Assignment No.: B10

Roll No.

- **Title :**
  To implement Reader Writer problem in Openmp

- **Problem Definition :**

  Implement a the Reader Writer problem using openmp

- **Learning Objective :**

  To study reader writer problem using openmp.

- **Learning Outcome :**

  Successfully implemented reader writer using Openmp.

- **Software and Hardware Requirement:**

- Openmp enabled machine
- Gedit and terminal

Theory :

1. Introduction to Reader Writer

A. What is Openmp?

OpenMP (Open Multi-Processing) is an API that supports multi- platform shared memory multiprocessing programming in C, C++, and Fortran, on most processor architectures and operating systems, including Solaris, AIX, HP-UX, Linux, Mac OS X, and Windows platforms.
A data set is shared among a number of concurrent threads. Readers-only read the dataset; they do not perform any updates. Writers-can perform both read and write operation.Openmp problem is to allow multiple readers to access the shared resource at a time and only one writer to access the shared resources.To implement this we use the concept of mutex and semaphore in openmp which performs the task of locking and unlocking the resources parallely.We can set the priorities to the reader and writer tasks.

Semaphore is a variable or abstract data type that is used for controlling access by multiple threads to a common resource in a parallel or multi-user environment.Semaphores which allow arbitrary resource count is called counting semaphore.Counting semaphores are equipped with two operations that is V(signal) and P(wait).The signal increments the semaphore value while the wait decrements the value of the semaphore.

To avoid starvation the semaphore has an associated queue of processes.If a process performs P operation on the semaphore that has value zero,the process is added to semaphore queue aand the execution is suspended.When another process increments the semaphore value by performing the V operation,and there are processes on the queue,one of them is removed from the queue and its execution is proceeded.If the processes have the priorities than the one with highest proirity is executed first and then the rest of the processes.
In reader writer problem one thread generates the data items and he other threads readds the generated data items.They communicate using queue having size N and are subjected to following condition:The reader must

wait for the writer to write the data if the queue is empty.The writer must wait for the readers to read the data if the queue is full.The semaphore solution to the reader writer problem in openmp is to keep the track of the state of the queue with two semaphores.emptycount to check number of empty places in the queue and ReadWritecount to check number of elements in the queue.To maintain integrity the value of emptycount should be less than the number of empty places in the queue and the ReadWritecount may be lower than the actual number of items placed in the queue.

2. Concept of programming language use

Shared Memory

- Shared Memory is an efficient means of passing data between programs. Shared memory is a feature supported by UNIX System V, including Linux, SunOS and Solaris.
- One program will create a memory portion which other processes (if permitted) can access.
- A shared segment can be attached multiple times by the same process.
- A shared memory segment is described by a control structure with a unique integer the Shared Memory ID that points to an area of physical memory. A shared memory is an extra piece of memory that is attached to some address spaces for their owners to use.
- One process must explicitly ask for an area, using a key, to be shared by other processes. This process will be called the server.
- All other processes, the clients, that know the shared area can access it.
- To implement a simple lock we have used the omp init lock function and the syntax is: void omp init lock($omplockt * lock$);
- We have used used omp function to set or unset the lock provided on the resources and the syntax for it is:
- void omp set lock($omplockt * lock$);
- void omp unset lock($omplockt * lock$);
- To destroy the lock initailized we use the following omp syntax:
- void omp destroy lock($omplockt * lock$);

3. Algorithm:

1. Include all the header files required for the program
2. Initialize the variable i,NumberofReaderThreads and NumberofWriterThread and readcount to 0
3. Initialize the omp lock variable i.e writelock
4. Initialize the lock by using omp init lock function
5. Accept the number of reader and writer threads from the user
6. Call the omp pragma parallel function
7. Inside pragma omp for in for($i = 0; i < NumberofReaderThread; i++$)
8. Initialize time variable and structure tm for timeinfo
9. Print the localtime and call omp set lock function
10. Increment readcount and if equal to 1 print reader is reading
11. Else call omp unset lock function and print reader is leaving if value of readcount is 0.
12. Call pragma omp parallel shared(tid)
13. In pragma omp for nowait in for(i=0;i¡NumberofReaderThread;i++);

