Fall 2024
CS3413
Lab 4

For this lab, please use the skeleton C file provided on D2L. Part of the skeleton is commented out, as you are to find where the functions are supposed to be used.

1. Write a C program to create two child processes using *pthread_create().* Create a global *integer* variable (**gv** initialized to 1) that should be incremented by one 100 times by one pthread (running the "inc" function) while the second pthread does the same but **decrements** the same variable by one (in the "dec") function. Have the main function wait for the two pthreads to complete and then print the value of the variable. Do not forget to join the pthreads.
2. Run your program several times. Do you always see 1 as the result? Explain in the comments in the file.
3. Alter your program for the threads to repeat addition and decrementing 1000 times
4. Run your program several times. Do you always see 1 as the result? Explain in the comments in the file.
5. Alter your program for the threads to repeat addition and decrementing 10000 times
6. Run your program several times. Do you always see 1 as the result? Explain in the comments in the file.
7. Modify your program so that your child processes protect the global variable using *pthread_mutex_lock()* and *pthread_mutex_unlock()*.
    a. You can declare your mutex as a global variable
    b. Do not forget to initialize your mutex in the main before using it and destroy it after you no longer need it. From the manual:
        i. pthread_mutex_init(&mutex, NULL);
        ii. pthread_mutex_destroy(&mutex, NULL);
8. Run your program several times. Do you always see 1 as the result? Explain in the comments in the file.
9. The second part of your task for the day is to create threads for the "plus" and "minus" functions. This should be done in a way that makes sure that + and – are always alternating. You need to use the semaphores for that. Recall that *sem_wait()* and *sem_post()* functions can be used for ensuring the execution order between statements. Hint: Initialize the value of semaphore to 0; and do not forget to join the two threads.
10. Run the program multiple times to make sure it always generates the same output.
11. Comment out the usleep() statement in the plus function and try running the program multiple times. Observe the output, and recall that sleep "helps" synchronization, as nothing prevents the plus function from signaling the semaphore repeatedly. Note that

relying on sleep for synchronization is incorrect, as it introduces delays that potentially can be avoided.

12. Uncomment the usleep statement. Reduce the maximum number of iterations of a loop in the plus function to three.
13. Compile and run your program, explain in the comments in the file what happens.