2018 Systems Programming Final Review:

- Bash History DONE
- Bash Commands & Options DONE
- VIM Commands DONE
- C vs C++ DONE
- Regex vs. Filename Expansion DONE
- Tarball DONE
- Command Substitution DONE
- Globbing
- Scripting
- Two Types of Returns DONE
- Relative and Absolute Paths DONE
- Expansion Suppression
- Arithmetic In Bash
- Soft and Hard Links DONE
- Piping
- Input Redirection
- System Calls vs Stdio
- Forking DONE
- Piping for Communication
- Signals
- Execute Family
- Make
- Debuggers
- Threads DONE
- Semaphores
- Extra Stuff

BASH HISTORY:

unix = developed in 1970's at Bell Labs

- Bell Labs = most prestigious place to work in 1970's
- needed a way to interact with computers
- soon after went to FAX machine
- was developed as multi-user multi-tasking operating system = opposite of windows at time
  - Multi-tasking does NOT mean concurrent run time, just appearance of so

- People behind unix were Dennis Ritchie, Kim Thompson, and Brian Kernighan
- The philosophy of unix was based on 5 principles:
  1. Simplicity - KISS (keep small and simple)
  2. Focus - make one thing really well instead of many eh quality
  3. Filters - (piping) one applied as filter to transform input
  4. Open File Format - well published format so apps can read and interpret
  5. Flexibility - don't limit the users of the system

C Language = developed at the same time as unix in the same place (concurrently)

- Developed by Dennis Ritchie
- became popular because it was portable

Linux

- You needed a license in order to use unix which was very expensive

- other products coming out so lots of lawsuits
- an undergrad @ Helsinki, named Linus Torvalds wrote own OS for only one processor, which was the x86 processor.
- linux is pretty much identical to unix, so much so that many exe will run on both
- freely distributed
- the major event that helped linux take off was the GNU project
  - "GNU's not unix" is what GNU stands for
  - produced gcc and g++ (GNU compilers) and gdb (debugger)
  - setup to develop different apps for linux
  - gpl = GNU public license (write whatever but have to provide source code)
- Kernel - we don't normally interact with it. Essentially the nucleus. Essential center of a computer operating system.
- linux distributions = typically the kernel packaged with utilities that work with kernel. doesn't come with bash. (examples below)
  - Ubuntu
  - Fedora
  - Madriva
- Windows has no kernel
- executables = instructions that can be run by a computer
- shell = find program we want to execute
- all bash does is find and execute the program you type in
- path variable changes way to look
- common paths in linux

- /bin	- user/bin
- users/bin	- user/local/bin

- Linux apps are represented by two types of files
  - executables = programs can be run directly by computer (.exe)
  - scripts = collections of instructions for another program, an interpreter, to follow (.bat .cmd)
  - Linux does not require that either of these have a specific extension

MORE

- VI released in 1991
- VIM = VI improved
- EMACS available in 1976
- standardizations of C = C89, GNU89
- C libraries = pre compiled pieces of code introduced in linker stage. Package of multiple pre compiled codes.
- DRAWING OF KERNEL AND SHELL STUFF FROM NOTES

- shell gives us interactions with kernel
- shell used for maintenance and configuration
- shell script inefficient and slow
- shells
  - bash = bourne again shell
  - sh = bourne shell (original)

BASH COMMANDS AND OPTIONS:

pwd = report the present working directory (current directory)

ls = List all the files in the current directory. Usage: ls or ls path
    -l = long listing, with info about size, permissions, ownership, etc…
    -a = List all files, including those that start with a dot
    -R = recursively list all files in subdirectories
    -F = put a character at the end of filename to indicate its type

cd = change the current directory. Usage: cd path

mkdir = make a directory. Usage: mkdir path

rm = remove all files and directories. Usage: rm path
    -r = recursively remove files and directories
    -i = interactive mode, prompt before removing
    - - = interpret all subsequent parameter as file names rather than options. Useful for deleting a file named something like "-r"

cp = Copy one or more files. Usage: cp old_file new_file OR cp file1 file2 … directory
    -r = recursively copy directories

mv = Move files to new filenames or directories. Usage: mv old_name new_name OR cp file directory
    -i = interactive mode, prompt before overwriting anything

cat = Read one or more files and write them out one after another to standard output (often used as a quick way to look at contents of a small file). Usage : cat file OR cat file1 file2 file3

who = report who is logged in.

