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:

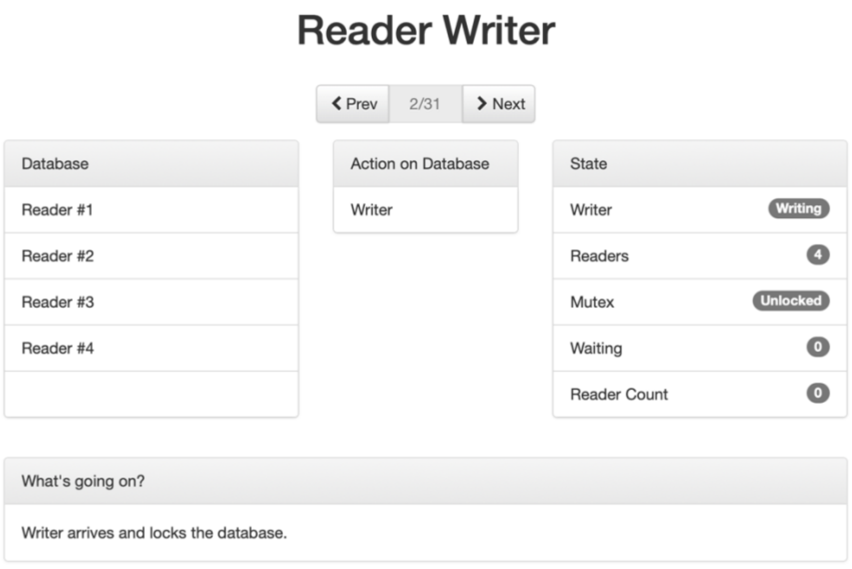
The readers-writers problem is deals with process synchronization wherein a data set such as a file is shared between multiple processes at the same time. In this problem particularly, we have readers and writers as processes. Readers can only read the data while writers can write data as well.

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:



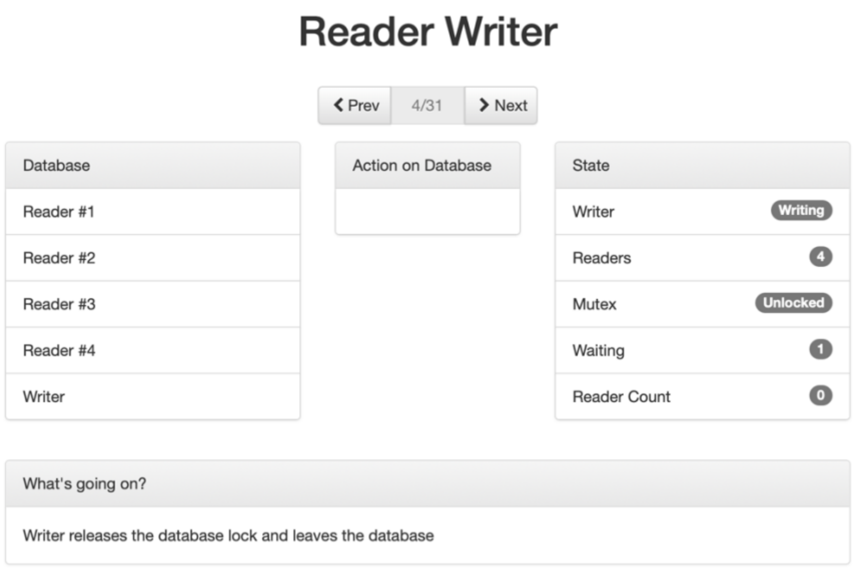
Database	Action on Database	State	
Reader #1	Writer	Writer	Writing
Reader #2		Readers	4
Reader #3		Mutex	Unlocked
Reader #4		Waiting	0
		Reader Count	0

What's going on?

Writer arrives and locks the database.

