CS39002: Operating Systems Lab
Spring 2017
Assignment 2
Due: Part 1 on January 23, 2017, 11:59 pm, Part 2 on January 30, 2017, 11:59 pm

1. Write a program to copy a file into another by following the steps below:
   - Takes two filenames, *file1* and *file2*, as arguments from the command line (use command line arguments, do not scanf)..
   - Parent opens *file1* for read and *file2* for write (use open call) If *file2* is not present it should be created; if it is present it will be overwritten. If there is any error, exit with a message (use perror).
   - If no error, parent creates two pipes, *pipe1* and *pipe2*.
   - Parent creates a child process.
   - Parent reads 100 bytes at a file from *file1* and sends it to child process through write end of *pipe1*.
   - Parent then waits to receive an acknowledgement (the integer 0) or error (-1) from the child in the read end of *pipe2*.
   - Child, on receiving each 100 bytes, appends it to *file2*, and then sends 0 to the parent through write end of *pipe2* if the write is successful, and -1 if there is an error. If there is any error, after sending the -1, the child prints an error message and exits. If the child receives less than 100 bytes, it sends 0, prints "file copied successfully" and exits.
   - The parent, on receiving 0, reads the next 100 bytes and repeats the process. It exits either on receiving a -1 from child, or on receiving a 0 after sending < 100 bytes for the last send (you can assume that the file size is not a multiple of 100 bytes, so the last send will have < 100 bytes).

   Name your file *fcopy.c*. Submit the single file *fcopy.c*.

2. In this assignment we will write a command interpreter (Shell). This will be similar to the shell that you use on a standard Linux system.

   The shell will give a prompt for the user to type in a command, take the command, execute it, and then give the prompt back for the next command (i.e., actually give the functionality of a shell). Your program for the assignment should do the following:

   - Give a prompt (like the $ prompt you get in Linux) for the user to type in a command. The prompt should have the current working directory name (full path) followed by the ">" sign (for ex., **/usr/home/agupta/temp>**).

   - Implement the following commands as **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 **ls** both without any option and with the option "-l"
- **cp <file1> <file2>:** copies the content of "file1" into "file2" only if the last modification time of "file1" is more recent than that of "file2". The filenames may contain a full pathname. You can assume "file1" and "file2" are simple files and not directories. No option of **cp** needs to be supported.
- **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 C library functions **chdir**, **getcwd**, **mkdir, rmdir, readdir, stat** etc. to implement the calls.

All calls should handle errors properly, with an informative error message. For example, cp will fail if the user calling it does not have read permission on "file1"; the call should print a proper error message. Look up the **perror** call.

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 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 command typed can have any number of command line arguments. You can assume that the full pathname of the file to be executed will be specified.

- 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 (for ex., a.out&) should make it execute in the background.

- Should be able to redirect the output of a program to a file using ">" and read 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. You should support both input and output redirection in the same command.

- Should be able to redirect 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, an 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. You should support at least 2 levels of piping (for ex., a | b | c).

To run your shell, write another C program that will create a child process and call an appropriate form of exec to run the program above from the linux shell. The parent process simply waits for the child to finish (execute the "exit" command), after which it also exits.

Name the C file for the shell *shell.c*. Name the C program above that runs your shell *run.c*. Submit both the C files.