Programming Assignment 3

ReadMe

Implement a solution via semaphores and threads to the n reader 1 writer problem. Fairness always matters. You will accept the number of readers from the command line. In no case will more than 14 readers be used and always at least 1 reader will be used. Each reader must access a shared counter value 250000000 times in the critical section. Note, it does not update anything, just "reads". For convenience code is below that will do this in the function `relaxandspendtime`.
A reader reads just one time and a writer writes just one time. Each reader needs to print its name when done. The writer will update the value 25000 times and print done. The writer will also set a shared flag, in-cs, when it enters the critical section and reset it just before it leaves the critical section. The reader must, upon entering the critical section, check this flag and write an error message if the flag is set.

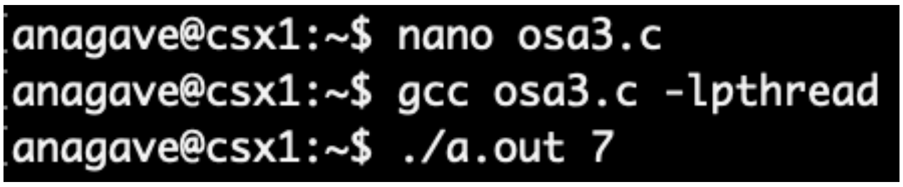
Source Code:

```
//Name: Achyutha Santhoshi
//CWID: A20314248

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

sem_t wrt;//writer's semaphore
pthread_mutex_t mutex;
pthread_mutex_t flag;//to ensure that there is only one thread in cs
int cnt = 0;//Shared variable
int numreader = 0;
int writercount = 0;


void relaxandspendtime() //reader thread's no return function
{
    int i;
    for(i = 0; i < 250000000; i++) i=i;
}
void *writer_thread(void *p)
{
    //both semaphore and flag are locked after entering in here
    pthread_mutex_lock(&flag);
    sem_wait(&wrt);
    for(int i=0;i<25000;i++)
    {
       //writer thread performing its operations
       writercount = 1;
       cnt = cnt + 1;
       writercount = 0;
    }
    printf("Writer %d: done, updated readcount to %d\n",(*((int *)p)),cnt);
    sem_post(&wrt);//Semaphore and flag are unlocked here
    pthread_mutex_unlock(&flag);


}
void *reader_thread(void *p)
{
    //checks whether flag is unlocked or not
     while (pthread_mutex_trylock(&flag) != 0){
       }
       pthread_mutex_unlock(&flag);
    //mutex lock
    pthread_mutex_lock(&mutex);
    numreader++;
    if(numreader == 1) {
       sem_wait(&wrt);
    }
    pthread_mutex_unlock(&mutex);//mutex unlock


    relaxandspendtime();
    printf("Reader %d: done, read readcount as %d\n",*((int *)p),cnt);
    //pops up errors message when writer thread is still in CS
    if(writercount == 1)
    {
       printf("ERROR: Writer is writing\n");
    }

    pthread_mutex_lock(&mutex);//mutex lock
    numreader--;
    if(numreader == 0) {
       sem_post(&wrt);//releases semaphore
    }
    pthread_mutex_unlock(&mutex);//releases mutex lock
}
```

```
int main(int argc, char **argv)
{
   int no_of_readers = atoi(argv[1]);//takes command line argument as
number of readers input
   int i = 0, j = 0, k = 0;
   pthread_t readers[no_of_readers];//initializes reader threads
   pthread_t writer[1];//initializes writer threads
   pthread_mutex_init(&mutex, NULL);//Mutex initialization
   sem_init(&wrt,0,1);//Semaphore initialization

   //printf("%d\n",no_of_readers);
   int p[15] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};//to avoid cast warning


   //checks the edge case whether there no more than 14 readers and atleast
more than 1 reader
   if (!(no_of_readers>1 && no_of_readers<15))
   {
      printf("In no case will more than 14 readers be used and always at least
1 reader will be used");
      exit(0);
   }


   //As given in the Assignment problem statement
   k = (int) (no_of_readers/2);

   for(i = 0; i < k; i++)
   {
      pthread_create(&readers[i],  NULL, (void *)reader_thread, (void *)&p[i]); //
creating half of the reader threads
   }
   pthread_create(&writer[0], NULL, (void *)writer_thread, (void *)&p[0]);  //
creating writer thread
   for(i = k ; i < no_of_readers ; i++)
   {
      pthread_create(&readers[i],  NULL, (void *)reader_thread, (void *)&p[i]); //
creating rest reader threads
   }


   //joining all the threads
   for(i = 0; i < no_of_readers; i++){
      pthread_join(readers[i], NULL);
```

