

**PROJECT REPORT.**

**READER WRITER PROBLEM:**

**GROUP MEMBERS:**

Hunain Arif (19k-0299)

Khizer Riaz (19k-1296)

Muhammad Hassan (19k-0238)

**PROJECT DETAILS:**

* GROUP SECTION: F
* LANGUAGE: C
* OPERATING SYSTEM: Ubuntu 16.04
* COURSE INSTRUCTOR: MISS TANIA IRAM / MISS ANAUM HAMID

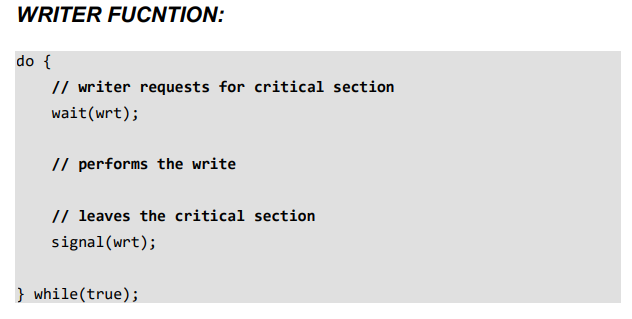
**OBJECTIVE:**

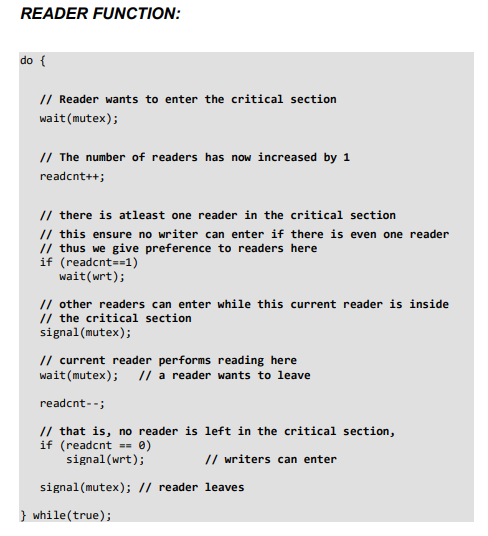
In the readers-writers problem, there is a critical section that both reader and writer can access. Reader only reads from the data while writer can both read and write to the data. More than one reader can read at the same time. A writer cannot access the data while a reader is reading. No other thread can access the memory while a writer is accessing the data.

**PROPOSED SOLUTION:**

To solve this situation, a writer should get exclusive access to an object that is: If one of the people tries editing the file, no other person should be reading or writing at the same time, otherwise changes will not be visible to him/her. However if some person is reading the file, then others may read it at the same time. From the above problem statement, it is evident that readers have higher priority than writer. If a writer wants to write to the resource, it must wait until there are no readers currently accessing that resource. Here priority means, no reader should wait if the share is currently opened for reading. Some problem parameters to acknowledge include: · One set of data is shared among a number of processes · Once a writer is ready, it performs its write. Only one writer may write at a time · If a process is writing, no other process can read it .If at least one reader is reading, no other process can write · Readers may not write and only read

**ALGORITHM USED:**





**GOAL:**

* The goal of this assignment is to implement a multithreaded solution for the above mentioned algorithm using Semaphores. Moreover, steps are also taken to calculate the average time of between a request and an entry into the critical section by the reader or writer

**APPROACH USED:**

1. Make a system call that simulates the readers/writers problem with a hard coded number of readers and writers.
2. 2. Use of average time as a parameterized system call which calculates and prints the average time between a request and an entry into the critical section listed above on kernel space.

**CALCULATING TIME:**

* Request Time: This is the time when the reader / writer as requested for getting access to the critical section i.e. to perform their operation.
* Entry Time: This is the time when the reader / writer achieves the permission and starts their work of reading / writing.
* Exit Time: This is the time when the reader / writer finished their work and decides to leave.

