**CS39002 Operating Systems Laboratory**
**Spring 2014**

**Assignment 2**
**Due on 31–Jan–2014, 1:00pm**

---

**Part 1**

Write a C program to implement the following game. The parent program *P* first creates two pipes, and then spawns two child processes *C* and *D*. One of the two pipes is meant for communications between *P* and *C*, and the other for communications between *P* and *D*. Now, a loop runs as follows. In each iteration (also called round), *P* first randomly chooses one of the two flags: MIN and MAX (the choice randomly varies from one iteration to another). Each of the two child processes *C* and *D* generates a random positive integer and sends that to *P* via its pipe. *P* reads the two integers; let these be *c* and *d*. If *P* has chosen MIN, then the child who sent the smaller of *c* and *d* gets one point. If *P* has chosen MAX, then the sender of the larger of *c* and *d* gets one point. If *c* = *d*, then this round is ignored. The child process who first obtains ten points wins the game. When the game ends, *P* sends a user-defined signal to both *C* and *D,* and the child processes exit after handling the signal (in order to know who was the winner). After *C* and *D* exit, the parent process *P* exits. During each iteration of the game, *P* should print appropriate messages (like *P*'s choice of the flag, the integers received from *C* and *D*, which child gets the point, the current scores of *C* and *D*) in order to let the user know how the game is going on.

Name your program *childsgame.c*.

**Part 2**

In this assignment, you start writing a command interpreter (shell). You will complete it in Assignment 3. The shell will give a prompt for the user to type in a command (from a set of commands), take the command, execute it, and then give the prompt back for the next command (that is, actually give the functionality of a shell). Your program in this assignment should do the following:

- Give a prompt "myshell> " for the user to type in a command
- Implement the following builtin commands:
  - **cd <dir>** : changes the directory to "dir"
  - **pwd** : prints the current directory
  - **mkdir <dir>** : creates a directory called "dir"
  - **rmdir <dir>** : removes the directory called "dir"
  - **ls** : lists the files in the current directory. It should support both ls without any option and with the option "–l"
  - **exit** : exits the shell

  The commands are the same as the corresponding Linux commands by the same name. Do "man" to see the descriptions. You can use the standard system calls like **chdir**, **getcwd**, **mkdir**, **rmdir**, **readdir** to implement the calls (standard C library functions are available for these; look them up).

  These commands are called *builtin* commands since your shell program will have a function corresponding to each of these commands to execute them; no new process will be created to execute them. (Note that all these commands are not builtin commands in the bash shell, but we will make them so in our shell).

- Any other command typed at the prompt should be executed as if it is the name of an executable file. For example, typing "a.out" should execute the file *a.out*. The file can be in the current directory or in any of the directories specified by the PATH environment variable (use **getenv** to get the value of PATH). The executable file should be executed after creating a new process and then **exec**'ing the file onto it. The parent process should wait for the file to finish execution and then go on to read the next command from the user. The executable can accept any arbitrary number of command-line parameters. Here is an example:

  myshell> *tar cvzf allprogs.tar.gz assignments/ myprogs/ robot.c seminar.c oldprogs/*

Give your program the name *myshell.c*.

---

Submit the two files *childsgame.c* and *myshell.c* separately.