CS39002 Operating Systems Laboratory
Spring 2014

Assignment 3
Due on 14–Feb–2014, 1:00pm

Part 1

In this part, you complete the shell that you started in Assignment 2. More precisely, you add the following features to your shell.

1. Support *background* execution of commands. Normally, when you type a command at the shell prompt, the prompt does not return until the command is finished. For background executions, the prompt returns immediately, the command continues execution in the background. Typing an *&* at the end of a command (like *a.out &*) should make it execute in the background.

2. Allow the redirection of the output of a program to a file using > and the redirection of the input of a program from a file using <. For example, typing *a.out* > *outfile* should send whatever was supposed to be displayed on the screen by *a.out* to the file *outfile* . Similarly, typing *a.out* < *infile* should make *a.out* take the inputs from the file *infile* instead of the keyboard. Both redirections may be used like *a.out* < *infile* > *outfile*.

3. Allow the redirection of the output of one command to the input of another by using the | symbol. For example, if there is a program *a.out* that writes a string *abcde* to the display, and there is a program *b.out* that takes as input a string typed from the keyboard, counts the number of characters in the string, and displays it, then typing *a.out* | *b.out* at your shell prompt should display *5* (the output *abcde* from *a.out* was fed as input to *b.out*, and *5*, the number of characters in *abcde*, is printed). Use the pipe command. Any number of redirections should be allowed (like *ls -l* | *grep -v A* | *wc* | *wc*). Moreover, pipes may be used in conjunction with input or output redirection (like *cat* < *input.txt* | *grep -v A* | *wc* | *wc* > *output.txt*).

4. When the shell is executing a command, the user may hit *Control+c* in order to terminate the execution of the command. However, hitting *Control+c* in the idle shell does not terminate the shell. The user should hit *Control+d* (or type *exit*) in order to terminate the shell.

Part 2

In this part, you are asked to implement a client-server system using message queues. In this system, we have a set of clients which send strings to the (single) server and the task of the server is to receive the string, change the case of the received string, and retransmit the string back to the right client (from the client it received the string). We assume that the two processes, client and server, communicate via two message queues “Up” and “Down” (known to all the processes a priori). The client reads a string from the standard input and sends it to the server via the Up queue, then waits for the server's answer on the Down queue. The task of the server is to convert characters from lower case to upper case and vice versa. For example, if the client sends the message *loweR case* via the Up message queue, the server will read the message, convert it, and send *LOWEr CASE* via the Down queue. When the client receives the message from the server, it prints it out. You may assume that the maximum size of any message is 256 bytes. Multiple clients must be able to connect to the Up and Down queues. However, the server must be intelligent enough to distinguish different clients and must send the converted string to the right client. The server should also print the time of receiving each message. Implement the client and server for this problem.

Sample Input-Output

Terminal 1
```
$./server &
$./client
Insert message to send to server: messaGe
Msg received at time: Fri Feb  7 12:51:47 IST 2014
Processed msg from server: MESSAgE
$
Msg received at time: Fri Feb  7 12:52:44 IST 2014
```

Terminal 2
```
$./client
Insert message to send to server: UPPER CASE
Processed msg from server: upper case
$
```

Submit the three files separately: the updated *myshell.c*, and *client.c* and *server.c* for Part 2.