KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS

Information and Computer Science Department

ICS431: Operating System

Project One (Term 071)

Due Date: November 17th 2007

Project Submission:

You are requested to submit both hardcopy and softcopy. Hardcopy should contain cover page, your source code, and sample outputs. Softcopy (source code only) will be submitted through WebCT.

Project Rules:

1. Submitting your project after the deadline will result in ZERO mark.
2. Any cheating in the project will result to ZERO mark.
3. THERE WILL BE NO DIFFERENTIATION BETWEEN THE PERSON WHO DOES IT AND THE PERSON WHO LETS OTHERS TO DO IT. Both will have the same penalty.

Project Question:

You are going to implement a simple command line shell. The shell application takes no arguments itself. It reads "commands" from the standard input.

Part 1: The shell must read commands, parse input and fork off programs using **execvp/execlp.** All programs should be started in the background and your shell should be available for more commands.

Part 2: You must add "*job status*" to the shell. This means you have to recognize the command "**ps**". This command should print all tasks that are currently running in the background. You must also recognize the "**fg**" command that will bring a background task to foreground.

Command:

A "command" is a line composed of words separated by any amount of "whitespace" (spaces, tabs) and ended by a single newline. The following logic is applied for each input line:

1. If the input line is empty (after removing whitespace), the next input line is fetched
2. An input line input no more than 8096 bytes is read (the **fgets** library routine can be used)
3. The line is broken into "tokens". Tokens are separated by whitespace. The library routines **strspn, strtok** and **strcspn** are useful for this. You must accommodate arbitrary whitespace (*e.g.* "ls ___-lt___/tmp" where "_" is a space). White space is spaces or tabs.
4. The command is always executed in the background.
5. After the command is started, and if any previous background child process has finished execution, a message of the format "**Child process XXXXX exits**" is printed, where "**XXXXX**" is replaced by the process id.
6. Your Shell should be available immediately to accept more commands.

The program to execute in each command is specified by the first token of input.

Part 1:

1. Fork a child process
2. Use the **execvp/execvl** system call to cause the indicated program specified in the command to be executed. If the **execvp/execvl** returns an error, you must report that the command can not be found and the child process should **exit()**.
3. All commands should be started in the background. One way for achieving this is to start a child process, that child process then starts another child process with the command while the child itself waits for the command. The parent should continue execution of the original shell application. In the child process you can use the **wait**() system call to wait for the child to exit. When the child exits, you should signal the parent that the child has exited and the parent should update the data structure that holds information about currently running child processes.

Part 2:

In this Part, you must implement two "pseudo commands".

1. The first pesudo-command is "**myps**"; this lists the set of tasks that have not finished. Keeping and displaying this information simply involves a data structure of your choice that keeps information about all tasks that were started. As background tasks finish execution, their entries should be removed from the array. You can assume that we will never have more than 100 background tasks. When this command is entered, you program should print the following information about each process

   Process Name:
   Process ID:

2. The second pseudo command is " **fg**", which takes an integer as a second argument. The integer specifies a background process id to be brought to foreground. This command should report an error if the specified process id is ***is not*** that of a background process. If it is a valid process id, the shell should wait for the child that started that process to finish execution before allowing the user to enter more commands.