National University of Computer and Emerging Sciences

School of Computing Spring 2020 Islamabad Campus

Operating Systems (Spring 2020)
Final Project
(Deadline: 15th May, 2020 09:00 PM)

Submission: All submissions MUST be uploaded on slate. Solutions sent to the emails will not be graded. To avoid last minute problems (unavailability of slate, load shedding, network down etc.), you are strongly advised to start working on the project from day one.
Only submit the source code files (i.e, .cpp, .h files and make file) combined in one .zip file. Submit zip file on slate within given deadline. You need to submit zip folder named as ROLL_NUM_SEC (e.g., 18i-1234_A).
Deadline: Deadline to submit project is **15th May, 2020 09:00 PM**. No submission will be considered for grading outside slate or after **15th May, 2020 09:00 PM**. Correct and timely submission of project is responsibility of every student; hence no relaxation will be given to anyone.
Plagiarism: **Zero marks** will be awarded to students involved in plagiarized. A code is considered plagiarized if **more than 20%** code is not your own work.

DEVELOPING A SHELL

In this project, you are required to develop a small shell, *gbsh*, which implements some of the features found in typical shells, like the *bash* (Bourne Again Shell) or *csh* (C-Shell). **Note:** *gb* is the abbreviation of introduction to operation systems (grundlagen betriebssysteme) in Deutsch language.

Because the C language is the standard for system programming in the Unix/Linux environment, we will also use it for implementing the shell. To get you started, we provide you with archive *pracex.tar*, downloadable along with this project description. The archive contains simplified *Makefile*. Create folder *gbs-spring20* in your home directory and extract *pracex.tar* to that folder. Alternatively, you may use any other directory of your choice, but we will refer to directory *gbs-spring20* in the following description.

Part 1 [40 marks]: In the first part, you will implement some basic shell functionality to get started. Some basic files for the shell are contained in folder *gbs-spring20*, and you should put all other files that you may need in this folder.

File *gbsh.c* should contain the main code for the shell. You may create other files if you want to split functionality in a modular fashion. The provided *Makefile* will built your program. If you use additional source code files, you need to add them to the *Makefile* in order for them to get built into the executable.

In this part, implement the following features of the first version of *gbsh*.

- **Prompt**: Create a loop that outputs a prompt that awaits user input. Once a command is entered and executed by the user by pressing ``return'', output the command to `stdout` and await the next user input. The prompt should have the following format:

  `<user>@<host> <cwd> >` (with space at the end)

Here, <user> should be your login name, <host> the host name you are logged in to, and <cwd> the current working directory. Use system calls to retrieve that information and do not hard-code that information in your program. [10 marks]

- **exit** command: Implement the *exit* command that quits the shell. [5 marks]
- **pwd** command: Implement the *pwd* command that prints the user's current working directory. [10 marks]
- **clear** command: Implement the *clear* command that clears the screen. [10 marks]
- For any other input, split the command line into single tokens and print each token followed by a new line to *stdout*. A token is a character sequence that is limited by a whitespace on either side. A whitespace can be a space, tab or the start and end of the line. Await the next command from the user after printing the tokens. [5 marks]

Part 2 [80 marks]: In this part, we will extend the shell by some more advanced commands and features.

a) **ls** command: Implement the list directory command, *ls <directory>*, which lists the contents of the directory specified by *<directory>*. You may use any format to display the directory contents, and you don't have to implement additional *ls* arguments. **Note:** You are not allowed to use *exec* system call. **[10 marks]**

b) **cd** command: Implement the change directory command, *cd <directory>*, which changes to the directory specified by *<directory>*. If no directory is specified, change to your own home directory. **[10 marks]**

c) **Environment:** Implement commands for listing, defining and un-defining environment variables. Use *environ* to list all the environment strings currently defined. Use *setenv <envar> <value>* to set the environment variable *<envar>* to *<value>*. If *setenv* is used with *<envar>* only, set the value for that environment variable to the empty string. Use *unsetenv <envar>* to undefine environment variable *<envar>*. Use appropriate output if a variable is already defined (in the case of *setenv*) or undefined (in the case of *unsetenv*). At startup, the shell environment should contain *shell=<pathname>/gbsh* where *<pathname>/gbsh* is the full path for your shell executable (not a hardwired path back to your home directory, but the one from which it was executed). **[10 marks]**

d) All other command line input is interpreted as program invocation which should be done by the shell forking and executing the programs as its own child processes. The programs should be executed with an environment that contains the entry *parent=<pathname>/gbsh* as described in (c).

- Hand over function calls like *top*, *ps* and *man csh* to see what's happening. **[10 marks]**

- Modify the program to avoid the creation of zombie processes. **[05 marks]**

e) Now add support for I/O redirection on either or both `stdin` and/or *stdout*. Use *>* for output redirection, and *<* for input redirection. Consider the following command lines:

```
<cmd> <arg1> <arg2>     >     <outputfile>
<cmd> <arg1> <arg2>     <     <inputfile>     >     <outputfile>
```

The first line will execute command *<cmd>* with two arguments and print the output to file *<outputfile>*. In the second line, the contents of file *<inputfile>* will be input to *<cmd>* and the result of the command will again be output to file *<outputfile>*. **[10 marks]**

Also, *stdout* redirection should be possible for the internal commands *pwd*, *ls*, `environ`. For output redirection (*>*) the *<outputfile>* is created if it does not exist and truncated if it does. **[05 marks]**

You can check if input and output redirection work together by executing *wc -m < wc-test.txt > wc-test.cnt*, which should write the number of characters in *wc-test.txt* (provided with archive *pracex.tar*) into the file *wc-test.cnt*.

Hint: To add I/O redirection, modify the child process created by *fork* by adding some code to open the input and output files specified on the command line. This should be done using the open system call. Next, use the *dup2* system call to replace the standard input or standard output streams with the appropriate file that was just opened. Finally, call *exec* to run the program.

f) Implement background execution of programs. An ampersand *&* at the end of the command line indicates that the shell should return to the command line prompt immediately after launching that program. **Hint:** The solution involves the *fork* function. **[5 marks]**

g) Programs can be run together such that one program reads the output from another with no need for an explicit intermediate file. In the following line,

```
<cmd1> | <cmd2>
```

command *<cmd1>* is executed, and its output is used as the input of *<cmd2>*. This is commonly called piping, since the | character is known as a ``pipe''. Pipes are an approach for inter-process communication by message passing, besides shared memory as discussed in the lecture.

- Add support for pipes to *gbsh*. **[5 marks]**

- Also allow the dynamic chaining of multiple pipes. **[10 marks]**

Hint: To do this, you'll need to use the *pipe* and the *dup2* system calls.

h) **Bonus:** Modify your program so that the SIGINT signal does not result in termination of the shell (i.e. your shell should ignore SIGINT). **[10 marks]**