Reading Guide for Quiz 1.

The quiz has 15 multiple choice questions. The individual part of the test is worth 75xp and the group part is also worth 75xp. So the possible total points you might get from this test is 150xp.

Letter-sized paper of self-produced notes

For the quiz you can use an 8.5x11 sheet of self-produced notes. By self-produced I mean that you-yourself produced them and they weren't mass-produced/printed by someone else. You can type and print them out as long as they are legible to you without magnification :-)

This test is on a general overview of OS and about processes.

It covers anything covered in lecture as well as OSTEP chapters 2,4,5,6, and 15.

General Intro to OS.

1. You should know what happens when you press the `on` button on your laptop (the bootloader lab material). You don't need to know the finer details. As long as you could give a 2 minute overview you should be fine. For example:
    A. what is the BIOS?
    B. what is a bootloader?

2. You should be able to read and understand small code snippets that include pointers and allocating memory on the heap.
3. Between the window manager and the memory manager, which is part of the kernel?

Processes

1. You should know the material in OSTEP 5 - Process API
2. Anything from the reading that deals with privileged instructions and dual mode operation you can consider important. (OSTEP 4, 5,6) You should be able to define these terms. (google)
   For example, of an instruction that writes a disk block, an instruction that sends output to the printer, an instruction that halts the CPU, an instruction that reads from the keyboard and an instruction that changes the value of the program counter, which is not a privileged instruction
3. You should know the Unix definition of a process ("Operating system abstraction to represent what is needed to run a single program") and what in memory and hardware represents a process. This is covered in lecture, OSTEP 4 and this short TA writeup. There is something in the TA writeup that has a high likelihood of being on the test.

4. You should know what happens when there is an interrupt, processor exception, or system call trap. This is covered in lecture and in Limited Direct Execution Protocol in OSTEP 6
5. You should know what stack buffer overflow (stack smashing), segmentation faults, and Address Space Layout Randomization are. You should have a rough idea of how to write a program that smashes the stack.

6. You should know about virtual addresses and process (OSTEP 15). This is also covered a bit in the process calisthenic. Remember, the virtual address space of a process always starts at zero.

7. You should read and understand the process calisthenic, the process creation calisthenic and the signal calisthenic
    a. What is a process
    b. What is fork?
       For example, what is shared between a parent process and a child process?
       What system call do we use to load and run a new program in the context of the current process?
    c. You should know how the shell process handles redirection. For example,
       `./transpose < inputFile > outputFile`
    d. Should should know about orphans and zombies.
    e. What happens when a fault is detected -- how is it handled and by what component (hardware, kernel, user code)?
       What is an interrupt vector table?

For these calisthenics, if I give you a code snippet that closely mirrors one of those in the calisthenics you should be able to know what it is illustrating and what it does. For example:

Example 1:

```
int main() {
   int pid = fork();
   if (pid == 0) {
     sleep(10);
     return 42;
   } else {
     sleep(50);
   }
   return 0;
 }
```

Example 2

```
char *stuff = malloc(20);
printf("X: %p \n", &stuff);
printf("Y: %p \n", stuff);
```

Example 3

```
void handler (int sig){
      printf("Are you sure?\n");
      count++;
}

int main(){
      struct sigaction sa;
      int pid = getpid();
      printf("Running %d \n", pid);
      sa.sa_handler = handler;
      sa.sa_flags = 0;
      sigemptyset(&sa.sa_mask);
      if (sigaction(SIGINT, &sa, NULL) != 0){
            return(1);
      }

      while (count != 2){
            printf("h");
      }
      return 0;
}
```