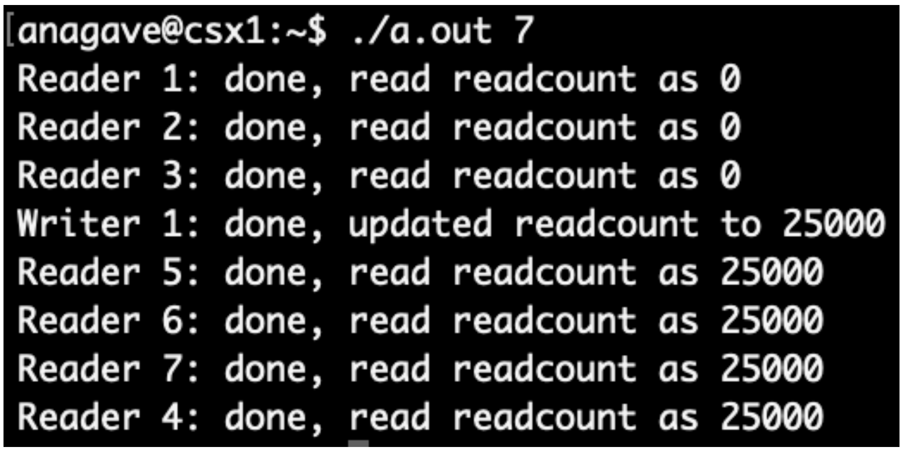
```
    }
    pthread_join(writer[0],NULL);

    //using destroy to free memory space
    pthread_mutex_destroy(&mutex);
    sem_destroy(&wrt);

    return 0;
}
```

Execution:



```
anagave@csx1:~$ nano osa3.c
anagave@csx1:~$ gcc osa3.c -lpthread
anagave@csx1:~$ ./a.out 7
```

Output:



```
anagave@csx1:~$ ./a.out 7
Reader 1: done, read readcount as 0
Reader 2: done, read readcount as 0
Reader 3: done, read readcount as 0
Writer 1: done, updated readcount to 25000
Reader 5: done, read readcount as 25000
Reader 6: done, read readcount as 25000
Reader 7: done, read readcount as 25000
Reader 4: done, read readcount as 25000
```

Steps to Compile and Execute:

Step 1: Created a new file using nano with .c extension
Step 2: Programmed with all requirements(Explained in Next Section)
Step 3: 3.1 Using Command gcc osa3.c -lpthread to compile the program
3.2 Using Command ./a.out 7 to execute and pass 7 as command line argument, Here my program is accepting any number between 1 and 14, if not it exits with an error message
Step 4: Displays the output showing how readers and writers are executing, printed done with the reader or writer number with shared memory variable value.

Explanation of Source Code:

- In main program, I have simply initialised all the required semaphores, mutexes, reader and writer threads.

- It takes a command line argument of integer variable no more than 14 and no less than 1, which is used to determine number of readers.
- Then, I am creating reader, writer threads as mentioned in the Assignment problem statement and joining them.
- I have created two different functions for Reader_thread and Writer_thread.
- In Reader_thread, Checks the flag and continues its operation by acquiring and releasing locks
- And a function for reader thread, relaxandspendtime.
- In Writer_thread, locks flag and continues its operation by acquiring and releasing locks, releasing flag.
- The reader writer functions are in the structure which is explained in Report.