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:



Database
Reader #1
Reader #2
Reader #3
Reader #4
Writer

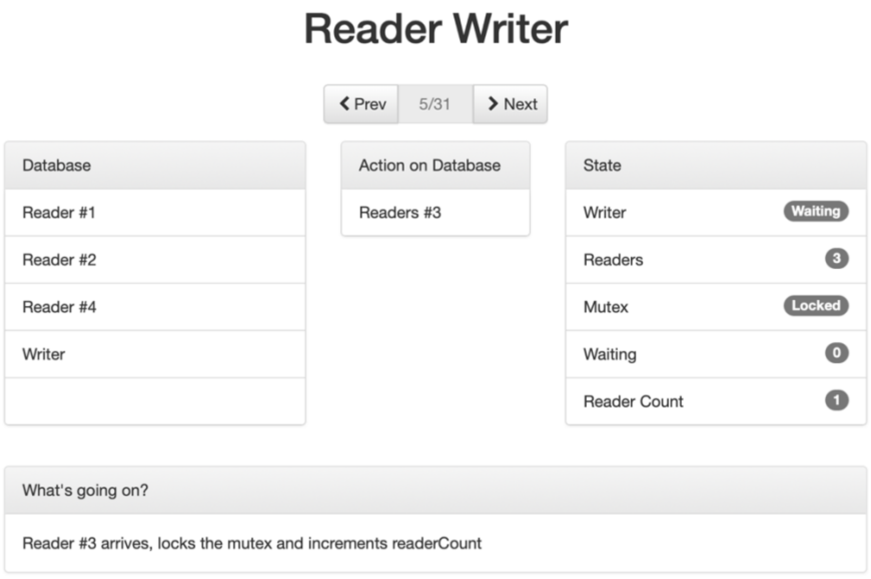
Action on Database

State	
Writer	Writing
Readers	4
Mutex	Unlocked
Waiting	1
Reader Count	0

What's going on?
Writer releases the database lock and leaves the database

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:



Database
Reader #1
Reader #2
Reader #4
Writer

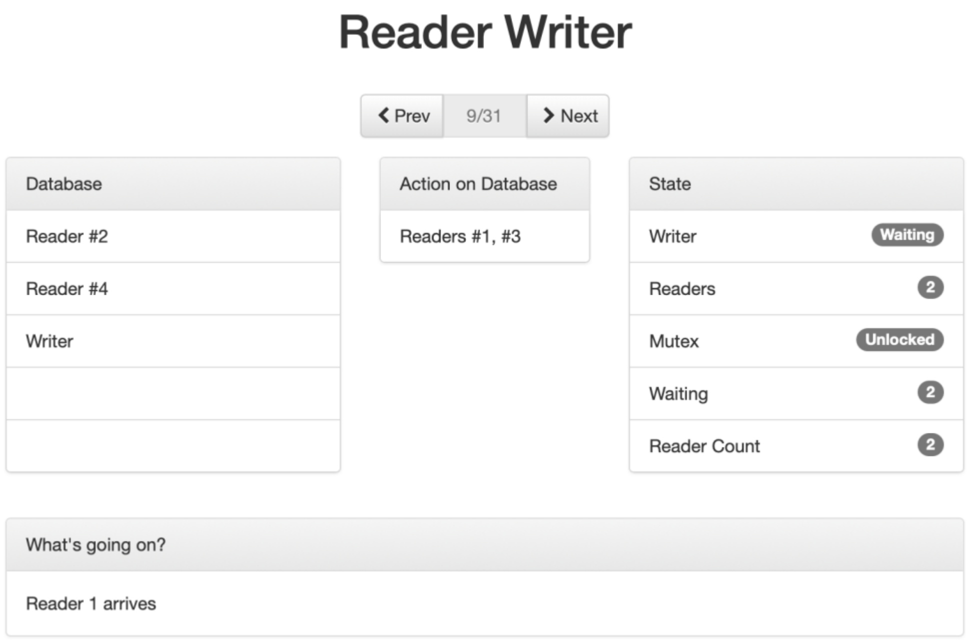
Action on Database
Readers #3

State	
Writer	Waiting
Readers	3
Mutex	Locked
Waiting	0
Reader Count	1

What's going on?
Reader #3 arrives, locks the mutex and increments readerCount

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:



