Files

Eric McCreath

What is a file?

- Information used by a computer system may be stored on a variety of storage mediums (magnetic disks, magnetic tapes, optical disks,

flash disks etc). However, as a programmer it much simpler to have one logical view/interface for this data.

- A file is a named collection of related information. Files consist of a sequence of bits, bytes, lines, or records. The information in a file is generally defined by its creator. There is numerous types of files. These include: text, source, object, executable, binary, compressed, graphics image, video, data base, etc...

- A file system provides a uniform logical view of this information.

- In Unix "Everything is a file". Hence it is important to be able to effectively use the c API for files.

Two main approaches

c provides two similar approaches for reading/writing files. These are:

- SYstem Calls - c provides set of libraries that give you direct assess to the underlining system calls. This doesn't provide much to the programmer, although, it gives the programer complete control of this resource. Most of these functions are in volumn 2 of the unix man pages. They include: open, read, write, ioctl, ...
- File streams - this provides a slightly higher level abstraction on files providing some buffering of the data. Most of these functions are in volume 3 of the unix man pages. They include: fopen, fread, fwrite, ...

We will mainly look at the first of these, however, the second approach is often used also.

open

The "open" system call will open (possibly create) a file or device. The function returns a "file descriptor" which is a small non-negative number which provides a reference to the opened file for latter system calls.

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

int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

The OS keeps a table of the files that each process has opened. Also permissions are checked during the opening of the file.

This table keeps track of a "file offset", the "file offset" is the current position within a file you are reading from or writing to.

close

Once your program has completed using a file it should close it. This will initiate the flushing of any data in the buffers. When a process exits all its open files will be closed automatically, however, it is good practice to close any files your program has opened.

```
#include <unistd.h>

int close(int fd);
```

read

The read system call reads a block of data from a file. By default this will start at the beginning of a file and proceed sequentially to the end of a file.

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
```

Your program provides a file descriptor, a pointer to a buffer to place the read data, and the size this buffer in bytes. Once the read has completed it will return the number of bytes sucessfully read and place into the buffer. This could be less than the size of the buffer.

Normally you will repeated read from a file until all the data is obtained.

read returns 0 at the end of file.

write

write will write to an opened file. This will normally grow the file size as more data is written to the file.

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
```

The parameters of this function are similar to that of read. However, the buffer contains the data to write.

truncate and ftruncate

Sometime you wish to resize a file you have open. truncate can make a file bigger or smaller.

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

int truncate(const char *path, off_t length);
int ftruncate(int fd, off_t length);
```

If the file is made bigger then the new part of the file is filled with zeros.

lseek

lseek lets you move the current file offset of the opened file. This enables you to access the data of a file using random access rather than the normal sequential access.

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

off_t lseek(int fd, off_t offset, int whence);
```

ioctl and stat

ioctl lets you control (or get information from) devices your program has opened. The exact control command you issue will depend on the device.

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

int ioctl(int d, int request, ...);
```

stat provides information about a particular file. So if you wanted to find the size of a file you could use stat.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int stat(const char *path, struct stat *buf);
int fstat(int fd, struct stat *buf);
int lstat(const char *path, struct stat *buf);
```

mmap

A file can be mapped into memory, this can be done with the 'mmap' function. 'mmap' enables the file's contents to be view and modified in normal memory. There is two basic versions of mmap:

- SHARED - where modifications to the memory of the mapped file are written back to the actual file, also other processes that maps the same file sees the same modifications.
- PRIVATE â€“ where the process has its own private copy of the file. Modifications are not written back to the file and other processes do not see any changes made to the memory by the process that made the private mmap (uses copy on write).

Exersizes

- Write a simple version of the "cp" program. It should take 2 file names on the command line and copy the first to the second. Compare the performance of your program to that of the normal "cp" command on big files.
- (extension) Modify the above to use mmap on both the source and destination files (hint use stat to find the size of the file and truncate to resize the destination file, and just use a memcpy to copy the data of the files over).