CS 532 HW 2
DUE Thursday Jan 23 before 6pm

Part I – Programming Exercise

Introduction

This exercise is designed to illustrate the process interface in UNIX systems. You will implement a Unix shell program. You will write a C program that will act as a simple shell command line interpreter for the Linux kernel. Your shell program should use the same style as the Bourne shell for running programs. [Note: you can run the Bourne shell by typing sh.] The Bourne Shell was originally designed by Ritchie and Thompson. *There are 4 parts to this exercise, A – D.*

(A) Command Line Reader and Makefile

WRITE a C program in a file named reader.c with this functionality:

1. Print the prompt: "MY SHELL>"
2. Read one line of user input (a single command line) using `fgets()`
3. Print the command line (the user input) to `stdout`
4. Repeat (1)-(3) until an EOF condition (ex: the user types Ctrl-D).

HINT: look at `feof` to see how to identify end of file; this works well with `fgets()` for reading the input from the stream `stdin`.

WRITE a make file named makefile1 to build your command line reader with gcc and with debugging enabled. The make target should be cmd_reader, and the name of the executable should be cmd_reader.

TEST your program with several different commands, numbers of arguments, etc.

SUBMIT reader.c and makefile1.

(B) Command Line Parser and Makefile

WRITE a C program in a file named parser.c with this functionality:

1. Print the prompt: "MY SHELL>"
2. Read one line of user input (a single command line) using `fgets()`
3. Determine the strings on the command line and store them in an array, i.e., something that functions like a `char *argv[]` array. Also compute the value for the number of arguments (this includes the name of the command), i.e., similar to `int argc`.
4. Print the argument count and the arguments from your argument array to `stdout`
5. Repeat (1)-(4) until an EOF condition (ex: the user types Ctrl-D)

WRITE a make file named makefile2, which is initially a copy of makefile1. Add a target named cmd_parser that builds your command line parser with gcc and with debugging enabled; the name of the executable should be cmd_parser.

TEST your program with several different commands, numbers of arguments, etc.

SUBMIT parser.c and makefile2.

C) Simple Shell and Makefile

WRITE a C program in a file named myShell.c with this functionality:

1. Print the prompt: "MY SHELL>"
2. Read one line of user input (a single command line) using `fgets()`
3. Determine the strings on the command line and put them into an array, i.e., something that functions like a `char *argv[]` array. Also compute the value for the number of arguments (this includes the name of the command), i.e., similar to `int argc`
4. Create a child process using `fork`
5. Call `execvp` to load the command input in (2) into the child process and run it
6. Call `wait` to pause the parent process until the child process completes
7. Repeat (1)-(7) until an EOF condition (ex: the user types Ctrl-D)

WRITE a make file named makefile3, which is initially a copy of makefile2. Add a target named my_shell that builds your shell with gcc and with debugging enabled; the name of the executable should be my_shell.

TEST your program with several different commands, numbers of arguments, etc.

SUBMIT myShell.c and makefile3.

(C) Implementing Background (&)

Copy your code from Part C to a new file named myBetterShell.c, with the following change:

1. Check each line of input for the "&" character at the end
2. If the "&" is found, run the child process in the background. This means the prompt should return immediately, and you can enter a new command before the first child finishes.
   Hint: see `waitpid`()

Part 2 –Simulator Exercise

Exercises 1-5 in the textbook on page 12 at the end of chapter 4.