CS232 Operating Systems
Assignment 02: Introduction to System Calls
Due : 23h55 Monday 20th October, 2019.

CS Program
Habib University

Fall 2019

1 Objectives

(a) Test the student's C programming skills.

(b) Test the student's ability to write clean readable code.

(c) Test the student's ability to use Linux System Calls.

(d) Test the student's ability do error handling in code.

(e) Test the student's ability to follow written instructions.

2 Description

In this assignment you will be doing C programming. It would help if you have completed the labs regarding C functions related to string manipulation, taking input from user, and using the Linux process management APIs.

3 Tasks

Q1. Create a program called mycat [1]

Implement [2]
a program in C which emulates the behaviour of Linux `cat` command. In particular your program should be able to do the following:

```
prompt> mycat f1 f2 // display contents of multiple files on terminal

prompt> mycat -n f1 // given the -n option, number the displayed lines

prompt> echo "cat" | mycat // be able to read from stdin
```

Q2. Make a mini shell [3]

We'll make a mini shell of our own. Letś call it hush Habib University SHell. The shell would:

[1] read about file descriptors, `open()`, `close()`, `read()`, `write()`, `lseek()`, `STDIN_FILENO`, etc.

[2] the word implement in this document means you have to write code for that part yourself; can't use an already existing command

[3] read about `fork()`, `exec()`, `wait()`, `waitpid()`, `getpid()`, `getenv()`, `setenv()`, `unsetenv()`, `int main(int argc, char* argv[], char* envp[])` , `kill()`

(a) When run, display a coool prompt of your choosing and prompt the user for input.

(b) When given a command with arguments, it would create a child process and run the command with its arguments. By default it would wait for the command to finish before displaying the prompt again. [4]

(c) Bash allows you to run multiple commands in one line when you separate them by the ampersand (&) character. If you type only one command followed by &, it runs the command in back ground and displays the prompt immediately. Hush should imitate the same behaviour.

(d) When given a command followed by a redirection operator (>, >>) followed by a filename, it should redirect the output of the command to the said file by overwriting or appending to it respectively.

(e) It should keep track of the names and PIDs of the commands it has launched that are still running. It should implement a command `myps` which should list all the names and PIDs of the commands that havent́ finished yet.

(f) We have seen environment variables maintained by the shell e.g. PATH, SHELL, etc. In Linux a process inherits the environment variables of its parent. A program can add, delete, modify, environment variables it has inherited. Our shell would:

- modify its PATH variable to add the current directory to its list of paths. This way we won't have to type the leading dot-slash (./) when running our executables.
- implement a command `mylsenv` which would list all its environment variables and their values.
- when typed a string VAR=xxxx on the command prompt, it should add the variable VAR to its list of environment variables assigning it the value xxxx. If the variable exists already, it should overwrite its old value with xxxx. Implement a command `show VAR` which would display the value of the variable VAR.

(g) It should implement the built-in commands `cd` and `pwd` which should update and print the current working directory respectively.

(h) It should exit if the user types `exit` at the prompt. It should check if there are any child processes still running and kill them before exiting.

4 Error Checking

When working with Linux System calls, it is extremely important that you always check the return values of the system calls to verify whether the call was successful or not. The function documentation would usually contain all the information you will need. You should always read the function documentation. You might also want to read about the use of `errno`, `perror()`, and `strerror()`.

Your programs should guard against invalid user input and in case of an invalid input, it should print a helpful error message.

You would lose marks if your program crashes during use.

It is the programmer's responsability to free any system resources i.e. memory , file descriptors, etc., they have acquired from the system. Your program should always free system

[4] you can't use the `system()` syscall here. Have to use `fork(), exec()`

resources once it is done using them.

Code should be properly indented, readable and commented.

You should always your compile your code using the -Wall option.

5 Submission Instructions

1. You submission shoud consist of .c files and a make file.
2. You can name your submission as LecX_gp0X_A2QX_XXXX.c:
3. You'll also submit a PDF file combining all the codes and the submitters' info. The name should follow the pattern LecX_gp0X_AX.pdf. A template will be provided later.
4. The final submission should combine all these file together and submit one .tar.gz file e.g. LecX_gp0X_A2.tar.gz. Explore the linux 'tar' command to archive and zip multiple files/directories together.

6 Rubric

Category	Marks
Followed submission insns	10 marks
Code was readable + Compiled without warnings + Program handles errors well	10 marks
Q1 working properly	20 marks
Q2 working properly	60 marks
Total	100 marks