***LIBRARIES USED:***

* **#include <semaphore.h>**
* **#include <stdio.h>**
* **#include <time.h>**
* **#include <stdint.h>**
* **#include <stdlib.h>**
* **#include <unistd.h>**
* **#include <pthread.h>**

**Reader Function():**

* 1. Reader requests the entry to critical section.
* 2. If allowed: it increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the wrt semaphore to restrict the entry of writers if any reader is inside.
* It then, signals mutex as any other reader is allowed to enter while others are already reading.
* After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore mutex as now, writer can enter the critical section.
* 3. If not allowed, it keeps on waiting.
* 4. time.hused in order to calculate times between request and entries. Thus, the semaphore mutex is queued on both readers and writers in a manner such that preference is given to readers if writers are also there. Thus, no reader is waiting simply because a writer has requested to enter the critical section.

**Writer Function():**

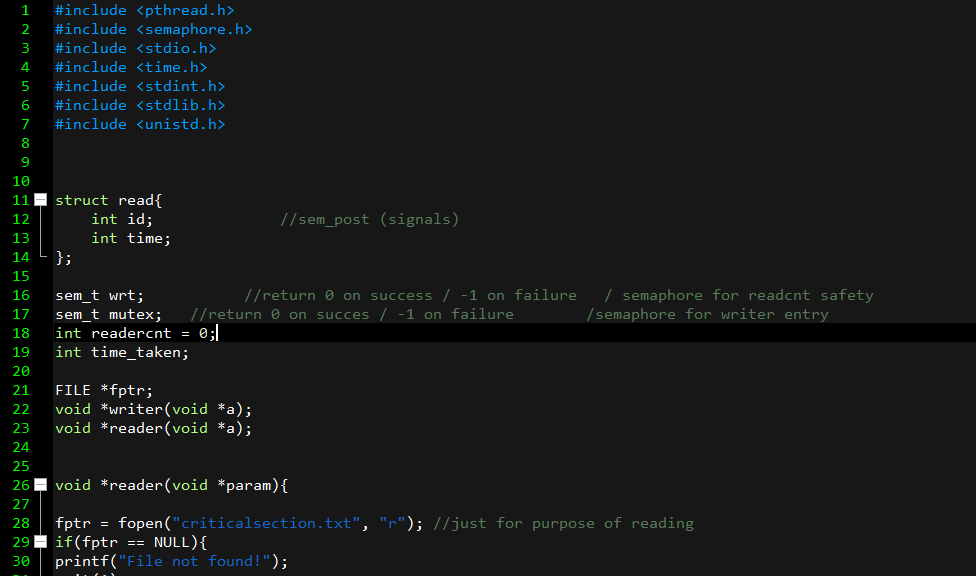
1. Writer requests the entry to critical section.

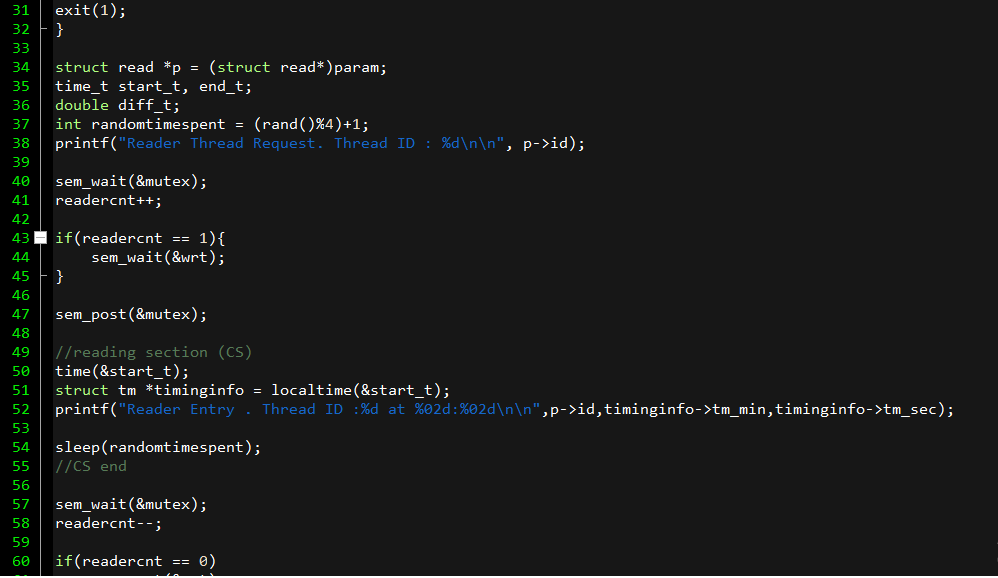
2. If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.

