EE 4374 Operating Systems Design
Programming Assignment #4: Unix Shell
Due Date: April 25, 2022 (before Midnight)

Objectives:

1) To learn how to use POSIX system calls in C.
2) To practice creating processes using the *fork()* system call.
3) To practice updating the memory image of a process with the *execvp()* system call.
4) To practice blocking a parent process until a child process terminates using the *wait() or waitpid()* system calls.
5) To learn how to overwrite a file descriptor with another file descriptor to redirect standard output to a file with the *dup2()* system call.

Unix Shell:

You will develop a shell. A shell is a program that interprets commands and acts as an intermediary between the user and the operating system. There are several Unix shells available for you to use, some of them are:

1. Korn shell (ksh)
2. TC shell (tcs)
3. Bourne-again shell (bash)

You will make use of the following system calls to develop a shell:

a. **fork**(): int fork();
b. **execvp**(): int execvp(const char *file, char *const argv[]);
c. **wait**() or waitpid(): int wait(status);
d. **close**(): int close(file_descriptor);
e. **open**():int open(const char *path, int flags); and
   int open(const char *path, int flags, mode_t modes);
f. **dup2**(): int dup2(int filedes, int filedes2);
g. **read**(): int read(file descriptor, buffer, n_to_read);
h. **write**(): int write(file descriptor, buffer, n_to_write);

To use these system calls, your program might include the following libraries:

- #include <fcntl.h>
- #include <sys/types.h>
- #include <unistd.h>

Tasks:

1) Uncompress the Assignment 4 template provided by the instructor into your home directory. This will create a directory called ‘student_prog4’, please rename this directory to firstinitiallastname_prog4 using the ‘mv’ command. For example, the instructor would rename the directory by executing ‘mv student_prog4 mmcgarry_prog4’. Go into the directory and rename all of the files from ‘student_*’ to ‘firstinitiallastname_*’ much like you renamed the directory.
2) Write a main() function that prints a prompt, accepts input, tokenizes the input using the argument tokenizer in firstinitiallastname_argtok.c, and passes the

returned argument vector to an executeCmd() function with the following prototype:

int executeCmd(char **args);

This function returns -1 on error, otherwise 0.

Your shell should exit when an 'x' is entered by itself at the prompt.

Your main() function will be in the file named *firstinitiallastname_prog4.c* and the executeCmd() function and any supporting functions will be in a file named *firstinitiallastname_exec.c* and the function prototype for executeCmd() should be in a file named *firstinitiallastname_exec.h.*

3) Write an executeCmd() function that can execute any program in the foreground or background as well as redirect the output of any program to a file. You can write any number of functions to support executeCmd().

More specifically, executeCmd() should be able to:

3.1) Execute any program with any number of arguments, below are some examples
- *ls*: lists your files in your current directory (format: $ ls)
- *ls -l*: lists your files in long format (format: $ ls -l)
- *ls -a*: lists all files (format: $ ls -a)
- *pwd*: displays current working directory (format: $ pwd)
- *wc*: displays the number of lines, words, and characters in a file (format: $wc filename)
- *mkdir*: make a new directory (format: $mkdir dirName)
- *cat*: creates, concatenates and displays text files

3.2) Same as 3.1 but the program should be run in the background, i.e., the shell will not wait for completion before displaying the prompt again.
- *&*: runs job in background (format: $ cat prog1.c &)
- Note: The instructor has provided a function in student_exec.c called execBackground() that can be used to determine when a job should run in the background. Additionally, that function will strip the '&' character from the argument vector.

3.3) Allow the output of a running program to be redirected to a file
- >: redirects standard output to the filename after >

(format: $ grep if prog1.c > foo)

4) Use a Makefile to build your program. The Makefile provided in the Assignment 4 template only has a 'clean' target. You must add the other targets.

5) Submit the deliverables, indicated below, as a single zip file named firstinitiallastname_prog4.zip through Blackboard.

Deliverables:

1) Submit all of the source files in your Assignment 4 directory as a single zip file. This file will be submitted through Blackboard. Submissions that do not follow these specifications will be ignored!

Scoring:

Operation/Successful Demonstration	**75%**
Does the program compile without errors or warnings? 30%	
Does the program support executing any program? 25%	
Does the program support executing any program in the background? 10%	
Does the program support redirecting the output of any program to a file? 10% **[Extra credit]**	
Adherence to Interface Specification	**20%**
Does the program use specified libraries/APIs? 10%	
Does the program use specified function prototypes? 10%	
Comments/Adherence to Submission Specification	**15%**
Does the submission adhere to the filename guidelines? 10%	
Every function is documented with a header? Source code well documented? 5%	
Lateness	**10%** per day (including weekends and holidays)