CS 361: Computer Systems Fall 2019	**Lab Session 4 Worksheet**

Introduction to Homework 3

In this lab, we will be writing a shell program. A shell is the program that all of you use when you ssh to systems, or bertvm, or any other server. It's the program which prints out a prompt, reads in what you type, and then executes whatever instruction you told it to do. It is different from ssh (which is the program you use to connect to the server), or linux (which is the OS on which your shell is running). Read the instructions on gradescope to learn more about what our specific shell will do. However, the basic loop for our (and any) shell, will be:

1. Prompt the user
2. Read and parse the command the user types in
3. Create a new process to run the command the user typed
4. Wait for the new process to end and print out its exit status
5. Start over with the prompt

Reading and Parsing Strings

1. Before we begin, log in to systems{1-4}.cs.uic.edu, and check out the github repository which contains the practice code for this lab session (https://github.com/uicsystems/LabSession4-PracticeCode.git).
2. Compile and run stringparsing.c. Note that it takes in a line from the terminal, tokenizes it into words, and copies each word into a separate entry in char** array.
3. Carefully read through stringparsing.c so you are sure you understand what it's doing. Next, answer the questions 1 & 2 on gradescope.

Fork and wait

1. Compile and run forkexample.c. Read the code, and make sure you understand what it's doing.
2. Look at the man pages for fork, exit and waitpid. (To view a man page, type "man fork" or "man waitpid" into terminal.)
3. Answer the questions 3 & 4 on gradescope.

Exec

1. Compile and run execexample.c. Read the code, and make sure you understand what it's doing.
2. Look at the man page for exec.
3. Answer the questions 5 & 6 on gradescope.

Getting Started Writing Your Shell

1. Homework 3 does not have any skeleton code. Begin by checking out your Homework 3 github repository via the link on the course website. (You should also probably read the assignment description while you're there.)
2. Create a file named "hw3.c" and add it to your github repository - this is where your code for the lab should live. You should also create and add a "netid.md" file that contains your netid.
3. Start your shell by writing code to print "361 ⟩" and read in whatever the user types in an infinite loop. It's important that your prompt be exactly "361 ⟩" so we can grade your assignment. The fgets command may be helpful to read in strings from standard in.
4. Add a check to exit your program if the user's input was "exit."
5. Next, parse the user's input into an array, where array[0] is the command they typed, and every other entry in the array is an argument to that command. You're essentially breaking the string up into words, and making each word its own entry in an array of string pointers. So if the input was "/bin/ps -u cynthiat" you would create the following array:

   ```
   array[0] = ‘‘/bin/ps’’
   array[1] = ‘‘-u’’
   array[2] = ‘‘cynthiat’’
   ```

   You can assume that the command will be less than 500 characters, it will have less than 20 arguments, and that each argument will be less than 100 characters.
6. You will want to use fork, exec and wait to have your shell run the commands the user enters. First, try forking off a child process and collecting the pid of that child. (You will need to print it later.) Look at the man page for fork if you want more information on its return value or anything else.
7. Next, try using exec to have your forked child do useful work. Look at the man page for exec to get a list of the different exec functions, and choose the one you want to use. Note that we will expect your commands to use a full path - so to run "ps -e" in your shell, you would type "/bin/ps -e"

8. Now we want your shell to politely wait until the command has finished running. Use the wait system call to pause your shell program, and to collect the exit value of the child process you just created. After it exits, print ``pid:n status:m'' where `n` is the PID and `m` is the child's exit status. The "man waitpid" command will give you the man page for wait.