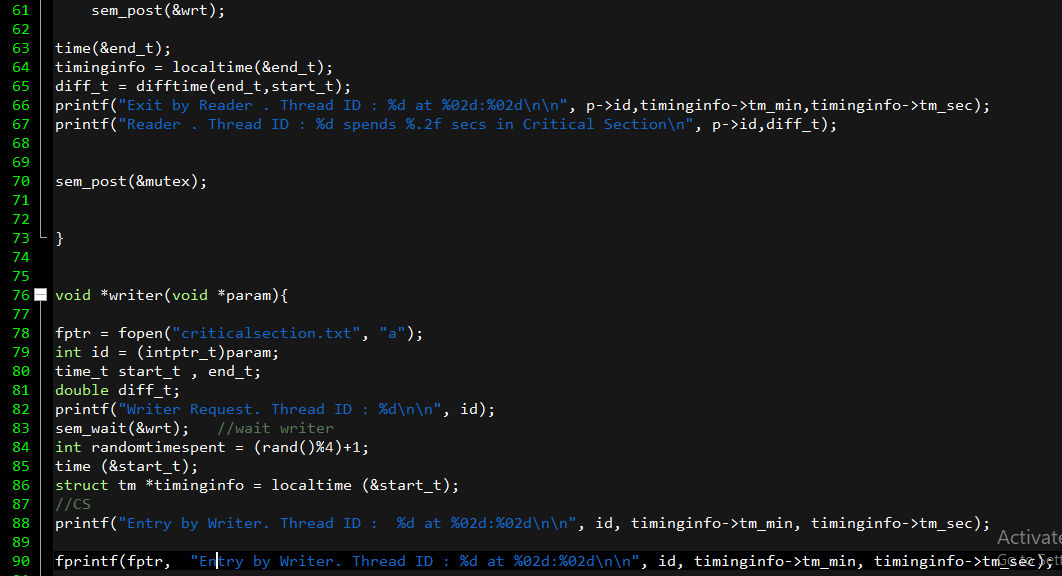
3. It exits the critical section.

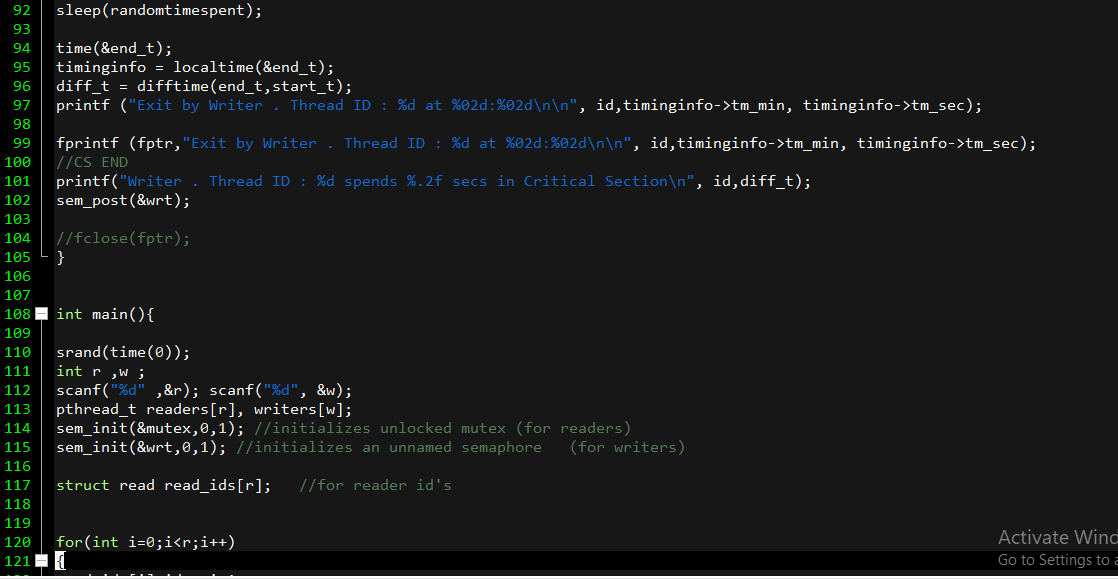
4. time.h used in order to calculate times between request and entries

