**OPERATING SYSTEM LAB 9**

|  |  |
| --- | --- |
| Roll No.: K041 | Name: Anish Sudhan Nair |
| Batch No.: A2/K2 | Date: 21/02/2022 |

Aim: To research the readers writers problem

Apparatus: Atom IDE, clang compiler, ZSH terminal, <http://uos-simulator.herokuapp.com/process_synchronization/reader-writer.html>

Theory:

The readers-writers problem is deals with process synchronization wherein a data set such as a file is shared between multiple processes at the same time. In this problem particularly, we have readers and writers as processes. Readers can only read the data while writers can write data as well.

The problem aids in managing synchronization among various reader and writer process so that there are no problems of data inconsistency.

For example, if four readers and a writer want to access a data set, when the writer is accessing it, the other processes cant while if a reader is accessing it, other readers can access it but a writer cant. In other case, there would be generation of inconsistent data since the changes implemented by the writer wont be reflected to the readers.

To enable this synchronisation, we use two semaphores (write: deals with writing permission and mutex: denotes the mutual exclusion of processes for reader processes). We use the atomic operations, wait and signal, to decrement and increment the values depending on the readerCount (which denotes the number of reader processes accessing the data) and after writing operations.

Case 1:



The initial value of semaphore write is 1. When the writer process enters, wait(write) decrements the value of write to 0 thereby denying permission to any other writer process to enter the database since that would be stuck in an infinite loop until the first process is done writing and the write value is incremented.

Case 2:



After the writer is done writing, it leaves the database and signal(write) increments the value of write to 1, thereby allowing other writer process to enter the database.

Case 3:



When a reader process arrives, the value of mutex which is 1 initially, decrements due to wait(mutex). Additionally, the readerCount increments by 1. Since readerCount is equal to 1, wait(write) will decrement the value of write to 0, denying any writer process to enter the database. After checking for writer process, signal(mutex) increments mutex to 1, thereby allowing more reader process to enter the database.

Case 4:



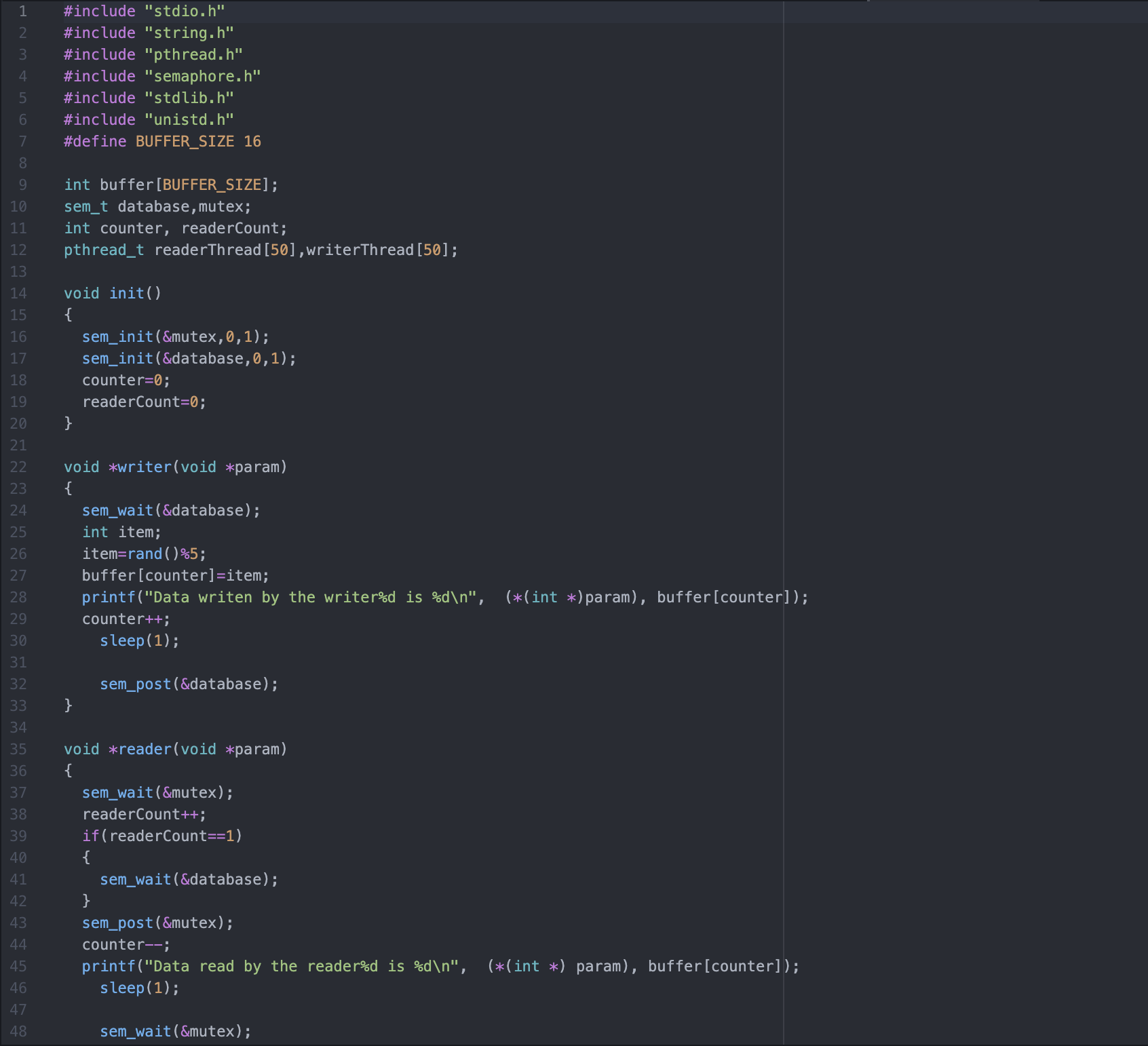
After mutex is incremented to 1, another reader process enters. Wait(mutex) decrements mutex again, followed by incrementing the readerCount which is now 2. Since readerCount is 2, the if block is not executed for checking for writer process and signal(mutex) increments mutex to 1 allowing more reader processes.