Reader Writer

< Prev 9/31 > Next

Database
Reader #2
Reader #4
Writer

Action on Database
Readers #1, #3

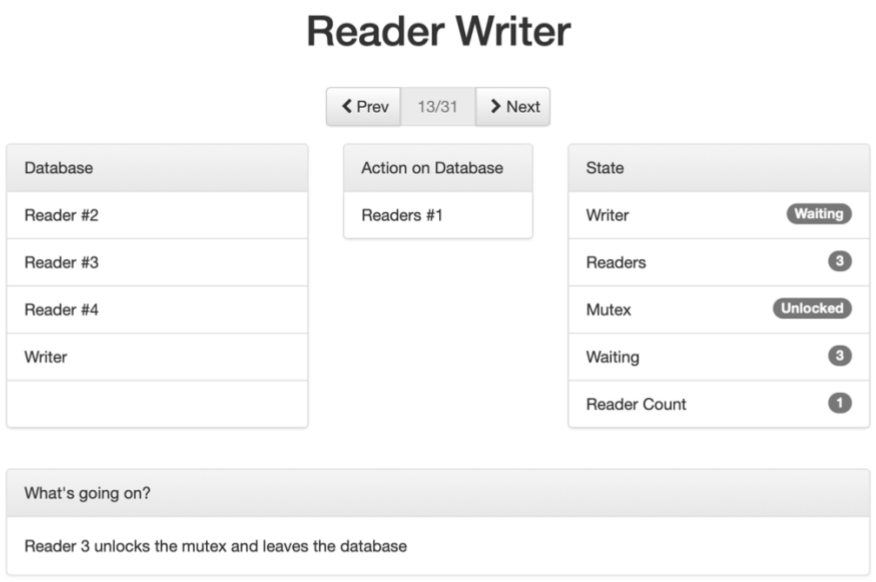
State	
Writer	Waiting
Readers	2
Mutex	Unlocked
Waiting	2
Reader Count	2

What's going on?

Reader 1 arrives

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:



Reader Writer

< Prev 13/31 > Next

Database
Reader #2
Reader #3
Reader #4
Writer

Action on Database
Readers #1

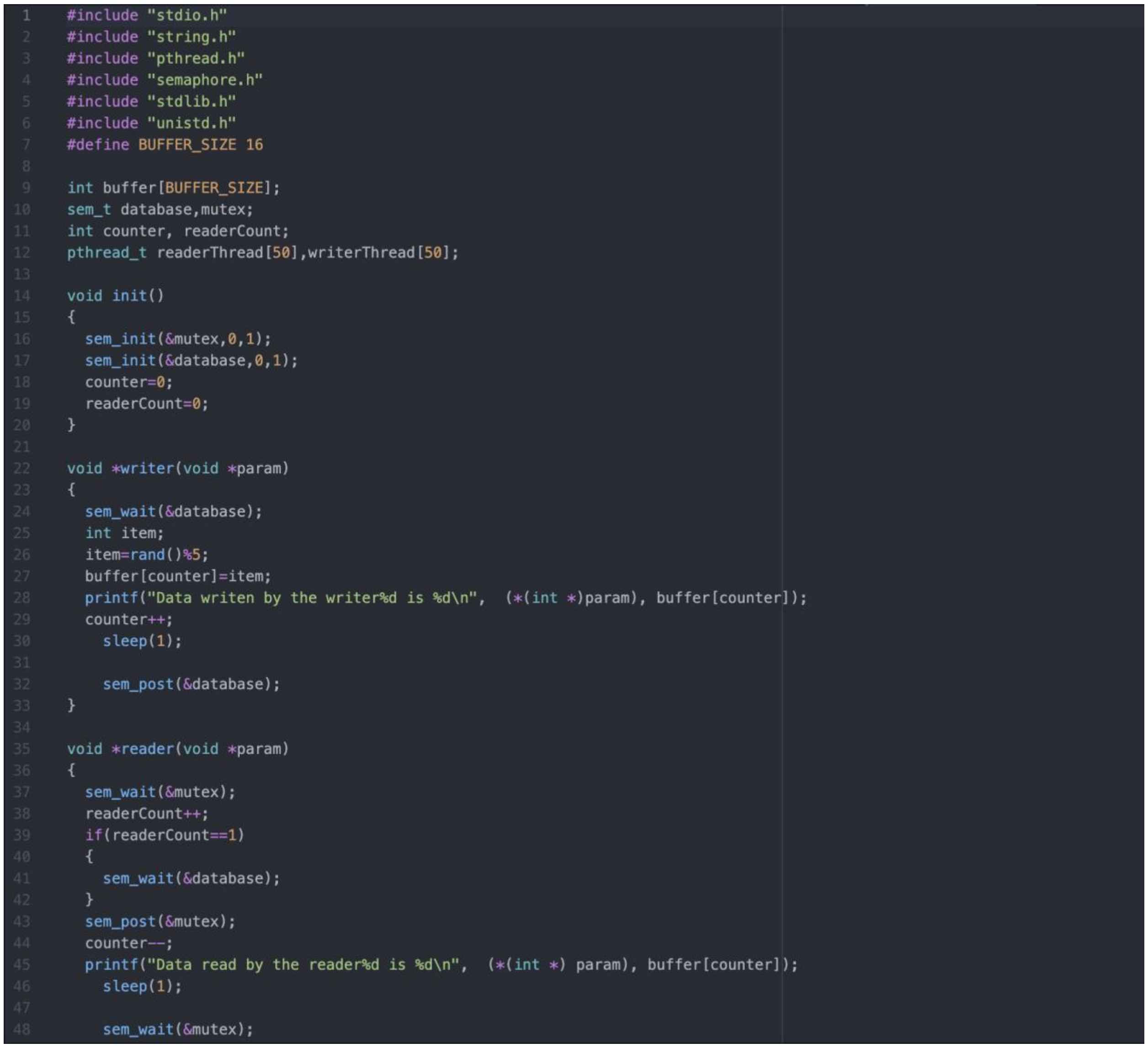
State	
Writer	Waiting
Readers	3
Mutex	Unlocked
Waiting	3
Reader Count	1