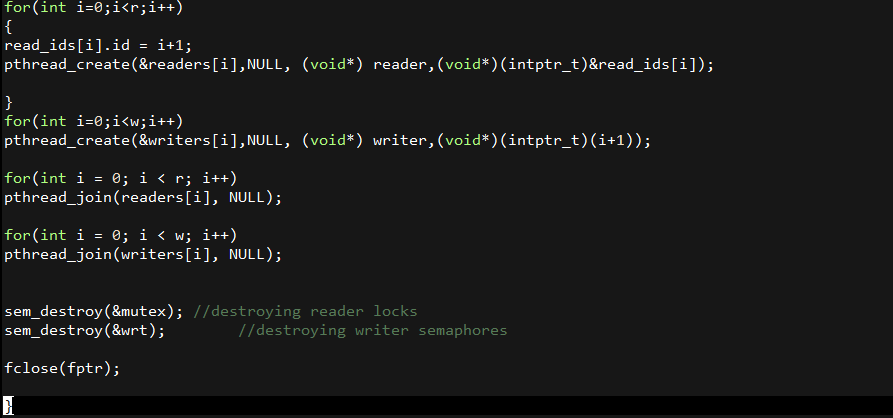
**C-CODE:**

****

****

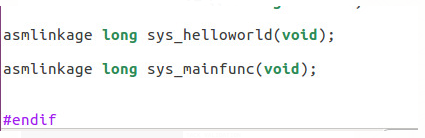
****

****

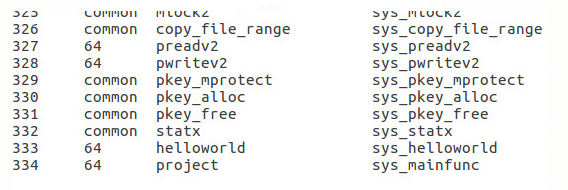
