

Lab 7 – ps, kill, and signals

Part I

Use shell terminal to launch a GUI-based application to be executed in the background. Here are examples to start image viewer *gimp* on CS server:

```
>>>>> gimp &
```

```
>>>>> gimp imagefile.ppm &
```

Use command *kill -9 pid* to terminate this application.

Part II

Repeat the above. This time, do not use the ampersand. The application is executed in foreground. Open another terminal. Use command *ps* to find out the process id of the application that you have launched, and then use command *kill -9 pid* to terminate it.

Part III

Write a C program where two child processes are created using *fork()*. The parent process and child processes all write into a shared file using system call *write()*. The file name is given from command line.

Define three strings like this:

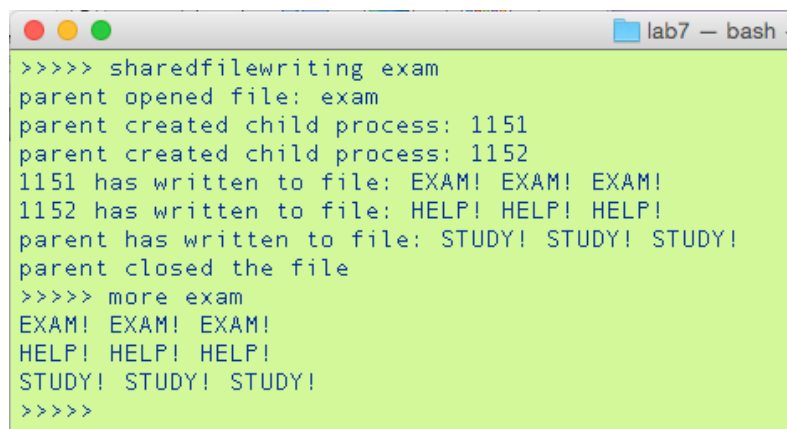
```
buf[0] = "EXAM! EXAM! EXAM!\n";  
buf[1] = "HELP! HELP! HELP!\n";  
buf[2] = "STUDY! STUDY! STUDY!\n";
```

The parent process will open the file using *open()* before the fork. The first child will write the content of *buf[0]* into the file. Then the second child will write the content of *buf[1]* into the file. Finally, the parent will write the content of *buf[2]* into the file and close the file. Add *sleep(5)* after each *write()* statement.

Use signal to coordinate the access to the shared file, so that the file is accessed with mutual exclusion.

In addition to writing to the file, both parent and child processes should also write some related information to the terminal according to what the sample shows.

Sample run:

A terminal window titled 'lab7 - bash' with a light green background. It shows the execution of a program named 'sharedfilewriting'. The program creates a file named 'exam' and then creates two child processes, 1151 and 1152. Process 1151 writes 'EXAM! EXAM! EXAM!' to the file, and process 1152 writes 'HELP! HELP! HELP!' to the file. The parent process then writes 'STUDY! STUDY! STUDY!' to the file and closes it. Finally, the user runs 'more exam' and sees the concatenated output of both child processes.

```
>>>> sharedfilewriting exam
parent opened file: exam
parent created child process: 1151
parent created child process: 1152
1151 has written to file: EXAM! EXAM! EXAM!
1152 has written to file: HELP! HELP! HELP!
parent has written to file: STUDY! STUDY! STUDY!
parent closed the file
>>>> more exam
EXAM! EXAM! EXAM!
HELP! HELP! HELP!
STUDY! STUDY! STUDY!
>>>>
```