WEEK 1

- Why is the system broken up into these different parts? (Kernel, Shell, Programs)
    - Fair allocation of resources with multiple users
    - Prevents user access to kernel
    - Allows kernel to handle some of the nitty gritty work
- How do I find where files are on the system?
    - Use find, whereis
- How do I find out what options are available for a particular utility?
    - man pages
- When is a file a file and when is it a process?
    - Process is an executing program with a PID, file is a collection of data. Process gets instructions to execute from a file
- What types of links are there?
    - Soft links (file/dir), hard links (physical data), sausage links.

WEEK 2

- What else does locale affect other than user's cultural preferences (language, country, area-specific things)
  - Rendering text, alphabet, time and date formats, sort order
- Why do we have environment variables, what functions do they serve?
  - They define the environment. They specify how a user wants a process to be run. Processes created by the original process can access the same environment variables.
- Why do we want to redirect I/O? What are some examples of use cases for I/O redirection? How do we implement this in C?
  - So we can use the output of one command as the input of another command. You can use >, >>, < and |.
  - In C, you use fopen and fclose
- Why do regex exist? Are the expressions the same across languages?
  - Regex exists so that we can find things using patterns
  - The expressions are not the same across languages, for instance Perl and Java define their own expressions
- What's the difference between a basic and extended regex? When do I use either?
  - BRE does not recognize meta-characters
- What are the differences between tr, sed, and grep? When do I use which?
  - Tr translates/deletes character by character
  - Sed translates/deletes string-by-string
  - Grep is searches for matches for a string that you give it, but it doesn't translate/delete anything
  - You would use tr when you want to look at characters, sed when you want to analyze strings, and grep when you want to find a match to a string
- How do you implement comm and tr?
  - Aka know the assignments where we implemented comm.py and tr2u.c and tr2b.c
  - Comm [options] file1 file2
    - Most common options: -123
      - Choose which column to suppress
    - Only works on sorted files
  - Tr [options] str1 str2
    - Ex: tr ‘abc’ ‘xyz’
      - This will change all occurrences of a to x, b to y, c to z

WEEK 3

- Why do we have the compilation process?
  - Don’t want to do entire process again if we make one change, just relink new file
- How does gcc work?
  - gcc is a compiler, a set of packages with a compiler, assembler, linker, loader, etc.
  - Does process in specific order unless told otherwise
- What are the different components of the process?
  - Source code (.cpp) --preprocessor→ --compiler→ assembly file (.s) --assembler→ object code file (.o) --linker→ executable file
- Why can’t I execute individual object code files?
  - Might want other libraries or have dependencies so it can’t be run alone
- What are the differences between open source and closed source software? When do I use one or the other?
  - Open source is publically available and modifiable, mass collaboration
  - Closed source is closely guarded, comes with usage/modification restrictions
  - Open source is free and flexible, constantly updated while closed source all bugs/issues taken care of by the seller. Better support/documentation
- Why do we have make?
  - Don’t want to write gcc command each time, life is easier
  - Don’t want to recompile entire program for small change, just want needs to be recompiled
- Why not just fix the original source code? Why use patches?
  - Lots of users, users can just patch rather than have to fix it themselves

WEEK 4

- Why bother with source control?
    - Keep track of changes to code
    - Can create versions and back track
- What are the strengths and weaknesses of source control?
    - Can make new versions of code without affecting the main branch
    - Can work on many bugs simultaneously
    - But you have to communicate changes to everyone if you’re using distributed vcs
    - If you’re using centralized vcs, the version history sits on a central server, so if you screw up, you screw everyone over
- When do I use it and how?
    - If you want to take a look at a particular part of code. Use git checkout.
    - If you want to create a new version. Use git branch
    - Collaborating with other people
    - If you want to make backups/be able to revert changes. Use git branch to make new versions, git revert/reset to go backwards
- What is the difference between a working copy and the repository?
    - A working copy is a copy of the repository that you are modifying or looking at.
- What is a commit? What should be in a commit? How many files should commits contain?
    - A commit is a “snapshot” of the code/project that you are working on.
    - A commit should consist of the modifications that you made to a piece of code
    - There should be as many files as you modified/added to a piece of code in a commit ???
- What are the differences between head and HEAD?
    - A head refers to a commit object
    - HEAD is the currently active head (that’s pointing to a commit obj)
    - There can be multiple heads, but there’s only one HEAD
- What happens with a merge vs. rebase?
    - Merge creates a new branch that ties the histories of both branches together There will be a new “tip” that includes the ancestors of both branches that you merged.
    - Rebase moves the entire branch to the “tip” of the master branch. It rewrites the project history by creating new commits for each commit in the org branch
- Why bother having branches at all? Why can’t we just all work on the same single master branch?
    - If people all work on a single master branch and someone messes up, it will be hard to fix and backtrack if you don’t have backups
    - You can make different versions
    - You can easily visualize a project if there are branches that represent things like bugs
    - You can work on bugs independently, or simultaneously
- What happens when we perform a merge? How does it work?
    - See above