****

**KERNEL SCREEN SHOTS:**

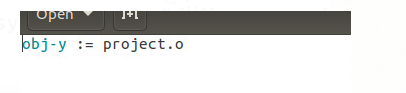
**LINUX MAKE-FILE:**



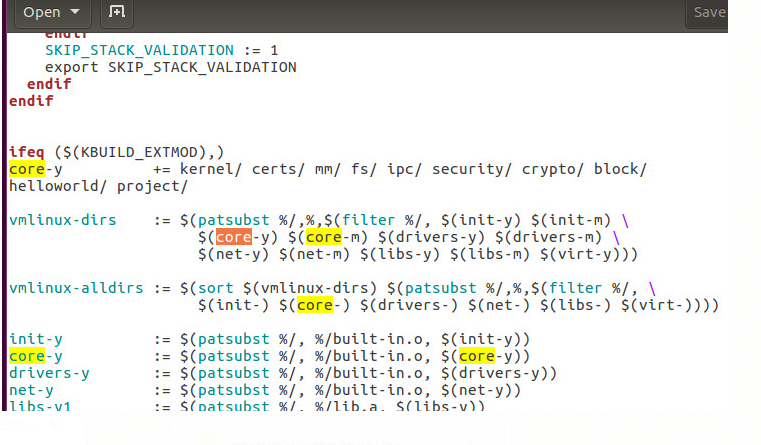
