UNIX System Programming
Lecture 5: Working with Files

- **Outline**
  - File Descriptors
  - Read, Write, Open and Close
  - File Management
  - File Information

- **Reference**
  - BLP: Chapter 3
  - Man pages (Ubuntu: manpages-dev package, Cygwin: man-pages-posix)

Lecture 5: Working with Files
Introduction

- C, C++, Java, etc. each provide some means of file input/output. Each of these languages build upon the lower-level (system) file access routines provided by the OS.
- The system routines provided by UNIX for file/device I/O are: open, close, read, write, and ________. They are documented in section 2 of the man pages: man 2 write.

Lecture 5: Working with Files
Why use low-level routines?

- For portability you should use the file access methods in your language for basic I/O.
- The system routines give access to features not available in the language routines.
- Interprocess communication (______) requires that we work with the OS routines.
- Writing kernel code (device drivers) also requires knowledge of the low-level routines.

Lecture 5: Working with Files
File Descriptors

- Information about every open file is kept in a system-wide file table. (A file may be opened multiple times, there would then be multiple entries in the system file table.)
- Every process has a file table that contains indexes into the system file table.
- Indexes into the per-process file table are called file ________

**Diagram:**

- Per Process File Table
- System File Table

Lecture 5: Working with Files
File Descriptors

- Programs usually (automatically) have standard input, standard output, and standard error open. These are associated with file descriptors 0, 1, and 2 respectively.
- You can associate other file descriptors with other files and devices by using the ______ system routine.
- System I/O routines require a file descriptor as an argument in the routine call.
Lecture 5: Working with Files

Example – Byte I/O

```c
#include <unistd.h>

int main()
{
    char c;
    ssize_t nread, nwrite;

    while ((nread = read(0, &c, sizeof(c))) != -1) {
        if (nread == 0)
            break; // break on end-of-file
        if (nwrite = write(1, &c, nread)) == -1)
            break; // break on write error
    }

    return 0;
}
```

Example – Block I/O

```c
#include <unistd.h>

int main()
{
    const int blksize = 512;
    char c[blksize];
    ssize_t nread, nwrite;

    while ((nread = read(0, c, blksize)) != -1) {
        if (nread == 0)
            break; // break on end-of-file
        if (nwrite = write(1, c, nread)) == -1)
            break; // break on write error
    }

    return 0;
}
```

Lecture 5: Working with Files

Read/Write Notes

- There is no guarantee that `read` will read the number of bytes requested. (Fewer bytes will be read when near the end of the file. Also, only a single line of data is read when reading from a __________ not 512 bytes.) Notice how we make a request to write `nread` bytes and not `blksize` bytes.
- `write` may also write fewer bytes than requested, but this can usually be attributed to more serious problems.

Open and Close

- The open call is used to associate a file descriptor with a __________.

  ```c
  // Required header files
  #include <sys/types.h>
  #include <sys/stat.h>
  #include <fcntl.h>
  
  // open returns a file descriptor (an int)
  // Use this form
  int open(const char *pathname, int flags);
  // or this one.
  int open(const char *pathname, int flags,
           mode_t mode);
  ```

- The close call closes a file descriptor.

  ```c
  int close(int fd);
  ```

Lecture 5: Working with Files

Open and Close

- `fseek` is used to reposition the file _________ pointer associated with a file descriptor:

  ```c
  #include <sys/types.h>
  #include <unistd.h>
  off_t lseek(int fd, off_t offset, int whence);
  ```

- `offset` is the # of bytes to move. `whence` is either `SEEK_SET`, `SEEK_CUR`, or `SEEK_END` to move relative to the start of file, the current position, or the end of the file (offset may be negative). Returns offset from beginning or -1 (on error).
Lecture 5: Working with Files
Duplicating File Descriptors

- **dup** and **dup2** are used to duplicate a file descriptor. The duplicated descriptor will use the same file __________. These calls are useful for interprocess communication.

```c
#include <unistd.h>
int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

- They return the new file descriptor (or -1 on error).

Lecture 5: Working with Files
File Information

- **stat**, **fstat**, and **lstat** are used to obtain all file __________ (type, owner, group, permissions, times, # of links, etc):

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
int stat(const char *file_name,
         struct stat *buf);
int fstat(int filedes, struct stat *buf);
int lstat(const char *file_name,
          struct stat *buf);
```

Lecture 5: Working with Files
File Information

- **stat** and **lstat** differ in regard to __________. **stat** returns info on the file pointed to while **lstat** returns info on the symlink. **fstat** acts like **stat**.

- Each of the routines require a pointer to a **stat** structure. The header files define the **stat** structure and also define several useful macros. Refer to the **stat** man page.

- Refer to the **getstat.cpp** program.

Lecture 5: Working with Files
File Information

- There is also a **stat** command line __________ that will display file information:

```
$ stat /etc/passwd
File: '/etc/passwd'
  Size: 27034  Blocks: 56  IO Block: 4096  regular file
  Device: 801h/2049d  Inode: 19449943  Links: 1
  Access: (0644/-rw-r--r--)  Uid: (0/root)  Gid: (0/root)
```

Lecture 5: Working with Files
File Information

- The **chattr** command can change **attributes** on files on an Linux __________. The letters __________ select the new attributes for the files: append only (a), no dump (d), immutable (i), data journalling (j), no atime updates (A), synchronous directory updates (D), synchronous updates (S), and top of directory hierarchy (T). **chattr** requires administrative privileges. (See **lsattr** also.)

Lecture 5: Working with Files
Other File Related Routines

- **chmod(path, mode)** # change file perms
- **chown(path,owner,group)** # change file owner (root)
- **unlink(path)** # delete file, (decr # of links by 1)
- **link(path1, path2)** # create a hard link to a file
- **symlink(path1, path2)** # create a symbolic link
- **mkdir(path, mode)** # create a new directory
- **rmdir(path)** # delete directory
- **chdir(path)** # change process directory
- **getcwd(buffer, size)** # get current working directory