What's going on?

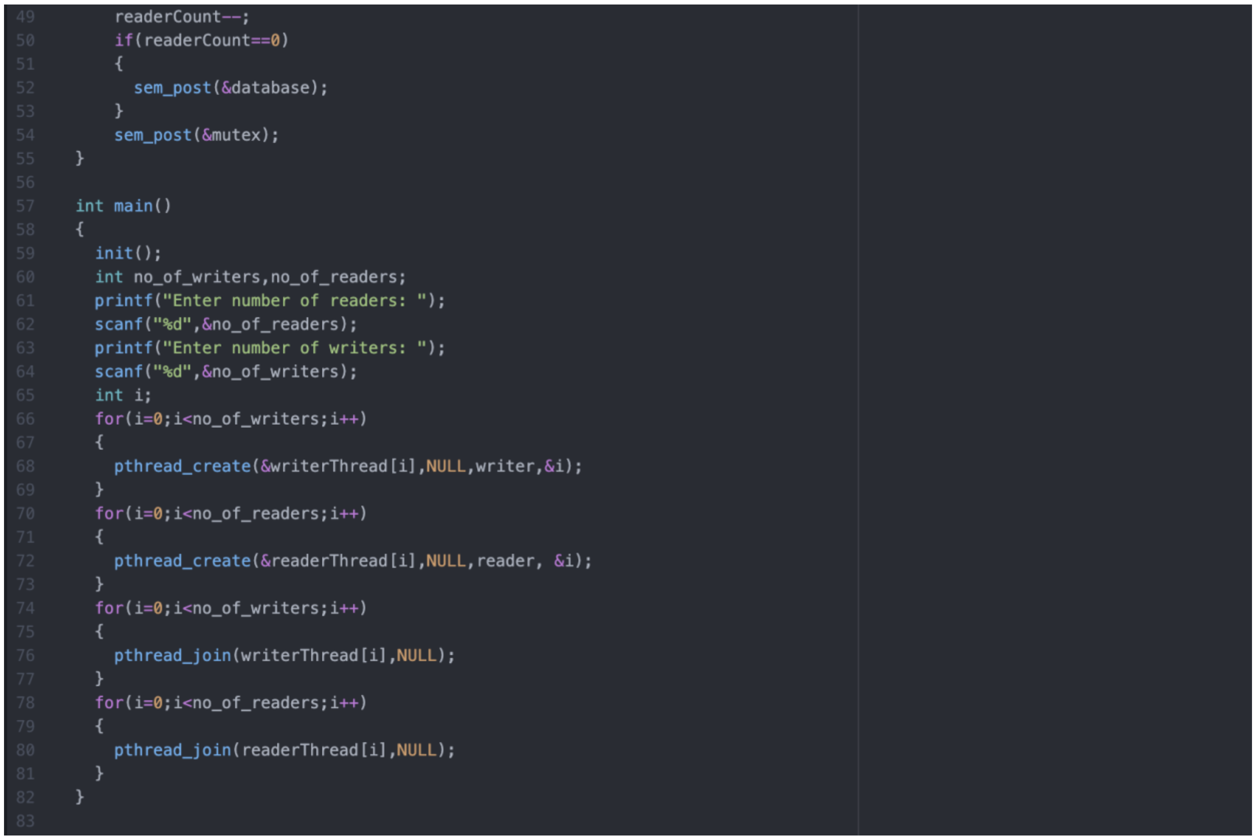
Reader 3 unlocks the mutex and leaves the database