**TABLE ENTRY:**



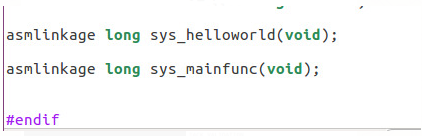
**PROJECT KERNEL CODE MAKE FILE:**

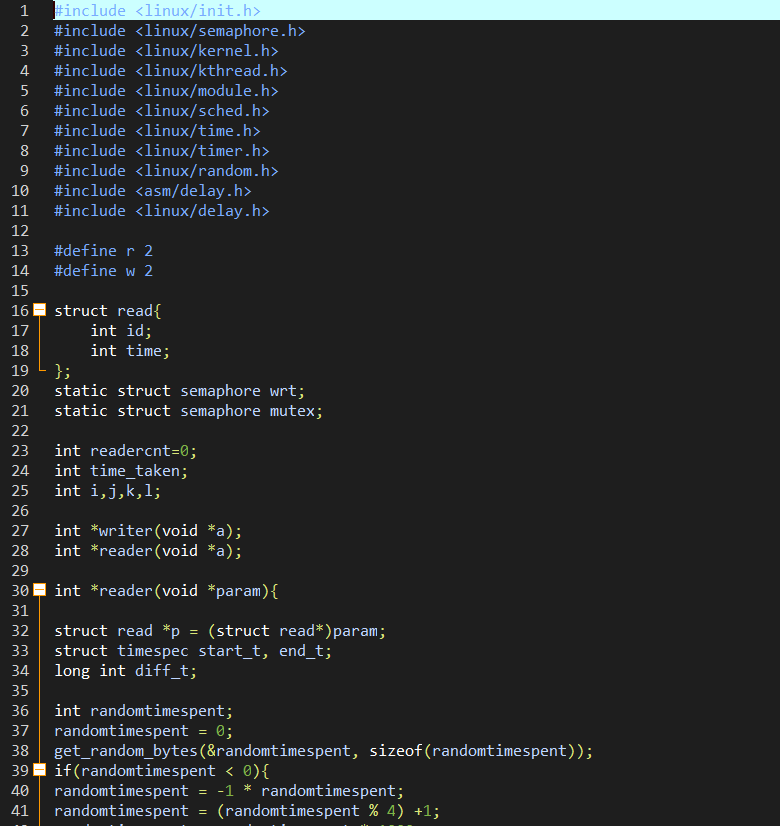


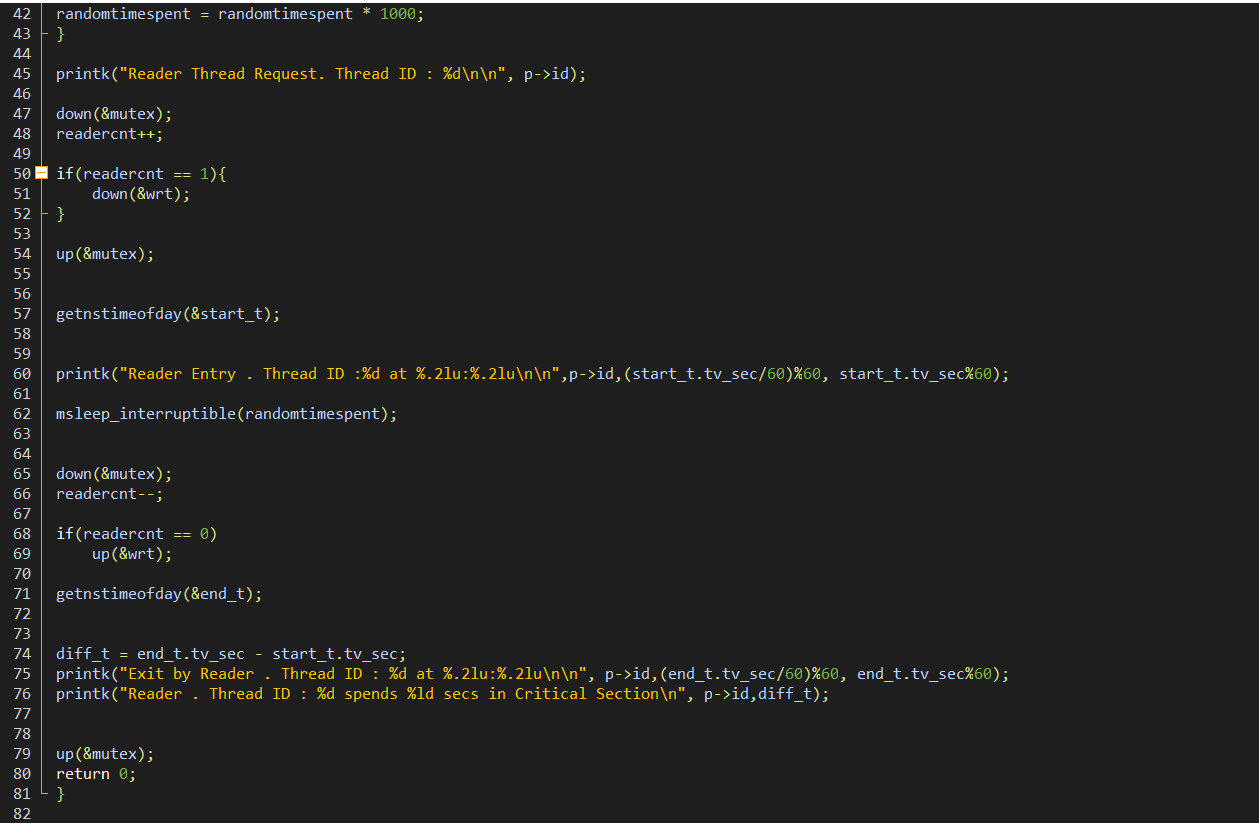
