Advanced File I/O

UNX511 Week 4 Class 1

John Sellens

May 27, 2025

Seneca Polytechnic

Outline

Bitwise Operations and Flags

Programming, Flags, and Bit Fields

- In programming, we often want to keep track of boolean flags
    - e.g. a person could be a club member, with gym access, and restaurant charge privileges – 3 yes/no flags
- We could use separate variables, or a one variable that holds all flags
- Yes/no, on/off flags or settings can be indicated by binary digits
- A 32 bit unsigned integer can hold 32 different boolean flags
- When used in this way, we call the variable a “bit field”
- We can use bit fields to pass a set of options to a function e.g. `open(2)`
- `https://en.wikipedia.org/wiki/Bit_field`

Bit Fields in C

- In C, the typical method is
  - Define numeric constants that set one bit per constant
  - Declare variables to contain bit fields
  - Use bitwise operators to combine or query bit fields
- Compare logical `&&` and `||` with bitwise `&` and `|`
- Also bitwise exclusive OR `^` and bitwise NOT `~`
- e.g. `open(file, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR)`
  - See defines in `/usr/include/asm-generic/fcntl.h`
- `https://en.wikipedia.org/wiki/Bitwise_operations_in_C`
- See example `week4_1/2_fcntl_race/fcntl.cpp`

Advanced File I/O

File Descriptor Manipulation and the Shell

- Recall that the shell (e.g. `bash`) provides I/O redirection and command pipes
  - And starts commands with stdin, stdout, stderr attached elsewhere
- To redirect output to a file, shell opens file, and runs the command with file descriptor 1 open to that file
- But when the shell `open()`ed the file, it didn't get file descriptor 1
- The functions `dup()` and `dup2()` allow you to copy file descriptors to new integers
  - `dup2()` lets you specify e.g. file descriptor 1
- The functions `pipe()` and `pipe2()` create pipes and return a file descriptor pair that can also be `dup()`ed
- A file descriptor can be `fdopen()`ed for stdio

File Offsets

- An open file descriptor has a related “file offset”
    - Basically where in the file the next read/write will happen
- Reading or writing the file advances the file offset
- You can manipulate the file offset with `lseek(2)` `fseek(3)` and `ftell(3)`
- `pread(2)` and `pwrite(2)` let you specify an offset
    - Saves a separate explicit `lseek(2)` call
- `readv(2)` and `writev(2)` let you read or write multiple buffers at a time

Code Examples

- `week4_1/1_fileDup` – introduction to `dup()`
- `week4_1/2_fcntl_race` – file writing with overlapping offsets
- `week4_1/3_offset` – reading at various offsets
- `week4_1/4_structures` – writing C structs

Summary

- Lots of details and functionality for reading and writing