**Operating System Project 1**

**Linux Process Concurrency and Synchronization**

**Objectives:**

* Design and develop systems programs using C/C++
* Effectively use Linux system calls for process control and management, especially, *fork, exec, wait, pipe and kill system call API*.
* Concurrent execution of processes.
* Solve inter-process communication problems during concurrent execution of processes.
* Use semaphores.

**Problem Statement:**

This project is to be developed in several small steps to help you understand the concepts better.

1. Write a C program, deploy the ***signal.*** Let the program print “sleep\n” per second. When the progress receives the SIGINT signal, it starts to print “hello\n” every two seconds, while it still prints “sleep\n” at the original frequency. The progress will exit after a certain time, e.g. 10s. (Sig.c → Sig)
2. Write a C program that makes a new copy of an existing file using ***system calls***for file manipulation, such as ***open, read, write***. The names of the two files, source and the destination are to be specified as command line arguments. Open the source file in read only mode and destination file in read/write mode. While the main function will carry out file opening and closing, a separate copy function needs to be written to do the actual copying. Copying can be done in blocks of suitable size. (MyCopy.c → MyCopy)

An optional usage can be:

>>./MyCopy srcFile desFile

1. Write a C program that forks two processes one for reading from a file (source file) and the other for writing (destination file) into. These two programs communicate using ***pipe***system call. Once again the program accomplishes copying files, the names of which are specified as command-line arguments
2. Write a C program that forks two processes one for reading from a file (source file) and the other for writing (destination file) into. These two programs communicate using ***pipe***system call. Once again the program accomplishes copying files, the names of which are specified as command-line arguments. (PipeCopy.c → PipeCopy)
3. Write a C program that forks two processes one for reading from a file (source file) and the other for writing (destination file) into. These two programs communicate using ***Socket***. Once again the program accomplishes copying files, the names of which are specified as command-line arguments. (SockCopy.c → SockCopy)
4. Write a shell-like program that illustrates how Linux spawns processes. This simple program will provide its own prompt to the user, read the command from the input and execute the command. It is sufficient to handle just ``argument-less'' commands, such as ***ls*** *and* ***date***. (MyShell.c → MyShell)
5. Make the mini-shell (from the previous part) a little more powerful by allowing arguments to the commands. For example, it should be able to execute commands such as ***more filename*** *and* ***ls –l ~/tmp***etc. (MoreShell.c → MoreShell)
6. Add to the mini-shell ability to execute command lines with commands connected by pipes. Use ***dup***system call to redirect IO. Example: ls -l | wc. (DupShell.c → DupShell).

**Implementation Details:**

In general, the execution of any of the programs above will is carried out by specifying the executable program name followed by the command line arguments.

1. Use Ubuntu Linux to test your programs.
2. See the man pages for more details about specific system or library calls and commands: fork(2), pipe(2), execve(2), execl(3), execlp(3), cat(1), wait(2) etc.
3. When using system or library calls you have to make sure that your program will exit gracefully when the requested call cannot be carried out.
4. One of the dangers of learning about forking processes is leaving unwanted processes active and wasting system time. Make sure each process terminates cleanly when processing is completed. Parent process should wait until the child processes complete, print a message and then quit.
5. Your program should be robust. If any of the calls fail, it should print error message and exit with appropriate error code. Always check for failure when invoking a system or library call.

**Material to be submitted:**

1. Compress the source code of the programs into **StudentID\_p1.tar** file. Use meaningful names for the file so that the contents of the file are obvious. It is better to provide a single **makefile** that makes the executables out of any of the source code in the compressed file. Enclose a **README** file that lists the files you have submitted along with a one sentence explanation.
2. Only internal documentation is needed. Please state clearly the purpose of each program at the start of the program. Add comments to explain your program. (-5 points, if insufficient.)
3. Test runs: It is very important that you show that your program works for all possible inputs. Submit online a single typescript file clearly showing the working of all the programs for correct input as well as graceful exit on error input.
4. Write a simple **experiment report** to explain your work, enclose your **PDF format** reports into the **tar** file.
5. Also, you should upload your **Prj1+StudentID.tar** file to FTP.  
   • Address: ftp://public.sjtu.edu.cn  
   • Username: qndeng  
   • Password: public

• /upload/2018二专操作系统/Project1

1. Due date: **April 15, 2018**, submit on-line **before midnight**.