Midterm Notes

David Robinson

File systems

What makes the unix file system "hierarchical"?

It organizes files and directories in a tree-like structure, where there is a root directory and each directory contains a list of other directories and files.

What is the difference between absolute vs. relative paths?

Absolute paths start from the root directory while relative paths start from the working directory.

How are parent directories referenced in the file system?

Parent directories are referenced using `..` in the filepath.

Navigation

What is the working directory and how do you display it?

The working directory is the directory that a program is currently in and can be displayed with the `pwd` command.

What is the unix standard command to rename a file?

```
mv old-file-name new-file-name
```

What is tab-completion?

Tab completion is a feature that automatically completes commands and file/directory.

What unix standard will show you the text of a file?

```
cat filename
```

What does grep do?

`grep` searches through text for a phrase. For example, `grep "this phrase" filename` will search for "this phrase" in the file. If a filename is not provided, it will search through standard in.

How do you change the working directory to your home directory?

```
cd
```

What is the unix command to delete a file?

```
rm filename
```

How does the implementation of deleting a file work? Does it remove the file's contents from the storage medium?

When you delete a file using `rm`, the file's metadata, including its directory entry in the parent directory, is removed, but the data on disk remains until it is overwritten by new data.

Processes

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?

Use the `>` or `<` command: `grep "pattern" filename > grep.txt`

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 the pipe `|` command: `ls | wc`

Editor

How do you edit files in vim or emacs (pick one)?

```
vim filename
```

How do you quit the editor in vim or emacs (pick one)?

```
:wq
```

Build automation

What does the (target, recipe, prerequisite) of a makefile rule do.

Target is the output that the Makefile rule produces, the prerequisites are the files required to build the target, and the recipe includes the commands to execute to build the target

By convention, what does the clean target do?

The clean target removes any generated files.

Here is a makefile, add a clean target to remove the binaries.

```
clean:
    rm -f *.o target_name
```

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.c hello.c
    gcc -o hello main.c hello.c
```

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 filename
```

What git command creates a log of the change to a staged file to the local repository?

```
git commit -m "commit message"
```

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_RDWR);
```

How do you check for and terminate the program on an error with opening a file?

```
if (fd == -1) {
    perror("open");
    exit(EXIT_FAILURE);
}
```

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

```
char buffer[200];
int bytes = read(fd, buffer, 200);
if (bytes == -1) {
    perror("read");
    exit(EXIT_FAILURE);
}
for (int i = 0; i < bytes; i++) {
    printf("%c", buffer[i]);
}
```

Write a program that uses the opendir/readdir syscalls to list the names of files in the current working directory.

```
#include <dirent.h>
#include <stdio.h>

int main() {
    DIR *dirp = opendir(".");
    if (dirp == NULL) {
        perror("opendir");
        exit(EXIT_FAILURE);
    }

    struct dirent *curdir;

    while ((curdir = readdir(dirp)) != NULL)
        printf("%s\n", curdir->d_name);

    if (closedir(dirp) == -1) {
        perror("closedir");
        exit(EXIT_FAILURE);
    }

    return 0;
}
```

What syscall can you use to find the (size, number of hard-links) of a file?

```
stat
```

What syscall can you use the find the name of a file?

```
readdir
```

Process, pipe, syscalls

Write code that uses unix standard syscalls to create a new process that runs the ls command.

```
pid_t pid;
switch (pid = fork()) {
    case -1:
        perror("fork");
        exit(EXIT_FAILURE);
        break;
    case 0:
        if (execlp("ls", "ls", NULL) == -1) {
            perror("execlp");
            exit(EXIT_FAILURE);
        }
        break;
    default:
        break;
}
```

Write code that creates a new process, where the original process writes "parent" and the new process writes "child", both to stdout.

```
pid_t pid;
switch (pid = fork()) {
    case -1:
        perror("fork");
        exit(EXIT_FAILURE);
        break;
    case 0:
        puts("child");
        sleep(10);
        _exit(EXIT_SUCCESS);
        break;
    default:
        printf("parent\n");
        sleep(10);
        exit(EXIT_SUCCESS);
        break;
}
```

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

```
execlp("ls", "ls", NULL);
```

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

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

int main() {
    int pipefd[2];
    char buffer[8];

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

    write(pipefd[1], "message", 7);
    read(pipefd[0], buffer, 7);
    buffer[7] = '\0';
    printf("%s\n", buffer);

    close(pipefd[0]);
    close(pipefd[1]);
    return 0;
}
```

Write a program that redirects the standard output to a file called "output.txt".

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

int main() {
    int fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd < 0) {
        perror("open");
        exit(EXIT_FAILURE);
    }
    if (dup2(fd, STDOUT_FILENO) < 0) {
        perror("dup2");
        close(fd);
        exit(EXIT_FAILURE);
    }
    close(fd);
    return 0;
}
```

Common Syscalls in C

- `int open(const char *pathname, int flags);`

  Opens a file specified by `pathname` and returns a file descriptor.

- `int close(int fd);`

  Closes the file descriptor `fd`, freeing the resources associated with it.

- `ssize_t read(int fd, void *buf, size_t count);`

  Reads up to `count` bytes from the file descriptor `fd` into the buffer `buf`.

- `ssize_t write(int fd, const void *buf, size_t count);`

  Writes up to `count` bytes from the buffer `buf` to the file descriptor `fd`.

- `DIR *opendir(const char *name);`

  Opens the directory named `name` and returns a pointer to the directory stream.

- `struct dirent *readdir(DIR *dirp);`

  Reads the next directory entry from the directory stream `dirp`.

- `int closedir(DIR *dirp);`

  Closes the directory stream `dirp`.

- `int dup2(int oldfd, int newfd);`

  Duplicates the file descriptor `oldfd` to the descriptor `newfd`, closing `newfd` first if necessary.

- `pid_t fork(void);`

  Creates a new process by duplicating the calling process, returning 0 to the child and the child's PID to the parent.

- `pid_t wait(int *wstatus);`

  Waits for the termination of a child process, returning its PID and storing its status in `wstatus`.

- `int execvp(const char *file, char *const argv[]);`

  Replaces the current process image with a new process image specified by `file` and arguments `argv`.

- `int pipe(int pipefd[2]);`

  Creates a unidirectional data channel (pipe) with one end for reading and one for writing, stored in `pipefd`.

- `int stat(const char *path, struct stat *buf);`

  Retrieves file information (like size, permissions) for the file at `path` and stores it in `buf`.

- `int lseek(int fd, off_t offset, int whence);`

  Repositions the file offset of the open file descriptor `fd` to `offset` based on `whence`.

- `int chmod(const char *pathname, mode_t mode);`

  Changes the permissions of the file at `pathname` to the mode specified by `mode`.

- `int unlink(const char *pathname);`

  Deletes the file specified by `pathname` by removing its directory entry.