Operating System Homework

Semaphores are used to provide better synchronization between processes.

In this homework you are asked to write a full program based on Reader Writer Lock example using Semaphores.

- Your program should:
  - Perform write and read operations on same data array
  - If two threads should access the same array
  - If any of these threads are writing data in the array, no other thread should be to read or write data
  - If there are no writes on the array, both threads should be able to read concurrently

- Submission should include:
  - Source code
  - Example screenshots of the program
  - All files should be zipped in one file

You can use the implementation of Reader Writer Lock with semaphores as shown below:

```
1  typedef struct _rwlock_t {
2    sem_t lock;      // binary semaphore (basic lock)
3    sem_t writelock; // used to allow ONE writer or MANY readers
4    int   readers;   // count of readers reading in critical section
5  } rwlock_t;
6
7  void rwlock_init(rwlock_t *rw) {
8    rw->readers = 0;
9    sem_init(&rw->lock, 0, 1);
10   sem_init(&rw->writelock, 0, 1);
11 }
12
13 void rwlock_acquire_readlock(rwlock_t *rw) {
14   sem_wait(&rw->lock);
15   rw->readers++;
16   if (rw->readers == 1)
17     sem_wait(&rw->writelock); // first reader acquires writelock
18   sem_post(&rw->lock);
19 }
20
21 void rwlock_release_readlock(rwlock_t *rw) {
22   sem_wait(&rw->lock);
23   rw->readers--;
24   if (rw->readers == 0)
25     sem_post(&rw->writelock); // last reader releases writelock
26   sem_post(&rw->lock);
27 }
28
29 void rwlock_acquire_writelock(rwlock_t *rw) {
30   sem_wait(&rw->writelock);
31 }
32
33 void rwlock_release_writelock(rwlock_t *rw) {
34   sem_post(&rw->writelock);
35 }
```