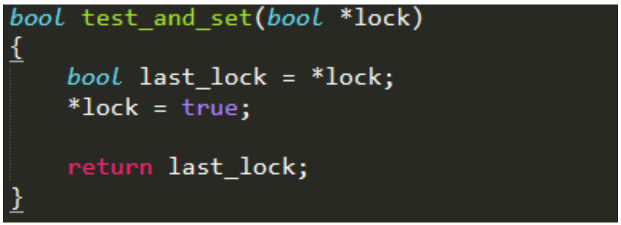
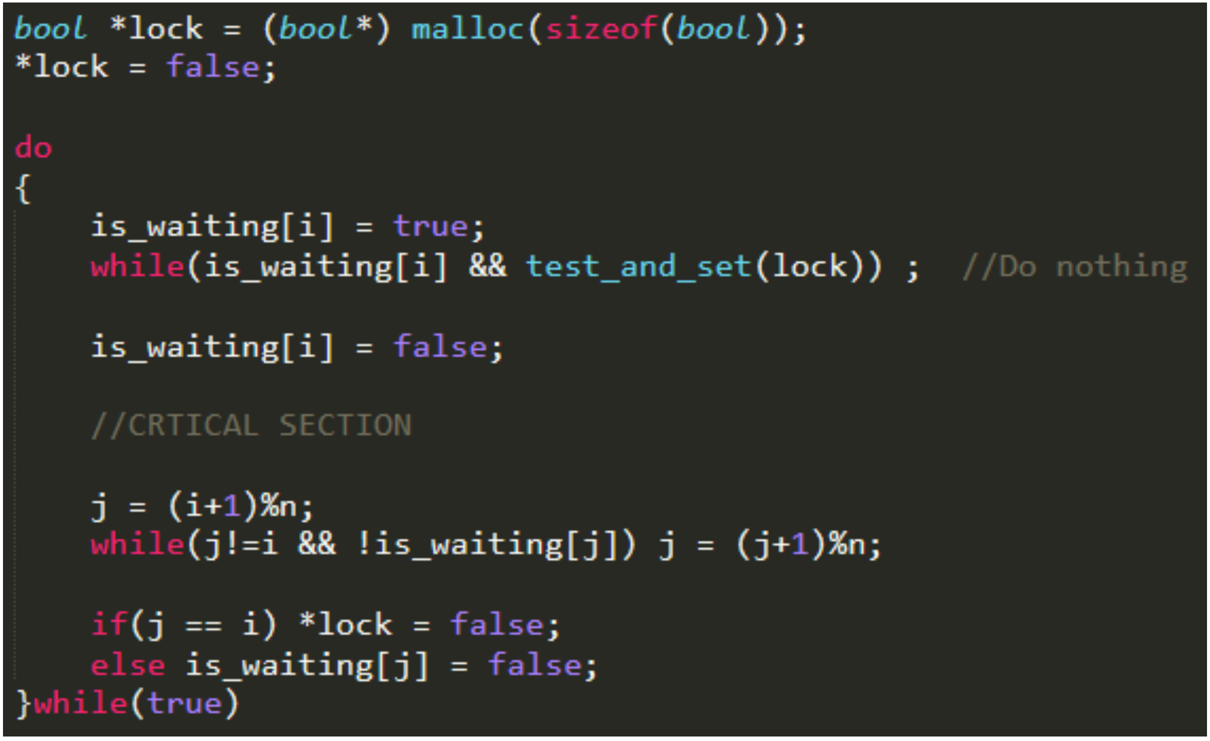
HandwrittenHomework1

1. Prove TestAndSet is correct. (30%)



```
bool test_and_set(bool *lock)
{
    bool last_lock = *lock;
    *lock = true;

    return last_lock;
}
```



```
bool *lock = (bool*) malloc(sizeof(bool));
*lock = false;

do
{
    is_waiting[i] = true;
    while(is_waiting[i] && test_and_set(lock)) ;  //Do nothing

    is_waiting[i] = false;

    //CRTICAL SECTION

    j = (i+1)%n;
    while(j!=i && !is_waiting[j]) j = (j+1)%n;

    if(j == i) *lock = false;
    else is_waiting[j] = false;
}while(true)
```

i. Mutual Exclusion

*lock is initialized to false. Because test_and_set is atomic hardware instruction, when first process calls test_and_set, it changes *lock to true and get return value false without being interrupted. This process can break while loop and enter critical section. After first process calls test_and_set, other process gets true from test_and_set and they also set there is_waiting to true, so they cannot enter critical section.

