OS 2/25

Reagan Shirk

February 25, 2020

Pipes

- We spent most of class going over sorting in so I'll try to remember to post the code
- Pipes usually work with parent and child processes
- The pipe is usually unidirectional
- A shared file descriptor that allows both processes to talk to each other
  - You can write in one file descriptor and read in another
- Look at figure in chapter 3

C File

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

typedef struct reports {
    int counts;
    const char * sickness;
} reports;

int compare(const void * left, const void *right) {

    const reports * a = (const reports *) left;
    const reports * b = (const reports *) right;

    if (a->counts > b->counts) {
        return -1;
    }
    else if(a->counts < b->counts) {
        return 1;
    }
    else {

        // return 0;
        // return -1;
        //return &left - &right;
        return &right - &left;
    }
}

int main(int argc, char **argv) {

    struct reports log[] ={
        {70, "heart ache"},
        {30, "sniffles"},
        {70, "fed up"},
        {100, "flu"},
        {50, "food poisoning"},
        {70, "miller lite virus"},
        {70, "corona virus"},
        {100, "headache"}
    };

    // Get the length of log
    // option 1: iterate until null and count
    // divide the output by the size
    int length = sizeof(log) / sizeof(log[0]);

    for (int i = 0; i < length; ++i) {
        dprintf(STDERR_FILENO, "%d>>%s\n", log[i].counts, log[i].sickness);
    }

    dprintf(STDERR_FILENO, "------------------\n");

    qsort(log, length, sizeof(reports), compare);

    for (int i = 0; i < length; ++i) {
        dprintf(STDERR_FILENO, "%d>>%s\n", log[i].counts, log[i].sickness);
    }

    exit(EXIT_SUCCESS);
}
```