Spring 2024
CPSC/ECE3220 Assignment 1

This is an *individual assignment.* You are not allowed to discuss the solution or any details with anyone except except your teacher and your TA.

Submission to Canvas:
zipped archive named *asg1.tar.gz* that contains two source code files: asg1.c and hash.c. It should not contain any subdirectories or executable.

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

(4) number of CL arguments is checked, error message is printed if incorrect. Allow user to try 3 times before exiting.

(15) First child process correctly executes hashing routine via exec call. Result is printed on the screen by the child, along with its own id and ppid,

(15) Second child correctly creates a thread that calculates days until, prints its own id, and returns values correctly to the child process that prints it. Casting is correct and results in correct value.

(2) Parent waits for its child processes to finish.

(8) Program output matches shown output (pids/tids will be different from the sample)

(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)

Program
1. First write a small program called *sha512hash.c* that takes one command line argument (character string) and calculates the SHA512 hash value of it. This value will be 86 characters long and will start with 6. This will later be executed by your child process. You will compile it to the executable named *sha512hash*.

2. Using C programming language, write a program named *asg1.c.* The program will take a character string variable named *word* (that will be later passed to the first child process via one of the exec calls) and any future

date in the format mm/dd/yyyy as command line arguments. You should give an error message, if user did not provide the required number of command line arguments, and allow user to try again, up to three times.

3. Your program will create the first child process using the fork() system call. This child will be given it's own routine to execute – sha512hash, and the first CL argument that you passed to your program (*word)* will be passed to that routine via the exec system call as a parameter to that routine. (This is similar to the count.c example we looked at in class) This child will print its own pid, its parent's pid, and the result of hashing to the standard output (as shown below) and exit.

4. The second child will execute its own routine. In this routine the child will create a thread that will take the struct as a parameter and calculate years, months and days left until the future date entered on the CL. It will then return its value to the child process to print. This return can be done via a struct, since you need to return three values. Remember that thread function pointer will point to a function that takes a void* and returns a void*.

5. 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 your hashing routine and the main file and rename their executables:

```
gcc sha512hash.c -o sha512hash
gcc asg1.c -o asg1 -pthread
```

To run:

```
./asg1 hello 11/04/2024
```

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

```
Child1: pid 39810, ppid 39809, SHA512 of hello is $6$15626378dhu$sdkn...
Child2: pid 38839, ppid 39809
    Thread 1: tid 123458474646678
    Time until 11/04/2024 is X days, Y months and Z years
```

Omissions

Some of the details were intentionally omitted to encourage you to experiment and be creative. 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 hashing routine and test it separately using CLAs.

4. Get the first child to work, then move on to the second one.

5. Create a 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.

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.