Unix System Programming - Chapter 4

Neal Nelson

The Evergreen State College

Apr 2010

USP Chapter 4 - Unix IO

- Sections 4.1, 4.2 - Unix read and write
- Section 4.3 - Unix open and close
- Section 4.6 - ISO C standard IO Library (fopen, fread, etc.)
- Section 4.7 - Filters and Redirection

Unix IO

- Unix device IO abstraction
- USP Section 4.2
- #include <unistd.h>
- open, close, read, write, ioctl
- uses file descriptors (type int)
- STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO

C stdlib IO

- Stream IO abstraction
- USP Section 4.6
- #include <stdio.h>
- fopen, fscanf, fprintf, fread, etc
- uses file pointers (type FILE)
- stdin, stdout, stderr
- should be OS independent

Unix Device read, write

- Devices are in the /dev directory
- Devices are represented by *special files* (vs regular files)
- Block special files (eg, disks)
- Character special files (eg, terminal)
- All devices use a uniform device abstraction (open, close, etc)
- Low level and lots of conditions and errors to check and handle
- Example read - see readline.c p95
- r_read, r_write from the restart library make programs simpler
- Example write - see copyfile.c p100 using r_read and r_write

Unix Device open, close

- open associates a file descriptor with a file or physical device
- pass a path string and a flags parameter
- flags parameter has access modes (read, write, append, block or no, etc)
- bitwise or together all the desired flags
- creating a new file requires a third parameter with permissions
- low level and lots of conditions and errors to check and handle
- Example - a whole copy program copyfilemain.c p106 using copyfile.c
- Oh yeah, don't forget to close or r_close

select and poll

- Sections 4.4, 4.5
- Handling input from multuple sources.
- select and poll functions

File Representation

- Unix IO uses file descriptors
- C stdio (fopen, fread, etc) uses FILE pointers
- Both are considered *handles* for device IO
- C stdio FILE pointers are one level abstracted from Unix file descriptors
- The USP discussion of file representation is a model (so details may differ on actual systems).
- Open associates a file or physical device with a handle used in a program

File Tables

- See Figure 4.2 p120
- File Descriptor Table - per-process, in user address space
- system file table - shared by all processes, entry for each active open, in kernel space.
- in-memory inode table - entry for each active file in the system, copies of the inodes on disk, in kernel space.
- system file table has file offsets, access modes, and the number of descriptor table entries pointing to it.
- An active file may be shared by distinct processes, but each process will have its own system file table entry.
- Open creates a file descriptor table entry pointing to an entry in the system file table.
- A fork causes parent and child to share a system file table entry and therefore share offsets. Hmmmm.

File Pointers and Buffering

- C stdio functions use file pointers not file descriptors.
- See Fig 4.3 p122. FILE structures are in user space and contain buffers and file descriptors.
- a file pointer is a handle pointing to a handle.
- C stdio buffering requires specific handling of buffer flushes (fflush).
- Unix IO no doubt has buffering of its own in kernel space, but I'll bet buffer flushing is only an issue in error or crashing circumstances. (See p124)
- **Stderr is not buffered!**

Forks and file descriptors

- Section 4.6.3 pp124-128. Interesting and detailed examples of the consequences of shared system file table entries.

Filters and Redirection - command line

- Command line redirection
- See Figure 4.6 p130.
- cat $>$ my.file replaces stdout file descriptor table entry with my.file

Filters and REdirection - C program

- See Figure 4.7 p131.
- The trick is to use dup2 system call for copying file descriptors.
- See example redirect.c p131 for how to do this in a C program.