exit = tell your shell to exit (logging you out of telnet or secure shell connection)

finger = Give information about a user ;). Usage: finger username

date = report what time the system thinks it is.

more = Display the contents of a file to the user one page at a time. Pressing space will let you go on to the next page. Usage: more file

less = Like more but better. Usage: less file

man = Provide online documentation for Unix commands, system calls, configuration files, and other features. Usage: man command_name
    -S n = Look for documentation in section n of the manual. Section 1 is for shell commands, section 2 is for system calls, section 3 is for other functions.
    -k words = look for man pages about the given key-words

head = print the first ten lines of each file parameter (or from stdin if no parameter is given). Usage: head file
    -n num = print the first num lines of the file

tail = print the last ten lines of each file parameter (or from stdin if no parameter is given).

Usage: tail file
-n num = print the last num lines of the file

touch = bring the modification time of a file up to the current time. Also, create an empty file if it doesn't already exist. Usage: touch file

ps = show currently running processes
-e = report on every process
-H = give hierarchical listing of parent/child processed
-l = give a long listing

top = like p, but give a continuously updating report

jobs = show all processes that your shell is keeping up with along with their job numbers (shell builtin)

bg = move a suspended process into the background (shel builtin). Usage: bg job_number

fg = move a background or suspended job to the foreground (shell builtin). Usage fg job_number

kill = send a signal to a process asking it to terminate. Usage: kill process_id

killall = kill all processes running a given command. Usage: killall command_name

echo = just print out its string parameters. Often used with variable expansion to generate output from a shell script. Example: echo "My home is $HOME"
-n = dont automatically print a newline at the end of the output

env = report exported environment variables

grep = read from files listed on the command line (or stdin) and report lines that match a given pattern. Usage: grep pattern file1 file2 … Example: grep "printf" test.c test.c
-v = report only lines that don't match the given pattern
-c = dont report matching lines, just ripen the number of lines that match each input file
-E = interpret pattern as extended regular expression syntax
-i = ignore case
-n = prefix each reported match with the line number
-q = quiet mode. Don't print anything. Just use the exit status to report whether or not a match was found

find = recursively search directories to find matching files. By default, just print out matching pathnames. Usage: find path options
-print = print out each matching pattern
-type f = only report matching files
-type d = only report matching directories
-mmin -n = report file modified less than n minutes ago
-mmin +n = report files modified more than n minutes ago
-mtime -n = report files modified less than n days ago
-mtime +n = report files modified more than n days ago
-exec command = execute command for each matching file

stat = display information about files. Usage: stat options file
-c '%s' = report size in bytes
-c '%U' = report name of the owner
-c '%G' = report group ownership of the file

sort = sort lines of given file (or stdin) and output them in sorted order. Usage: sort file OR sort
- -n = interpret each line as a number and sort by magnitude
- -r = sort in reversed order

wc = report the number of bytes, words, and lines in a file
- -c = just report the number of bytes
- -w = just report the number of words
- -l = just report the number of lines

chmod = change permissions on a file or directory. Usage: chmod options file
- +x = add execute permissions
- -w = remove read permissions
- u+w = add write permissions for the user that owns the file
- g+r = add read permissions for the group that owns the file
- o-x = remove execute permissions for others (other than the files owner or group)
- a+r = add read permissions for everyone

chmod _ _ _ treat each blank as binary number
so _ _ _ _ _ _ _ _ _
first three blanks associated with user
next three blanks associated with group
last three is all other people
so chmod 777 will give permissions to all people because 1 1 1 1 1 1 1 1 1