WEEK 5

- Why do I have a debugger? Advantages/Disadvantages?
    - Because your code sux and you need to fix it b/c it more broke than u
    - Can debug/step through source code line by line, interact/inspect program
    - Advantages: line by line, breakpoints, watch vars
    - Disadvantages: not real-time, uninterrupted so some errors may not be found
- When do I use a debugger?
    - When program has error or unexpected output
- How do we find bugs? What kinds are there?
    - Logic errors & runtime errors
    - Runtime will cause error/crash/failure like your life
    - Logic: Runs and exits successfully but it's still wrong :(
- How do we know where to set breakpoints?
    - Look at functions, look at error, be logical and place them where you expect unexpected behavior to come from
    - Can also set a breakpoint at a location IF expression = true
- What's the difference between s and n?
    - S steps IN to the next line (so in to a function call), n just goes to the next instruction in the current frame
- When do we use c/s/n/f?
    - After a breakpoint to resume execution and continue monitoring values
    - C: to continue running, want to go to next breakpoint or run to end
    - S: Look into next source code line
    - N: Keep going in current stack frame, similar to S
    - F: Finish execution of current function, once hits a return pauses again
        - displays function's return value and line in next statement
- What's the point of displaying data?
    - See what the values are at this point in time, see if there's anything unexpected
- What sort of data might we want to display?
    - Decimal, hexadecimal, octal, binary or other data types
- Why does the stack exist? How is it different than the heap?
    - Stack is LIFO, for function call stack frames. Things in particular order, so less flexible than the heap.
    - Heap is much more flexible, and will not delete memory unless done so explicitly. With heap can allocate/deallocate in any order
- Why is memory allocated at runtime in dynamic memory?
    - Exact amount of space/number of items doesn't have to be known by compiler in advance
    - Don't need to know info about inputs beforehand, and can allocate more as you go
- What happens if I never call free?
    - Usually OS reclaims it anyways, but you're wasting resources so that's bad
- What happens if I try to put data into dynamic memory but I haven't called malloc yet?
    - Error since you're trying to put something in a space that doesn't exist yet

WEEK 6

- Why have different processor modes? How do we switch modes?
    - So the user does not have access to important things like memory
    - So user cannot perform illegal i/o operations
    - So user cannot use CPU for too long
    - You can switch modes using system calls
- What happens if user code is given unrestricted access to CPU?
    - It can do illegal i/o operations and hog up memory and CPU time
    - Unfair for other processes running
- Why do we have dual-mode operation?
    - I/O protection
    - Memory protection
    - Cpu protection
- When do we need to use system calls?
    - When you want to perform something as the kernel
- How do these library functions perform privileged operations? (getchar/putchar/fopen/fclose)
    - Buffered, so read in as many bytes as possible
    - It lets OS know it wants to perform a sys call
    - Process is interrupted and computer saves the current state
    - OS take control of CPU and verifies validity of the user(?)
    - OS performs the action
    - OS restores the current state
    - Go back to user mode and give CPU back
    - Continue running
- What are the benefits and tradeoffs of using either system calls or C library functions?
    - System calls
        - Let's you perform privileged operations safely
        - Can be sure that you are not harming the system in any way
        - But it takes some time if u use system calls a lot because of the overhead
    - Library functions
        - Also lets you use sys calls to perform privileged operations safely
        - Faster because they are buffered
        - Less overhead
        - But have to load standard libraries
- Which is faster in what applications? When would you use buffered or unbuffered I/O?
    - Buffered is faster, for instance in using fopen and fclose vs open and close
    - You should use unbuffered if there might be a power outage, because everything stored in the buffer will be lost
    - Use buffered if you want it to take less time or if just reading one element at a time

