Project 1: Realizing Concurrency using Linux processes and threads

February 20, 2023

1 Objectives

- Design and develop systems programs using 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.
- Use Posix Pthread library for concurrency.
- Use graphic tools or programs to analyze data.

2 Problem Statement

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

1. **Basic requirement:** Write a C program that forks two processes to copy file, one for reading from a source file, and the other for writing into a destination file, using system calls. These two processes communicate using **pipe** system call. The name of source file and destination file are to be specified as command line arguments.

   Advanced requirement:

   - Use a suitable size of buffer in order to copy faster.
   - Use various system calls for time to compare the performance of this program with at least 8 different buffer size.
   - Use graphic tools or programs for plotting the execution time over different buffer size and try to explain the result.

   More details of requirement:

   - Source file name: Copy.c
   - Executable file name: Copy
   - Command line: ./Copy <InputFile> <OutputFile> <BufferSize>

2. **Basic requirement:** Write a shell-like program that illustrates how Linux spawns processes. It should handle the commands with arguments and the commands connected by pipes. **Example**: *ls -l | wc*. Use **dup** system call to redirect I/O.

   Advanced requirement: Write the above shell program as a server. When the server is running, clients are allowed to connect to the server and run the commands. The communication between client and server should be implemented using Internet-domain socket.

   More details of requirement:

   - Server should work in an infinite loop.

- Using messages between server and client is suggested.
- There must be a command exit to disconnect the connection between server and client.
- There may be more than one '|' in the command line.
- Try to support more than one client.
- Server will be tested by using **telnet**, the command line is as follows.
- Command line for telnet: telnet <IPAddress> <Port>
- Source file name: shell.c
- Executable file name: shell
- Command line: ./shell <Port>

3. **Basic requirement:** Matrix multiplication is widely used in scientific computing. From linear algebra, it's known that partition matrix can be used to do matrix multiplication. The main idea of this task is to implement matrix multiplication using any matrix partition based algorithm you like with time complexity $O(n^3)$. Please follow the requirement below.

   - Find a partition-based matrix multiplication algorithm with time complexity of $O(n^3)$.
   - Implement a single-thread program for matrix multiplication.
   - Implement a multi-thread program for matrix multiplication with the help of partition matrix.

   Advanced requirement: Compare the performance of implementations with different number of threads and different size of matrix. Draw a diagram and try to explain the result. Please use at least 6 different numbers of threads and 8 different sizes of matrix.

   More details of requirement:

   - To make this problem easier, the sizes of two matrix are set to $n * n$, where n is a power of 2. n is up to 4096.
   - In the single-thread program, the input data should be read from the file "data.in", while the output data should be write to the file "data.out". For more details, please see the sample input and output files.
   - In the multi-thread program, 2 versions are required.
     - Same as the single-thread program's requirement 3b.
     - Generate those two matrix **A**, **B** randomly with a given size. Output **A**, **B** and **AB** to file "random.out". Please see the sample for the format.
   - Implementations using algorithms other than partition matrix in the multi-thread implementation are also acceptable. The requirement of using more than 1 thread in the program is mandatory. Better algorithms can get several points of bonus.
   - Source file name: single.c, multi.c
   - Executable file name: single, multi
   - Command line:
     - ./single
     - ./multi //When no command line argument given, the program automatically read input from "data.in", and output to "data.out".
     - ./multi <Size>

3 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.

- Use Ubuntu Linux and GNU C Compiler to compile and debug your programs.

- See the man pages for more details about specific system or library calls and commands: fork(2), pipe(2), cat(1), wait(2), telnet(1), etc.
- 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.
- One of the dangers of learning about forking processes is leafing unwanted processes active which wastes system resources. So please make sure that each process/thread is terminated cleanly when the program exits. A parent process should wait until its child processes finish, print a message, and then quit.
- Your program should be robust. If any of the calls fails, it should print an error message and exit with an appropriate error code. So please always check for failure when invoking a system or library call.

4 Material to be submitted

- Compress the source code of the programs into Prj1+StudentID.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. Enclose a README file that lists the files you have submitted along with a one sentence explanation. Call it Prj1README.
- Please state clearly the purpose of each program at the start of the program. Add comments to explain your program.
- 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.
- Submit a short report to show the performance analysis. You can also write down any other things you find when doing your project.
- Send your Prj1+StudentID.tar file to the Canvas.
- Due date: Mar. 28, 2023, submit on-line before midnight.