System Software COP3402

Midterm Review

> - 13 Multiple Choice
> - 6 Short Answer
> - 1 Long Answer
> - Homework questions, Lecture Notes, Coding Exercises

File Systems

> - What makes UNIX File System hierarchical
>   - They are in trees and you can go to subdirectories.
>   - **Directories** are special files that store a table of **names** and **i_node** numbers
>   - Without the table there would be no files by name
>   - A subdirectory will also have its own table of names and i_node numbers
>   - Hierarchy comes from the directories being special files with tables so they can exist in other directory files
>   - **Root** does not have a parent directory, but all other files should
>   - Root's parent directory is just itself
> - Absolute vs Relative paths
>   - Absolute starts at **Root**
> - How are **parent directories** referenced in the file system
>   - It uses **..**
> - Draw File System Tree

Navigations

> - What is the working directory and how do you display it?
>   - The current directory for process that is executed
>   - Bash automatically shows the working directory when asking for a prompt
>   - Command to display is **pwd**
>   - Will return something like this in Eustis if you're in the root directory for your account.
>   - `/home/net/nid`
> - What is the unix standard command to rename a file?
>   - **mov**
>   - Move one file to move it to another named file, so it renames it
>   - `mov oldName.c newName.c`
>   - Will rename oldName.c to newName.c

> - Doesn't change the i_node number just the name
> - What is tab-completion?
>   - Will auto-fill a file or directory name that matches the already written characters
>   - Auto-correct for the command line
> - What unix standard will show you the text of a file?
>   - **cat**
>   - `cat newName.c`
>   - Will show all the text in the file.
> - What does **grep** do?
>   - Searches within a file that matches the pattern that is passed through.
>   - Pattern is the first input
>   - File name is the second input
>   - Is case-sensitive
>   - `grep textPattern fileName` -> `grep word newName.c`
>   - Will print out all the lines that have the substring "word" appear in them.
> - How do you change the working directory to your home directory?
>   - **cd**
>   - `cd` will take you back to your root/home directory
>   - `cd ..` will take you up one folder level
>   - `cd ./` will keep you in the same directory
> - What is the unix command to delete a file?
>   - **rm**
>   - `rm newFile.c` will delete the file newFile.c
> - How does the implementation of deleting a file work? Does it remove the file's contents from the storage medium?
>   - Delete the directory entry
>   - Free the space that is malloced so the system can use it again for other stuff

Processes and Advanced Processes

> - What is a process?
>   - A running program
> - How does one CPU core run multiple programs at the seemingly same time?
>   - Time sharing
> - Command to List files
>   - **ls**
>   - `ls` will automatically list out the current working directory, but you can pass a path and it

> will list out the files in that directory
>
> - `ls newDirectory` will list out files in that directory
>
> - Difference between cat and echo
>   - **echo** takes command line arguments and writes them to stdout
>   - **cat** takes stdin and writes them to stdout
> - How do you redirect standard (out, in) of bash command to a file? for instance, i want to redirect grep's (out, in) to the file grep.txt what do i type?
>   - **grep hello > grep.out**
>   - Takes in input from the command line
>   - Writes to a file called grep.out
>   - `grep word textFile.txt > grep.out`
>   - Will search through textFile.txt for lines with the substring "word" and write them in the file grep.out
> - How do you redirect standard out from one command to another command's standard in? for instance, let's say i want to count the results of find with wc, what do i type?
>   - Use a pipe
>   - **find /usr/include/ | grep stdio**
>   - **find** will get the file tree and pass it into the grep, and grep will search through each one to find where the substring "stdio" appears.
>   - `find ./ | grep new > grep.out`
>   - find will get the file tree (so all the files recursively as a path that are accessible)
>   - The pipe "|" will pass the find output to *grep* as the input text
>   - grep will search through the text to see if any lines contain the substring "new"
>   - grep will write to grep.out with any lines it finds
> - **Find** command
>   - Will print out all the files throughout the file tree starting from the path, or if not path is passed in then at the current working directory.
>   - `find` will return ./Directory, ./Directory/File and so on

Editor

> - How do you edit files or emacs (pick one)?
>   - **emacs** *filename*
> - How do you quit the editor or emacs (pick one)?
>   - **CTRL X + CRTL C**
> - How do you save?
>   - **CRTL X + CRTL S**

Build Automation

> - What does the (target, recipe, prerequisite) of a makefile rule do.
>   - **Target** is the file that is generated
>   - **Recipe** is the sequence of commands that run
>   - **Prerequisites** are the files that need to exist to run the recipe
>
>   > - `main: main.c square.c exponent.c`
>   >
>   > > - `gcc -o main main.c square.c exponent.c`
>
> - Do makefiles use spaces or tabs?
>   - Makefiles use tabs as indentation to have proper syntax.
> - By convention, what does the clean target do?
>   - Deletes the generated file
>
>   > - `clean:`
>   >
>   > > - `rm -f main`
>
> - Here is a Makefile, add a clean target to remove the binaries.
>
> > - `clean`
> >
> > > - `rm -f main`
> > > - Write a Makefile that will create a program called `hello` from two source files, `main.c` and `hello.c`, when `make` is run.
> >
> > - `hello: main.o hello.o`
> >
> > > - `gcc -o hello main.o hello.o`

Version Control

> - What git command copies commits from the local repository to the remote repository?
>   - git push
> - What git command copies commits from the remote repository to the local repository?
>   - git pull
> - What git command stages a new file?
>   - git add
> - What git command commits a commit?
>   - git commit
> - What git command creates a log of the change to a staged file to the local repository?
>   - git log -> Shows you the history
>   - git commit