VIM COMMANDS:

```
          ^
          k
<h                  l>
          j
          V
```

- <ESC> makes sure in normal mode
- q! exits and disregards any changes
- x deletes character under cursor
- press i to insert text before cursor
- A to append data after line
- use :wq to save and exit
- type dw to delete a word
- type d$ to delete the end of a line
- 0 moves to the start of a line
- motions
    - w = until start of next word
    - e = to the end of the current word
    - $ = to the end of the file
    - typing a number with an operator repeats it that many time
    - dd to delete a whole word
    - press u to undo last commands
    - press U to fix a whole line
    - ctrl + R to redo commands

- type p to put previously deleted text after the cursor
- type r_ to replace character at cursor with _ or whatever you want
- to change until the end of a word ce
    - can be used with same motion as delete - c [number] motion
- ctrl+G to show location in the file and the file status
- type G to move to a line in the file ??????? CHECK
- gg to move to start of file
- G to move to bottom of file
- type # of line you were on and G to return to line you were on when you first pressed G
- type / followed by a phrase to search for the phrase
    - type n to go to next occurrence, and N to go up and search for occurrence
    - to search in backwards direction type ? instead of /
    - to go back to where you started from type ctrl-o, ctrl-i goes forward
- type % to find a matching),], } will go find next bracket or parenthesis or curly brace
- :s/wordtofind/wordreplacewith = only does first occurrence unless you type /g at the end
    - :#,#s/old/new/g = where # and # are line numbers of range of lines to be done
    - :%s/old/new/g = change every occurrence in whole file
- :! allows you to write any external command you want to execute inside your VIM
- to save part of a file highlight part with v and type w FILENAME after hitting :
- to insert the contents of a file, type :r FILENAME
- type o to open a line below the cursor and place you in insert mode
    - to open a line above type O
- type a to insert text after the cursor
- type R to replace more than one character
- use the y operator to copy text and p to paste it
    - yw yanks one word
- j jumps to next line
- to ignore case when searching or substituting
    - :set ic
    - to undo this do :set noic
- to highlight all found do :set hls is
- e moves to the end of a word
- LESSON 7 IN VIM NOT COVERED

C vs. C++:

- C has no bool type = instead use macros = text replacement
    - #define bool int
    - #define true 1
    - #define false 0
- C has no reference parameters. We'll eventually use pointers instead. Thats how reference parameters are implemented anyways
- C has no classes. Only has structs. Cannot have member functions inside of structs in C
- C has no virtual functions. We use pointers to functions instead. Store a pointer to a function in a variable or a field and then call the function when we need to. Remember the name of a function evaluates to the address where its stored. The most difficult part is writing out the type of a pointer to a function.
- const behaves differently. Cannot use it to define values that are treated as compile time constants
    - const int SIZE = 200;
    - int list[SIZE];

    - #define SIZE 200
    - int list[SIZE];

- C has no inline function support. You can use #define to do same job
    - inline eliminates need for stack fram
    - max(4, 5)
    - #define max(x,y) x>y? x:y
- #define is essentially text replacement
- C has no user defined operators. We write own functions to do that
- C does not permit function overloading. We just have to make sure that different functions have different names
- C has no general notion of value semantics. Which means we cant expect to assign structs by value or pass and return from functions by value
- C only supports block style comments
- C has no special purpose string type, we have to use null terminated character strings instead
- C doesn't support C++ I/O streams, we will use C standard I/O library instead
- no new operator = use malloc() and free() instead

REGEX VS. FILENAME EXPANSION:

- regular expressions (regex) are used in commands for pattern matching in text
- filename expansion is used by the shell for matching file and directory names

- regex
    - * Zero or more of previous pattern
    - ? Zero or one of preceding pattern
    - . Any character except line break
    - () Capture group, saves & stores into character
    - \ Escapes a character
    - ^ Anchor to beginning of line
    - [] Character class
    - $ End of line
    - {n,m} At least n occurrences, at most m occurrences
    - {n} Exactly n occurrences
    - {n,} At least n occurrences
    - {,m} At most m occurrences
    - + One or more
    - - Ranges

