Summer 2021
CPSC 3220 / ECE3220 Assignment 1

Due date:

June 12th before 11:59pm. You can submit within 24 hours after the due date with a late penalty of 10 points, deducted after your work is graded.

Submission:

Submit source code files to Canvas in a zipped archive named asg1_*username*.tar.gz, where *username* is you CU username (please do not submit any executables)

Task at Hand

1. In this assignment you will write a C program named *asg1.c* that will have a total of 2 processes: a parent, and a child.

2. Child process will be given its own routine to execute (file named child_function.c) and will be passed an integer that will come from the command line as the first argument. Child process will increment that integer by 10 and print the result to the standard output (screen). This can be accomplished by passing the name of the executable file (along with that integer) to the execl(...) call. Printing out the resulting value will be done inside that function.

3. Parent process will create two threads, call a function named thread_func and pass two integers (second and third CLA) to the corresponding threads. Inside that function this integer will be multiplied by 10 and results will be returned to the parent via calls to thread_exit/thread_join. The parent will then print all his information (pids, data values) from the main (NOT from the thread function). Please study the examples in Files/code examples/processes and threads folder.

4. When you execute your program, your output should be exactly as the one shown below, with the exception of the process/thread ids - those will be different from the ones shown. Also please keep in mind that different tasks may be scheduled to run at a different time than in the example below, so your output may appear in a different order.

Hints (hopefully helpful)

1. Work incrementally, save/compile/run often. First get one of the processes to work, then the other one.
2. Study the execl code example with count.c that does almost exactly what your child needs to do. The first CLA is passed to the child function via the execl call.
3. Do not forget that the parent process needs to wait for the child to finish executing.
4. Declare a void * variable to hold integers you will pass to threads, which will make it easier to deal with pointers.

5. Use a lot of debugging statements to see what values you are getting at different points of execution. This will be especially helpful when you are dealing with returning and casting pointers.
6. Do not forget to include header libraries for threads and processes and use -pthread to compile file that handles threads.
7. Have a plan A and plan B, do not wait till the last moment to start this assignment, as unforeseen events happen at the least convenient time.

To compile:

Compile your child_function code and then main file and rename their executables to something other than the generic a.out. You are nor required to have/submit a makefile.

```
gcc child_function.c -o child_function
gcc asg1.c -o asg1 -pthread
```

To run:

```
./asg1 17 11 22
```

(Note: 17 will go to the child via execl call and 11 and 22 to the threads in thread_create)

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

```
Child process: PID 39809 and PPID 39808, Initial value: 17, Result:  27

Parent process: PID 39808 and PPID 39676
    Thread 324723463634, Initial value: 11,  Result is: 110
    Thread 324723463642,  Initial value: 22,  Result is: 220
```

Grading Rubric:

Programs that do not compile (for any reason) get 0 points.
Incomplete/corrupt archives get 0 points.

Child is executing its own routine (execl call with correct arguments and math). (10)
Child prints its pid (1), ppid (1), initial value (1) and resulting value (1) from child_func

Parent creates two threads and passes correct arguments to each thread_create call. (14)
Parent prints from main its pid (1), ppid (1), and
for each thread (also from main): thread id(1), initial(1) and resulting value (2)

Sanity checks - check for number of CLA and give error message if needed
(do not need to check if user entered correct variable types) (3)
Check that calls to create/fork do not return errors (2)

Code is formatted nicely and has adequate number of comments (4)
Code is zipped in a tar.gz archive (3)

This is an *individual assignment* and will be checked by Turnitin and other honesty-encouragement tools. You can answer only conceptual questions from your classmates without giving out any code/implementation detail. Please contact your teacher or TA if you have questions.