Spring 2022
CPSC 3220 / ECE3220 Assignment 1

Submission to Canvas:

zipped archive named *asg1.tar.gz* that contains
three source code files: asg1.c, area.c, perimeter.c
do not submit any executable files

Grading:

Programs that do not compile get 0 points.
Programs with corrupt/empty/incomplete archives get 0 points.

(5) number of CL arguments is checked, error message printed if incorrect.
(12) Each of the 2 child processes executing different routines via exec calls.
(12) Two threads are executing correct code and returning values correctly.
(15) Program output matches shown output (pids/tids will be different from the sample)
- child1 should print its data before child 2 prints its data (5)
- threads print their data after children (5)
- parent prints last (5)

(3) Code is formatted nicely and has adequate number of comments
(3) Code is zipped in a tar.gz archive without any subdirectories (once unzipped, it should contain your source code files, not another subdirectory)

This is a *team of 1 or 2 assignment.* You are not allowed to talk to anyone except your teammate, your teacher and you TA.

Program

1. Using C programming language, write a program named *asg1.c.* The program will take two integers and a character string as command line arguments. You should give an error message, if user did not provide the required command line arguments, and exit the program.

In your program you will create a struct that has two integer members *num1* and *num2* and a char string *message.* They will be initialized to the CL arguments provided by the user.

Your program will have three processes that can be created by cleverly using the fork() system call.

The first child will be given it's own routine to execute. This routine is called *area* and the code for this routine lives in the file named *area.c.* This can be accomplished by passing the name of the executable file (along with other necessary arguments) to the execl(...) call. Please study the examples in

Files/code_examples/processes folder. This routine will calculate the square area by multiplying *num1* and *num2*. It will then print its own pid, its parent's pid, the result of the calculation to the standard output (as shown below) and exit.

The second child will execute its own routine. This routine is called *perimeter* and the code for this routine lives in the file named *perimeter.c.* As above, this will be accomplished by passing the name of the executable file (along with other necessary arguments) to the execl(...) call. This routine will calculate the perimeter by using variables *num1* and *num2*. It will then print its own pid, parent pid, the result to the standard output and will exit.

The parent will create two threads. Each thread will take a pointer to a function and your struct as parameters to thread creation call. The first thread will calculate the ratio of the two numbers by dividing *num1* by *num2,* and will return that result to the parent via *thread_exit* call. The second thread will reverse the *message* string, and also pass the reversed string to the parent. Each thread will print its own tig and the result of the calculation/reversal to standard output before returning result to the parent.

2. Normally, if you have a number of processes, each may be scheduled to run at a different time, may be preempted at least once, and their output may be interleaved. In you program you will need to figure out a way for your processes to produce output that is not interleaved. (hint – a process can be put to sleep to allow other process to finish printing, you do not need to use any locks or Cvs for synchronization)

3. When you execute your program, your output should be *exactly* as the one shown below, with the exception of the process ids - those will be different from the ones shown.

4. Do not forget that the parent process needs to wait for all the children to finish executing. Process creating threads needs to call join on each thread as well.

To compile your program you first need to compile each routine and the main file and rename their executables:

```
gcc area.c -o area
gcc perimeter.c -o perimeter
gcc asg1.c -o asg1 -pthread
```

To run:

```
./asg1 10 5
```

Output: (your pids/tids will be different than the ones shown below)

Child1: pid 39810, ppid 39809, area is 50
Child2: pid 39811, ppid 39809, perimeter is 30

Thread 1: tig 12345678, ratio is 2
Thread 2: tig 23456789, message is “sregiToG”

Parent: pid 39809, ratio: 2, message: “sregiToG”

Omissions

Some of the details were intentionally omitted to encourage you to experiment and be creative. Again, please study the code examples on Canvas. They have everything you need to compete this assignment.

Recommendations:

1. Examine code examples on Canvas. Compile and run them and understand how they work.

2. Create a skeleton program that forks once and try printing pids first.

3. Write an area routine and test it separately using CLAs. Same with perimeter.

4. Get one of the children to work, then move on to the second one.

5. Create one thread and get it to work. Pass parameters to the thread function and print them inside the thread function to make sure you have the right value in the correct format. Make sure you are passing correct values to the parent (casting will/may be necessary). When parent gets the return value, print is to ensure your casting is correct. When this is all working, create the second thread.

6. Practice creating tar ball ONLY after you have backed up your files, just in case.

7. Compile often. If you write more than 10 lines of code without compiling it, you are doing it wrong.

8. Use commenting out and print statements for debugging.