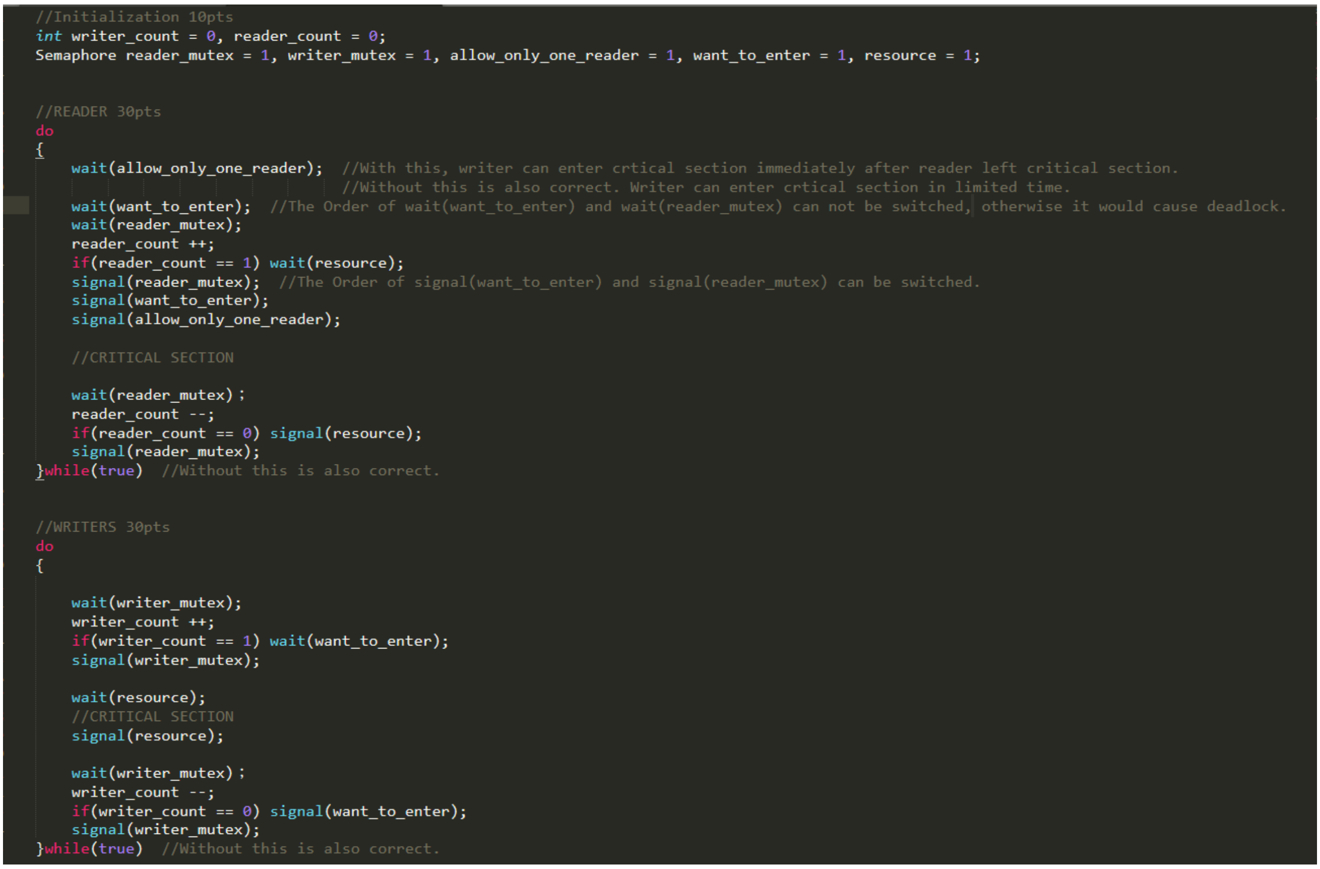
ii. Progress

After the process leaves critical section and finds no other process want to enter critical section at that time, it changes *lock to false. As the result, if there is no process in critical section, *lock is false. Next process which wants to enter critical section gets false from test_and_set, so it can enter critical section.

iii. Bounded waiting

If a process leaves critical section, it tests i+1 wants to enter critical section or not. If not, it increases the counter and test until the counter equals to itself. If it finds there is a process wants to enter critical section, it changes that process's is_waiting to false, so that process can enter critical section. As the result, after waiting no bigger than n+1 times, the process can enter critical section.

2. Second Readers and Writers Problem. (70%)



```
//Initialization 10pts
int writer_count = 0, reader_count = 0;
Semaphore reader_mutex = 1, writer_mutex = 1, allow_only_one_reader = 1, want_to_enter = 1, resource = 1;


//READER 30pts
do
{
    wait(allow_only_one_reader);  //With this, writer can enter crtical section immediately after reader left critical section.
                                  //Without this is also correct. Writer can enter crtical section in limited time.
    wait(want_to_enter);  //The Order of wait(want_to_enter) and wait(reader_mutex) can not be switched, otherwise it would cause deadlock.
    wait(reader_mutex);
    reader_count ++;
    if(reader_count == 1) wait(resource);
    signal(reader_mutex);  //The Order of signal(want_to_enter) and signal(reader_mutex) can be switched.
    signal(want_to_enter);
    signal(allow_only_one_reader);

    //CRITICAL SECTION

    wait(reader_mutex) ;
    reader_count --;
    if(reader_count == 0) signal(resource);
    signal(reader_mutex);
}while(true)  //Without this is also correct.


//WRITERS 30pts
do
{

    wait(writer_mutex);
    writer_count ++;
    if(writer_count == 1) wait(want_to_enter);
    signal(writer_mutex);

    wait(resource);
    //CRITICAL SECTION
    signal(resource);

    wait(writer_mutex) ;
    writer_count --;
    if(writer_count == 0) signal(want_to_enter);
    signal(writer_mutex);
}while(true)  //Without this is also correct.
```