - This is an advisory lock
  - An advisory lock requires all programs to use lockf() to enforce locking
- Prototype

```
#include <unistd.h>
int lockf(int fd, int cmd, off_t len);
```

- Locked section ranges from current offset to len
- If len is 0 then rest of file will be locked

File Locking

- cmd can be one of followings
  - F_LOCK
  - F_ULOCK
  - F_TEST
  - F_TLOCK

Using fcntl() to Lock Files

- fcntl(2) can do many things
  - Duplicating a FD
  - Changing FD flags
  - Advisory locking
  - Mandatory locking (Non-POSIX)
- Prototype

```
#include <unistd.h>
#include <fcntl.h>

int fcntl(int fd, int cmd, ... /* arg */ );
```

Using fcntl() to Lock Files

- cmd parameter may be
  - F_GETLK
  - F_SETLK (non-block)
  - F_SETLKW (block)
- struct flock

```
struct flock {
    ...
    short l_type;    /* Type of lock: F_RDLCK,
                        F_WRLCK, F_UNLCK */
    short l_whence;  /* How to interpret l_start:
                        SEEK_SET, SEEK_CUR, SEEK_END */
    off_t l_start;   /* Starting offset for lock */
    off_t l_len;     /* Number of bytes to lock */
    pid_t l_pid;     /* PID of process blocking our lock
                        (F_GETLK only) */
    ...
};
```

Anonymous Pipes

- Pipes are a one-way byte-stream connection between two processes
- Frequently they are used to connect STDIO of one process to STDOUT of another
- The easiest way to handle pipes is with popen(3) and pclose(3)
- Pipes are anonymous because they have no name in file system

Anonymous Pipes

- Prototype

```
#include <stdio.h>

FILE *popen(const char *command, const char *type);
int pclose(FILE *stream);
```

Anonymous Pipes

- Example

```
#include <stdio.h>
main()
{
    FILE *fp;
    int m;

    /* write your screen output through "more" */
    fp = popen("more", "w");
    if(fp==NULL)
        {fprintf(stderr, "popen failed.\n"); return 1;}
    for(m = 1 ; m<= 90; m++)
        fprintf(fp, "Lots and lots of stuff.\n");
    pclose(fp);
    return 0;
}
```

Anonymous Pipes

- A somewhat harder but more powerful way is to use pipe(2)
- pipe() fills in an array with two descriptors
  - p[0] is read-end
  - p[1] is write-end
- Prototype

```
#include <unistd.h>

int pipe(int pipefd[2]);
```

Anonymous Pipes

- After calling pipe(), process will fork() to create another process
- When a process forks, all file descriptors are duplicated
- Thus child inherits connections to both sides of pipe
- Child and parent must close sides of pipe that they are not interested in
- Writes to a pipe of less than PIPE_BUF size are guarnateed to be atomic

Anonymous Pipes

```
int p[2];
pid_t pid;
if(pipe(p) == -1) {
   perror("pipe");
   exit(1);
}
switch(pid = fork()) {
   case -1:
        perror("fork");
        exit(1);
        break;
   case 0: /* child */
        close(p[1]);
        if(p[0] != 0) {
            dup2(p[0],0);
            close(p[0]);
        }
        execlp("cat", (char
*)NULL);
        perror("execlp");
        exit(1);
        break;
```

```
   default: /* parent */
            break;
   }
   close(p[0]);

   write(p[1], "This is a message\n", 18);
   close(p[1]);
   if(waitpid(pid, (int *)NULL, 0) == -1)
   {
       perror("waitpid");
       exit(1);
   }
```