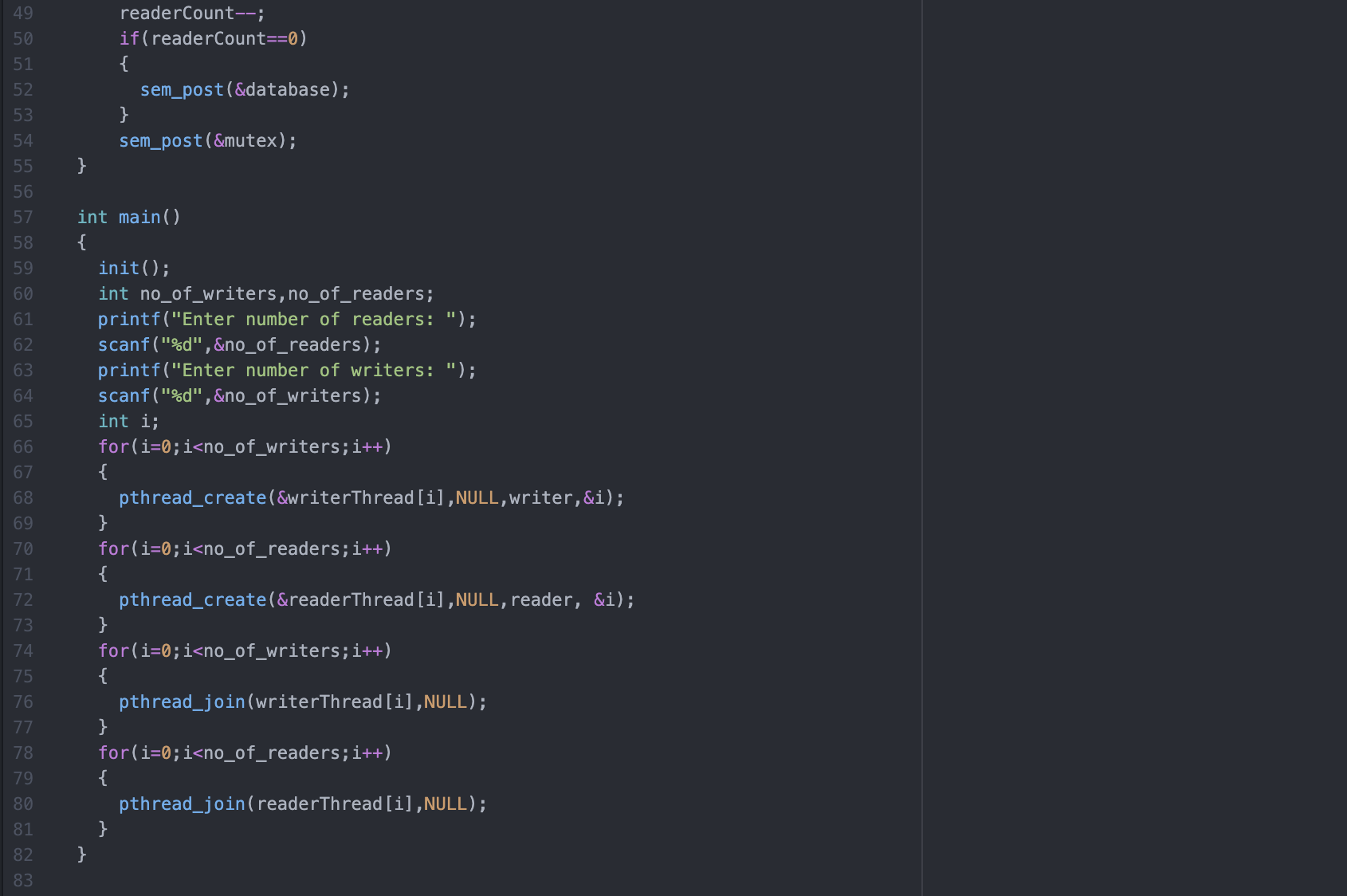
Case 5:



When reader process (3 here) leaves the database, wait(mutex) decrements mutex to 0 denying any process to enter, followed by decrement of readerCount. Since the readerCount is now 1 and not 0, the of block for signal(write) is not executed thereby continuing to deny any writers to enter the database. After the if block is skipped, signal(mutex) increments mutex to 1 allowing for other reader processes to enter the database.

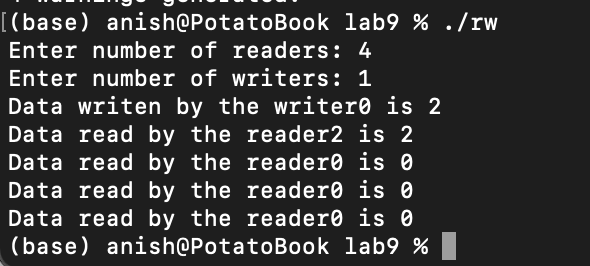
Code:





Output:

4 readers and 1 writer



CONCLUSION:

In this lab, we were to implement and demonstrate the readers-writers problem. By explaining the individual cases and executing the actual code, it helped to reinforce the working of this process synchronisation and the manner in which it’s solved.