**MAKEFILE KERNEL:**

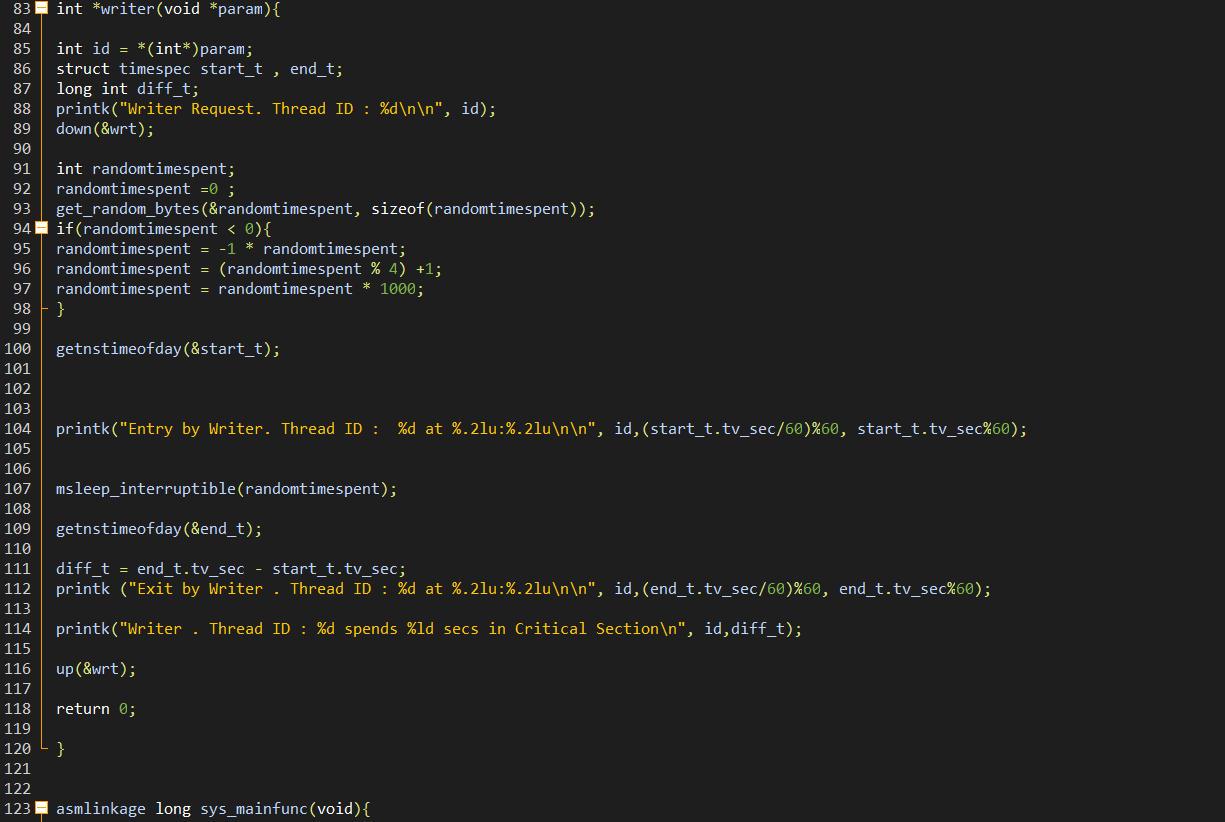


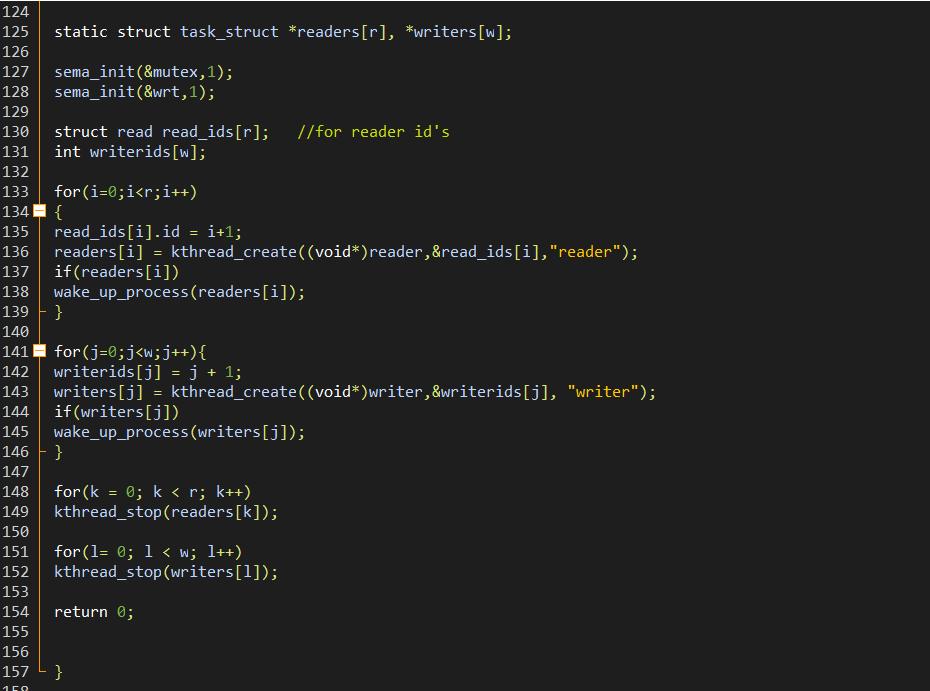
**SYSCALLS.H:**



**SYSTEM CALL.C**







**TEST.C**