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:



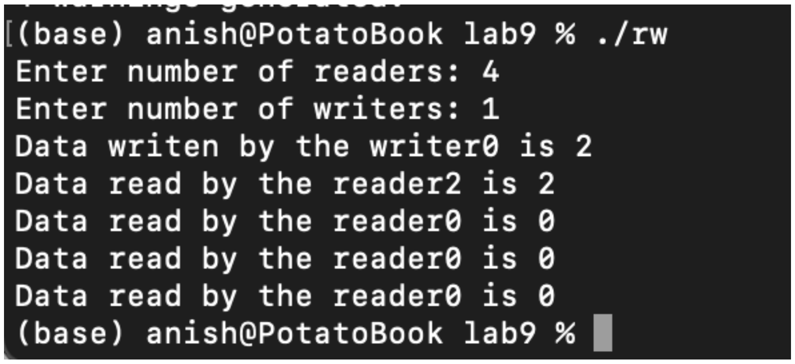
```
#include "stdio.h"
#include "string.h"
#include "pthread.h"
#include "semaphore.h"
#include "stdlib.h"
#include "unistd.h"
#define BUFFER_SIZE 16

int buffer[BUFFER_SIZE];
sem_t database,mutex;
int counter, readerCount;
pthread_t readerThread[50],writerThread[50];

void init()
{
  sem_init(&mutex,0,1);
  sem_init(&database,0,1);
  counter=0;
  readerCount=0;
}

void *writer(void *param)
{
  sem_wait(&database);
  int item;
  item=rand()%5;
  buffer[counter]=item;
  printf("Data writen by the writer%d is %d\n",  (*(int *)param), buffer[counter]);
  counter++;
    sleep(1);

    sem_post(&database);
}

void *reader(void *param)
{
  sem_wait(&mutex);
  readerCount++;
  if(readerCount==1)
  {
    sem_wait(&database);
  }
  sem_post(&mutex);
  counter--;
  printf("Data read by the reader%d is %d\n",  (*(int *) param), buffer[counter]);
    sleep(1);

    sem_wait(&mutex);
```



```c
    readerCount--;
    if(readerCount==0)
    {
      sem_post(&database);
    }
    sem_post(&mutex);
}

int main()
{
  init();
  int no_of_writers,no_of_readers;
  printf("Enter number of readers: ");
  scanf("%d",&no_of_readers);
  printf("Enter number of writers: ");
  scanf("%d",&no_of_writers);
  int i;
  for(i=0;i<no_of_writers;i++)
  {
    pthread_create(&writerThread[i],NULL,writer,&i);
  }
  for(i=0;i<no_of_readers;i++)
  {
    pthread_create(&readerThread[i],NULL,reader, &i);
  }
  for(i=0;i<no_of_writers;i++)
  {
    pthread_join(writerThread[i],NULL);
  }
  for(i=0;i<no_of_readers;i++)
  {
    pthread_join(readerThread[i],NULL);
  }
}
```

Output:

4 readers and 1 writer



```
[(base) anish@PotatoBook lab9 % ./rw
Enter number of readers: 4
Enter number of writers: 1
Data writen by the writer0 is 2
Data read by the reader2 is 2
Data read by the reader0 is 0
Data read by the reader0 is 0
Data read by the reader0 is 0
(base) anish@PotatoBook lab9 %
```

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.