CSE4/521 Introduction to Operating System Spring 2009
Project 1: Realizing Concurrency using Unix processes and threads

Objectives:

Learn to:

- Design and develop systems programs using C/C++
- Effectively use Unix system calls for process control and management, especially, *fork, exec, wait, pipe and kill system call API.*
- Concurrent execution of processes.
- Use Posix Pthread library for concurrency.

Problem Statement:

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

1. Write a C program that encrypted copy of an existing file using ***system calls*** for file manipulation. 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. Use substitution encryption where each character is substituted by another pre-determined character. (Encrypt.c → Encrypt) Substitution chart will be provided for you.

2. Write a C program that creates a new process to encrypted copy of the file using the Encrypt. This program should spawn a new process using ***fork*** system call. Then use ***execl*** to execute Encrypt program. The source and destination file names presented as command-line arguments should be passed to *execl* as system call arguments. The main process waits for completion of copy operation using *wait* system call. (ForkEncrypt.c → ForkEncrypt)

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 ***pipe*** system call. Once again the program accomplishes copying files, the names of which are specified as command-line arguments. (PipeEncrypt.c → PipeEncrypt)

4. Use various system calls for ***time*** to compare the three versions of the file copy programs as specified above. Use very large files (we will provide) to get some reasonable value for time. You may use delays to simulate finite processing time.

5. Write a shell-like program that illustrates how UNIX spawns processes. This program will provide its own prompt to the user, read the command from the input and execute the command. Let the shell allow arguments to the commands. For example, it should be able to execute commands such as ***more filename and ls –l*** etc. (MyShell.c → MyShell)

6. Add to this 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).

7. Write Pthread-based application to create an n-way concurrency using n threads. Each thread will wait for a random time and generate a random and synchronize with the main controller thread. The main thread then accumulates (sums up) all the returned values and prints them out. You may appropriate synchronization primitive to control the threads. (Barrier primitive?)

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. See the man pages for more details about specific system or library calls and commands: UNIX fork(2), pipe(2), execve(2), execl(3), execlp(3), cat(1), wait(2) etc.

2. 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.

3. 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.

4. 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 Prj1.tar file. Use meaningful names for the file so that the contents of the file are obvious. A single makefile that makes the executables out of any of the source code should be provided in the compressed file.
2. Submit a README file that lists the files you have submitted along with a one sentence explanation. Call it Prj1README
3. 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.
4. 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.
5.Submit. **submit_cse421 Prj1.tar Prj1README Prj1Script**
6.Due date: 2/20/2008, submit on-line before midnight.