CS3844 Lecture 6 Notes:

Objectives: -

Construct programs using the fork, wait, and exec unix commands

In class exercise:

Explain to students the concept of form and wait.

Show the man pages for these commands to students.

Construct a basic shell that uses these commands to allow the user to execute a new command.

Once this is done, add a new command (such as dir) to the shell.

From this, add the ability to verify that the user wants to do something (such as a y before removing files.

Then add something like a previous command to the shell.

Sample source code might be the following:

```
#include <sys/types.h>

#include <stdint.h>

#include <stdio.h>

#include <unistd.h>

#include <string.h>


int main (int argc, char*argv[])

{

  pid_t pid; /* Declare a pid type. */

  pid_t mypid; /* Declare a second pid type */

  char cmd[255];

  char prev[255];

  char sb[2];

  int keepGoing = 1;
```

```
 mypid = getpid(); /* Get my pid from the OS. */

 printf("The shell, wws shell, is executing.  My PID is: %d. (%x)\n", mypid, mypid);   /* Print out my process ID data.
*/


 char *args[] = {"cat", "hello.txt", ""};


 execvp("cat", args);



 while ((keepGoing != 0))

 {

  printf("\nWWS->");


            memset (&cmd[0], 0, sizeof(cmd) );

            fgets(cmd, sizeof(cmd), stdin);


            if (strncmp(&cmd[0], "p", 1)==0)

            {

              strcpy(&cmd[0], &prev[0]);

            }

            else if (strncmp(&cmd[0], "rm", 2)==0)

            {

              printf("Are you sure you would like to delete these files with this commend: %s (Y/N)", cmd);

    fgets(sb, 2, stdin);

              if (strncmp(&sb[0], "Y", 1)==0)

    {  // Do the command.

    }
```

```
        else
        {
          strcpy(&cmd[0], "echo I am not removing those files.");
        }
      }



      // Determine what to do.  If the command is exit, exit the shell.
      if ((strncmp(&cmd[0], "exit", 4)==0)||(strncmp(&cmd[0], "quit", 4)==0))
      {
        // Exit the shell.
        keepGoing = 0;
        printf("Exiting");
      }
      else
      {
        strcpy(&prev[0], &cmd[0]);
        // Fork a new process.
        pid = fork();   /* Fork a child process of this process. */

        if (pid < 0)
        { /* An error occurred. */ }
else if (pid > 0)
                { /* I am the parent process and my child is the following process. */
  printf("Start %s as pid is %x. \n", cmd, pid);
```

```
                // Determine if the command should execute in the background.
                if (cmd[0]=='&')
                {
                        // Execute the program in the background.
                }
                else
                {
                        // Wait for the command to finish.
                        wait(pid);
                }
        }
    else
        { /* I am the child process. */
                if (cmd[0]=='&')
                {
                  execv(&cmd[1], NULL);
                  //system(&cmd[1]);
                }
                else
                {
                  execv(&cmd[0], NULL);
                  //system(&cmd[0]);
                }
                exit(0);
         }
        }
}
```

```
  exit(0);  /* Exit the program, returning 0 as an error code.*/

}
```