WEEK 7

- What’s the point of parallelism?
  - Improve performance by maximizing use of CPU
- How can you decide whether your application should use multiple processes or multiple threads? Or both?
  - More complex processes will slow down with multiple threads as there will be less resources available for each thread, but processes require a lot
  - Use multiple threads for simple processes, less overhead to establish, easy data sharing
    - Gross with synchronization overhead, and be wary of data corruption/race conditions, debugging
  - Use multiple processes when tasks take significant computing power or memory, this way have multiple single-threaded processes that are isolated from each other so CPU can swap between them as is best
- What is a thread?
  - A flow of instructions/path of execution within a process
- Why share the same code? Why share the same heap?
  - Break one process into multiple parts with shared memory
  - Powerful, easy data access/sharing, efficiency in creation/destruction
- What happens when we have race conditions?
  - Several threads access/modify the same shared data at the same time
  - Result depends on order of execution
- How do we keep track of threads within a program’s execution? How many can we have?
  - Each thread is passed a unique identifier. We can have as many threads as desired, but to a certain point the program will no longer be optimized through parallelization with too many threads
  - Limited CPU, memory, I/O capabilities… think of it as a bottleneck
- How do we pass data to threads we create? How do we tell them what to work on?
  - We pass them as parameters, with a function pointer and an argument pointer
  - To pass more than a single argument, pass in a struct
- What happens if our application isn’t “embarrassingly parallel”?
  - Then there is probably communication required between tasks and dependency on intermediate values of other threads to complete one thread
  - Much harder to parallelize, requires “waiting” which can increase runtime
- Why join at all? What does a join guarantee?
  - Join so originating thread waits for completion of spawn/child threads
  - Prevents abortion of child thread before job completion

WEEK 8

- When would I use static linking? Why would I use it?
    - Portability reasons
    - It is faster than dynamic linking because dynamic linking has to load things at run time.
    - faster and more portable, since it does not require the presence of the library on the system where it is run.
- When would I use dynamic linking? Why would I use it?
    - You should use dynamic linking if you want to have a smaller file size and for easy updating.
    - Dynamic linking will allow you to load programs as needed at runtime and for you to share libraries.
    - When the library is changed, the code that it references does not need to be recompiled.
- Why isn’t everything written as one big program, removing the need to link?
    - More efficient
    - > 100k line program, easier to just recompile the one function and relink rather than recompiling everything
    - Good with multi-language programs
- When does linking happen? When does loading happen?
    - Loading happens after u have all the object code. It happens before runtime.
    - Linking typically happens during runtime and after you load
- How are libraries dynamically loaded?
    - Through the linker
    - The Linker collects procedures and links the object modules into one executable program
- If you are a sysadmin, do you prefer dynamic or static linking?
    - Probably dynamic bc of its smaller size
- What is the differance between linking and loading?
    - Loading loads all of the libraries into the main memory
    - It loads executables into memory prior to execution so that they can be run
    - Linking takes smaller procedures and links together the object modules into a larger executable. It’s easier to update.
- What are advantages of dynamic linking?
    - Small size, don't need to recompile functions that depend on libraries that have changes made to them
    - Multiple programs can access same libraries at same time
- What if the necessary dynamic library is missing?
    - Program crashes
- What if we have the library, but it is the wrong version?
    - Nothing, you just might not get the result you expect

WEEK 9

- Why do we want these guarantees (confidentiality, data integrity, authentication, authorization)?
  - Message secrecy + consistency, identity confirmation, specifying access rights to resources
  - Prevent interception of communication or attacks, allow trust of incoming data
  - Protect sensitive information
- Why have encryption?
  - To maintain the guarantees, in case of interception still indecipherable
- What are the differences between symmetric and asymmetric encryption? When would I use one or the other?
  - Symmetric: same key used for both encryption and decryption
  - Asymmetric: public and private key pairs, no one gets your private key. Public to encrypt, private to decrypt
  - Symmetric is more efficient, can handle large data sets, shorter keys, and can be combined to get stronger cryptography, but has risky key delivery
  - Asymmetric has efficient key management, good for digital signatures and key exchange, and can be used for years without risk
- What is used on the Internet?
  - Most info on Internet uses symmetric because it is fast, easy, low CPU. Even though asymmetric is harder to decrypt, also harder to encrypt. Asymmetric is generally for secure online exchanges via SSL (Secure Sockets Layer) like monetary/sensitive data transactions
- What is a certificate authority?
  - Issues digital certificates that verifies ownership of a public key by subject named on certificate
  - 3rd Party verifies that sender is legit
- How can I trust a message came from someone?
  - Use a digital signature, an electronic stamp appended/detached to a document that ensures data integrity
- How are signatures different than encryption?
  - Signatures show ownership of the private key, since public keys can be intercepted anyone can encrypt, but if we only want messages from people we trust, something signed with their private key could only come from them.
- Does a digital signature prove origin?
  - No it does not, it only shows that the public key and signature go along with each other and whether or not the data has been tampered with. If a message had been modified and signed with their own key, then technically signature is valid but we don't know it's origin
- Why use a detached signature?
  - Often to validate software distributed in compressed tar files
  - Cannot sign such a file internally without altering its contents, so signature is detached in separate file
- Why use a signature?
  - Data authenticity, sender/receiver verification/authorization
  - Copyright, intellectual property protection
- Who can create a signature? How do I verify a signature?
  - A 3rd party can if trusted and supply a certificate that details origin/validity
  - Anyone with public/private key can create a signature
  - Can verify a signature by taking message digest and decrypting it with public key. It should match the decrypted message -> hash -> message digest generated.