File Syscalls

> - Using the open syscall (man 2 open, not fopen) to open a path given in the `string char *filepath` variable.
>   - `int fd = open(filepath, O_RDONLY);`
> - How do you check for and terminate the program on an error with opening a file?
>   - `if (fd == -1)` -> Error value is -1
> - Using the read syscall (man 2 read, not fopen), you already have an open file with the file descriptor stored in fd, read the first 200 bytes of the file and print it to stdout
>   - `read(fd, buf, BUFSIZE);` -> BUFSIZE should be 200 for this question and buf should have enough memory
>   - **read is not guaranteed to give you all the data**
>   - Do it in a loop to make sure you get everything
> - What syscall can you use to find the (size, number of hard-links) of a file?
>   - `stat(path)` syscall
> - What syscall can you use the find the name of a file? (Not on exam)
>   - `stat`
>   - Get i_node from stat then search the file system for a name that points to it.
> - Write a code snippet that will print all files in a given directory, except do not print the "." and ".." names
>   - fs project
> - Write all the file open macros.
>   - O_CREAT -> Creates file if it doesn't exist
>   - O_TRUNC -> Deletes the file and 0's it out if it exists
>   - O_RDONLY -> Opens the file for reading
>   - O_WRONLY -> Opens the file for writing

Process, Pipe, Syscalls

> - Write code that uses unix standard syscalls to create a new process that runs the ls command.
>   - Fork and exec
> - Does fork create a new program?
>   - It duplicates the current program
> - Write code that creates a new process, where the original process writes "parent" and the new process writes "child", both to stdout.

Example code for fork problem.

```
#include <stdio.h>
#include <unitstd.h>
int main(void) {
    pid_t pid = fork();
    if (pid < 0) {
        // fork() failed
        perror("fork failed");
        return 1;
    } else if (pid == 0) {
        // This is the child process
        printf("child\n");
    } else {
        // This is the parent process
        printf("parent\n");
    }

    return 0;
}
```

> - Write code that replaces the current processes running program with the stat/ls command (not the stat syscall).
>   - exec

Example Code for exec problem.

```
#include <unistd.h>
int main(void) {
    char *args[] = {"ls", "-l", NULL};
    execvp("ls", args);  // Searches PATH for "ls"
    return 1;            // Only reached if execvp() fails
}
```

> - Write a program that opens a pipe and reads to and writes from it.

Example code for pipe problem.

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

int main(void) {
    int fd[2];  // fd[0] = read end, fd[1] = write end
    pid_t pid;
    char message[] = "Hello from parent!";
    char buffer[100];

    // Create the pipe
    if (pipe(fd) == -1) {
        perror("pipe failed");
        exit(1);
    }

    // Create a new process
    pid = fork();
    if (pid < 0) {
        perror("fork failed");
        exit(1);
    }

    if (pid == 0) {
        // Child process
        close(fd[1]); // Close unused write end
        read(fd[0], buffer, sizeof(buffer));
        printf("Child received: %s\n", buffer);
        close(fd[0]);
    } else {
        // Parent process
        close(fd[0]); // Close unused read end
        write(fd[1], message, strlen(message) + 1);
        close(fd[1]);
    }

    return 0;
}
```

> - Write a program that redirects the standard output to a file called "output.txt".
>   - dup_2 to copy a file descriptor to file IO and redirect

Example code for dup_2 problem.

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

int main(void) {
    int fd;

    // Open (or create) output.txt for writing
    fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd < 0) {
        perror("open failed");
        exit(1);
    }

    // Redirect stdout (file descriptor 1) to the file
    if (dup2(fd, STDOUT_FILENO) < 0) {
        perror("dup2 failed");
        close(fd);
        exit(1);
    }

    close(fd); // fd no longer needed after duplication

    // Now, anything printed with printf goes to output.txt
    printf("This text will be written to output.txt\n");
    printf("dup2() successfully redirected stdout!\n");

    return 0;
}
```

> - Fork Stuff
>   - Fork returns 0 for a new process
>   - Fork returns > 0 for original process

Fork Template

```
enter int pid;
  switch (pid = fork()) {
case -1:
    perror("fork");
    exit(EXIT_FAILURE);
    break;
case 0: // child
    break;
default: // parent
    break;
}
```

Pipe Example

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>

int main(int argc, char **argv) {
    int pipefd[2];
    char buf;
    char *msg;
    pid_t pid;

    if (argc < 2) {
        printf("USAGE: %s message_to_send\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    msg = argv[1];

    if (pipe(pipefd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    pid = fork();
    switch (pid) {
        case -1:
            perror("fork");
            exit(EXIT_FAILURE);
            break;

        case 0: // child
            close(pipefd[1]); // close write end (parent does writing)
            sleep(3);

            while (read(pipefd[0], &buf, 1) > 0) {
                write(STDOUT_FILENO, &buf, 1);
            }

            write(STDOUT_FILENO, "\n", 1);
            close(pipefd[0]);
            exit(EXIT_SUCCESS);
            break;

        default: // parent
            close(pipefd[0]); // close read end (child does reading)
            write(pipefd[1], argv[1], strlen(argv[1]));

                // it's very important to close the write end of the pipe so
                // that the read end will know to stop reading
            close(pipefd[1]);
            wait(NULL);

            sleep(3);
            exit(EXIT_SUCCESS);
            break;
    }
}
```