- When is final exam?
    - Friday, March 20th. 3:00-6:00PM
- Where?
    - In room: 3760 Boelter
- Short essay questions
- Writing code
- Recognizing errors in existing code
- Review all labs and assignments
    - material in the assignments as well as material covered in class can be on the exam
- Open notes, open-book (no electronic devices)

Week 1:
- basic linux commands
- man pages
- Unix permissions (chmod command), how to change permissions on a file
- Piping and redirection

Week 2:
- Regular expressions and wildcards
- Grep, sed (substitution command), tr, find
- Bash scripting (a lot of material here: different constructs: for loops, while loops, etc.)
- Single vs. double quotes vs. back ticks, etc.

Week 3:
- C compilation process (compiler, assembler, linker, etc.)
- Makefiles and the build process
- Python scripting
- Patch, diff

Applying patches: unified diff format, pnum stripping
diff –u file1 file2 > patch_file
--- /usr/local/bin/file1 (path to original)
+++ /usr/local/bin/file2 (path to modified)
@@ -l,s +l,s @@ hunk, l,s lines the change applies to in from/to files
You're in local, which num to use? => patch –pnum < patch_file

Week 4:
- Git version control

Centralized vs. distributed
4 object types in git architecture: blob, tree, commit, tag
2 ways to create a version controlled project (git init, git clone)
Different commands we used in the lab: commit, checkout, branch
Why do you have to add then commit?
What's a branch? A pointer to a commit object.
Default branch? Master. Keeps moving each time a commit is added
Create new branch with branch new_branch
Where is new branch created? Current commit where HEAD points

Checkout new_branch to make HEAD point to it and make it move instead of master

Week 5:

- C programming: pointers, function pointers, file I/O, memory allocation, etc.
- GDB and debugging commands

Debugging gdb, c pointers, function pointers, etc.
Used the qsort function which takes a function pointer as arg
Gdb: gdb <executable>, (gdb) run <args>, c/n/s, break file:line, bt (stack frames)

Week 6:

- SSH, key cryptography, private/public keys, digital signatures

How does SSH protocol work? Where is the symmetric/asymmetric encryption happening?
How does SSH guarantee confidentiality and authentication?
Public private keys, who uses which key
Alice sends message to Bob (which key does she use to encrypt and which does Bob use to decrypt)?
Digital signature: opposite, encrypt the digest with private key, decrypt with public.
"Why the difference?" different guarantees

Week 7:

- System calls (read, write, etc.)
- Library calls vs system calls

Library calls vs system calls
System calls are a way for user programs to perform privileged instructions like I/O.
OS executes commands on behalf of the user program.
Overhead: user process is interrupted, computer saves state, OS takes control, OS performs action, saves state, control back to user process
Why do we do this? Protection, user programs are untrusted, if they could write to any part of memory they would change each other's memory (processes) or change OS data structures (give themselves permission) => system calls tell the OS to do it, OS checks if we can and either does it or yells at us

Week 8:

- Buffer overruns (process memory layout, stack, etc.)

Put more data in a buffer than it can handle, we overwrite other parts of the process memory, return value etc. -> can execute malicious code

Put code we're trying to execute in buffer we're overflowing
Overwrite the return address to point back into buffer

How are stack frames built? How is the stack used?

Week 9:

- Multithreading
- Difference between a thread and a process

- Pros and cons of each

Difference between a thread and a process (Pros and cons of each)
-threads share memory except stack (easy communication), light-weight (no extra memory allocation by OS, inside process), an error in one thread brings down all others

Process: expensive creation destruction, communication through system calls
No shared memory/insulation: one buggy process can't bring others down
- race conditions and thread synchronization

More examples with multithreaded code (pthread_create, pthread_join)
Need to be able to parallelize a program

Not on final: EMACS