```
----------------
PL3 Class Script
----------------

- Race conditions

    Example of a race condition

- Synchronization mechanisms for processes and threads
    - Semaphores
        - sem_open()     - create/open a semaphore
        - sem_post()     - increment a semaphore
        - sem_wait()     - decrement a semaphore
        - sem_getvalue() - get the value of a semaphore
        - sem_unlink()   - remove the semaphore

    - Mutexes
        - similar to a binary semaphore (locks and unlocks) that protects access to critical sections of the code

        - pthread_mutex_init()    - creates a mutex
        - pthread_mutex_destroy() - removes a mutex
        - pthread_mutex_lock()    - acquires a mutex
        - pthread_mutex_unlock()  - releases a mutex

        example:

            #include <stdio.h>
            #include <pthread.h>

            #define NUM_THREADS 10

            int counter = 0;
            pthread_mutex_t mutex;

            void* thread_fn_1(void *arg)
            {
                // pthread_mutex_lock(&mutex);
                for (int i = 0; i< 10000000; i++) {
                    counter = counter + 1;
                }
                // pthread_mutex_unlock(&mutex);

                pthread_exit(NULL);
            }

            int main()
            {
                pthread_t threads[10];
                int i = 0;

                pthread_mutex_init(&mutex, NULL);

                for (i = 0; i< NUM_THREADS; i++) {
                    pthread_create(&threads[i], NULL, thread_fn_1, NULL);
                }

                for (i = 0; i< NUM_THREADS; i++) {
                    pthread_join(threads[i],NULL);
                }

                pthread_mutex_destroy(&mutex);

                printf("counter = %d\n", counter);
                return 0;
            }
```

Note: Remove the comments from the lock/unlock instructions and execute the code. What is the difference?

- Conditional variables

    - allows thread synchronization using the value of the data
    - it is up to the programmer to implement the logic that checks the value of the data and decide to wait or release the thread waiting in the conditional variable

    - pthread_cond_init() - initializes a conditional variable in the process
    - pthread_cond_wait() - waits until the conditional variable is released
        - The function blocks and suspends the thread, automatically releasing the mutex, which allows the conditional variable to be used by other threads
    - pthread_cond_signal() - signals (releases) at least one of the threads that is blocked in the conditional variable
    - pthread_cond_broadcast() - signals (releases) all the threads blocked in the conditional variable
    - pthread_cond_destroy() - removes a conditional variable (it can be reinitialized with pthread_cond_init())

    example:

```
        #include <stdio.h>
        #include <pthread.h>

        #define NUM_THREADS 2

        int value = 0;
        pthread_mutex_t mutex;
        pthread_cond_t  cond_var = PTHREAD_COND_INITIALIZER;

        void* thread_fn_1(void *arg)
        {
            pthread_mutex_lock(&mutex);
            while(value == 0){
                printf("thread_fn_1: value is %d - going to wait\n", value);
                pthread_cond_wait(&cond_var, &mutex); // mutex is released after
wait is called so that thread 2 can access the critical section
                printf("thread_fn_1: value is %d - after waiting\n", value);
            }
            pthread_mutex_unlock(&mutex);

            pthread_exit(NULL);
        }

        void* thread_fn_2(void *arg)
        {
            pthread_mutex_lock(&mutex);
            value = value + 1;
            if(value > 0){
                printf("thread_fn_2: value is %d - going to signal\n", value);
                pthread_cond_signal(&cond_var);
            }
            pthread_mutex_unlock(&mutex);

            pthread_exit(NULL);
```

```c
}

int main()
{
    pthread_t threads[2];
    int i = 0;

    pthread_mutex_init(&mutex, NULL);

    pthread_create(&threads[0], NULL, thread_fn_1, NULL);
    pthread_create(&threads[1], NULL, thread_fn_2, NULL);

    for (i = 0; i< NUM_THREADS; i++) {
        pthread_join(threads[i],NULL);
    }

    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond_var);

    printf("main: value is %d\n", value);

    return 0;
}
```

Exercises

1. Create a program that simulates the game of tic-tac-toe using two threads (one per player). The moves are randomly generated (obviously respecting the rules).

2. Create a program that simulates the Producer/Consumer problem using (1) an unbounded buffer and (2) a bounded buffer.

3. Create a program that simulates the following variations of the Readers/Writers problem:
    - Only one thread can do an operation on the critical section at a time;
    - No reader shall be kept waiting if the critical section is currently accessed for reading;
    - If there is a writer waiting, then it shall be kept waiting no longer than absolutely necessary;
    - No thread is allowed to starve.