- filename expansion (pattern matching)
    - * Zero or more characters
    - ? One character
    - . A period

TARBALL:

- tar -zcvf archiveName.tar.gz directoryName
    - to tarball a directory
    - -z = compress archive using gzip program
    - -c = create archive
    - -v = verbose i.e. display progress while creating archive
    - -f = archive file name

- -x extracts files
- tar -xvzf archive name.tar.gz
    - extracts tarball

COMMAND SUBSTITUTION:

- allows the output of a command to replace the command itself
    - this is done with either $(command)   OR   `command`
- used when you want the output of a bash command in your script, or to execute a bash command in your script

GLOBBING:

SCRIPTING:

http://ftp.psu.ac.th/pub/bash-howto/reference_bash-cheat.pdf

TWO TYPES OF RETURNS:

- There are two types of returns we use in bash. One is good the other is not as good.
- C-Style return (correct way)

```
sumIt(){
        sum = 0
        for var
        do
                let sum += $var
        done
        echo $sum
}
val = $(sumIt 1 2 3)
```

- Other type of return (less correct way)

RELATIVE AND ABSOLUTE PATHS:

- absolute path = full pathname from the root directory to where you are. Complete path from start of actual filesystem from / directory
- relative path = (pwd) path related to the present working directory
- / = root directory
- ./ = current directory
- ../ = parent directory

EXPANSION SUPPRESSION:

ARITHMETIC IN BASH:

SOFT AND HARD LINKS:

- two types of links: (ln -link)
    - symbolic (soft link) (ln -s)
        - ln -s foo foo2 (creates a soft link)
        - think of it as a pointer
        - knows only how to get to a name it points to
        - if you change the file linked to, changes all files
        - can move linked to other directories and will still be linked
    - hard link
        - think of it as a reference parameter

PIPING:

INPUT REDIRECTION:

pgm > file Output of pgm is redirected to file.
pgm < file Program pgm reads its input from file.
pgm >> file Output of pgm is appended to file.
pgm1 | pgm2 Output of pgm1 is piped into pgm2 as the input to pgm2.
n > file Output from stream with descriptor n redirected to file.
n >> file Output from stream with descriptor n appended to file.
n >& m Merge output from stream n with stream m.
n <& m Merge input from stream n with stream m.
<< tag Standard input comes from here through next tag at start of line.

SYSTEM CALLS VS. STDIO: (advantages and disadvantages)

FORKING:

- fork() = create a child process
    - exact copy of currently running process
    - return from function fork is diff (how you tell parent + child apart)
    - < 0 error
    - 0 child
    - > 0 parent
- wait(int *status)
    - a child process that is done executing bus has not been waited on is called defunct or zombie process (memory leak is about the same as zombies)
    - wait prevent zombies!
    -

PIPING FOR COMMUNICATION:

- There are two types of pipes (unnamed and named pipes)
  - Only concerned with unnamed pipes
  - pipe() = create pipe - a pipe is a FIFO structure (queue)
  - anything communication wise is done through a byte stream
  - pthread_create = creates a thread
  - pthread_join = equivalent to wait for processes
  - shared memory up front for free
    - what things are shared?
      - Not the stack
      - dynamic memory
      - global variable space

SIGNALS:

EXECUTE FAMILY:

MAKE:

DEBUGGERS:

THREADS:

- lightweight compared to processes
- do NOT communicate through pipes
- shared memory system
- pthread_create

SEMAPHORES:

EXTRA STUFF:

- PATH = list of directories to search for an executable
- root directory in linux is /
- . = current directory
- .. = previous directory
- using ./ in executables is typically a security thing
- dynamic or shared library extensions in bash are .so and .dll
- static libraries in bash are .a and .lib
- PID = process id
- PPID = parent process ID
- a struct in C with no name is an anonymous structure
- ERRNO = global variable functions will set a value to if encounter error
- 2 > &1 redirects std error to std output
- bit fields (more info)
- getopt