CS 4348/5348 Operating Systems -- Project 1

The goal of this project is to let you get familiar with programming in Unix (Linux) systems, including the steps to get a program to run and how to use some important Unix system calls. Most of the needed skills for this project are covered in CS 3377. If you do not have the background course, you can learn the skills from online tutorials or the given reference books for the projects.

Some code files to be used for the project and links to information that may be helpful for completing the project are available at https://personal.utdallas.edu/~ilyen/course/os/home.html.

1 The Cal Process

You need to create a program "**cal.c**" or "**cal.cpp**" to perform some calculations. The input file is "**cal.in**". The file contains $M + 1$ lines. The first line contains a single integer, M. The following M lines are the computing commands. Each line includes a command and 2 integers. The command can be "**quadratic**, "**triples**", or "**mod-sum**". Function "quadratic" finds the 2 roots of $x^2 + ax + b$ and a and b are the two integers given in the same line. Function "triples" finds a list of Pythagorean triples that satisfies $x^2 + y^2 = z^2$, with x, y, z all being integers. The two input integers are the maximum values for x and y. Function "mod-sum" computes $\sum_{n=1}^{x} n \ \% \ mod$ and x and mod are the two integers given in the same line.

The C code for quadratic, triples, and sum functions are given in files "quadratic.c", "triples.c", and "sum.c", respectively. You just need to implement the main program in another file "cal.c" or "cal.cpp".

2 Program Preparation

Prepare a **makefile** for compiling the program comprised of "quadratic.c", "triples.c", "sum.c", and "cal.c" or "cal.cpp". The generated executable should be named as "**cal.exe**". Then, write a shell script "**execute**" to compile the program using the "**make**" command and execute cal.exe with an input file "**cal.in**". In your script, the execution of cal.exe should be done twice. In the first invocation of cal.exe, let the output be redirected to "**cal.out**". In the second invocation of cal.exe, **pipe the output to the "fgrep" command** and find string s in the output. Your script "execute" should take string s as an input argument.

Note that you should not use #include to include the three C code files into your cal.c or cal.cpp. You should compile each C code file into an object code file (".o") separately and then link the ".o" files together.

3 Process Creation

Try to familiarize yourself with the Unix "fork" system call for process creation and the basic process related concepts. Write a program "**admin.c**" or "**admin.cpp**" to use fork() to create a child process. Let the parent process be the "Admin" process. From the Admin process, fork a process that runs "cal.exe" you created earlier. In the Admin process, print out:

Admin: returned-pid, PID

The returned-pid is the value returned from fork(). PID is the parent process ID and you can use getpid() to obtain it. Both returned-pid and PID are integers. Do not forget to use "wait" system call to wait for the child process (cal.exe) to finish.

4 Pipe for Process Communication

Instead of letting the Cal Process directly reads the input file cal.in, we now let the Admin Process reads the input commands from cal.in and send each input command to the Cal Process via a **pipe**. Modify your code in admin.c(pp) and cal.c(pp) to realize this.

For cal.c(pp), you do need to submit the original version to allow us to test your shell script “execute”. So, you should modify cal.c(pp) into cal-new.c(pp). In cal-new.c(pp), the input commands should be from the pipe. Also, the Admin Process should send the command one at a time and the Cal Process should send an acknowledgement (via a pipe) after finishing executing each command to inform the Admin to send the next command. For termination, you can let Admin and Cal both examine the input command to make the termination decision.

Note that in Step 3, you probably used “exec()” system call to run cal.exe in the child process. To make the use of pipe easier (avoid the need of named pipe), you should change the main() function of cal-new.c(pp) into a regular function so that admin.c can call it in the child process part after fork.

For admin.c(pp), you can directly update it and only submit the updated version. Now the Admin Process needs to read the input commands from a file and implement message passing via pipe. If you feel comfortable, you can merge the implementation of Steps 3 and 4 together.

5 Thread Creation

Complete the code “**pthread.c**” posted on the web page given above. Code segments should be added in pthread.c for thread creation and thread join at places marked “CODE NEEDED”. Check the comments and see what are expected. After finishing editing, compile and run the program. During program execution, you may notice that the tcounter value is not summed correctly. You can lower the value “Max” to something like 10 and see what happens. Try to figure out the cause for the problem and explain how it happens in a text file “**pthread.problem**”. To make your explanation clear, you should give a very simple example of execution sequence of the instructions in multiple threads to illustrate the issue. You may want to consider only two threads and a very small Max value to simplify the example execution sequence. For example:

- T1: load a: … (describe what happens)
- T1: add b: …
- T1: add c: …
- Switch out T1, switch in T2
- T2: load d: …
- …

To help with figuring out the problem, check the slides about the concepts of *shared address space* for multiple threads and consider thread switch at the *instruction level*.

6 Project Submission

You need to submit your program **before** midnight (11:59pm) of the due date (check the web page for the due date). Use UTD elearning system for submission: "elearning.utdallas.edu".

Your submission should include the following:

- All source code files making up your solutions to this assignment.
  - “cal.c” or “cal.cpp”, “makefile”, “execute”
  - “admin.c”, “cal-new.c” or “cal-new.cpp”
  - “thread.c”, “thread.problem”
  - Any additional files you have for the project

- A “readme” file
  - Indicate whether you use c or cpp
  - Indicate how to compile and run your programs in case it deviates from the specification
  - If you did not finish the project, you need to specify which part(s) you have completed and which part(s) are missing
- You can zip or tar all your submission files together and then submit it on elearning