14. call omp set lock function and then print writer is writing
15. call omp unset lock function and then print writer is leaving
16. Then call the omp destroy lock function

4. Flowcharts:

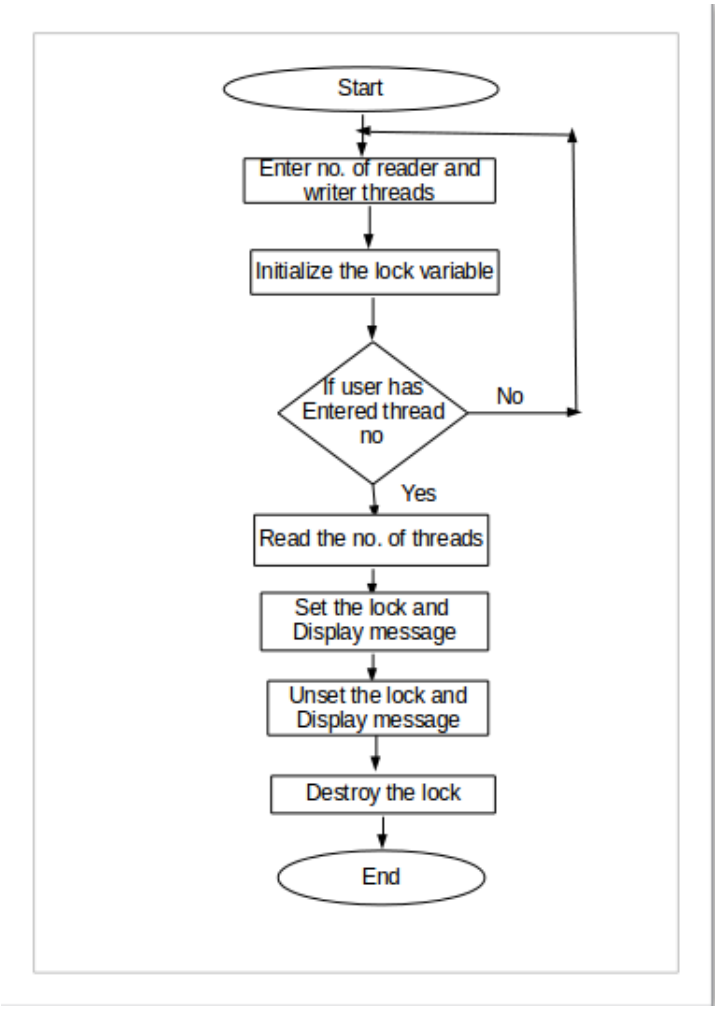


Figure 1: Reader-writer model

5. Mathematical model:

Aim:

Let system 'S' be the solution for the execution of the reader and writer threads.

Initialisation:

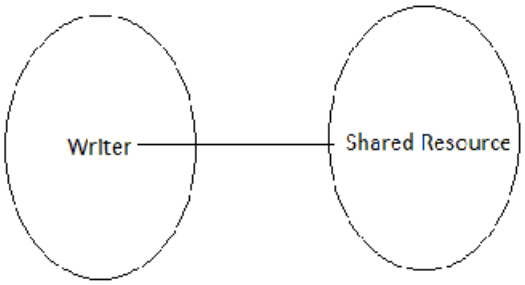
Data Members:Shared resources.
Calls= init(),set lock(),unset lock(),destroy()
Input=Number of reader and writer threads
Output=Reader and writer thread execution

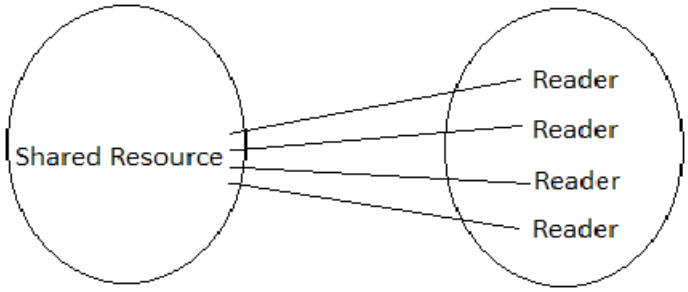
Mathematical model using Set theory:

$$Assignment of Reader and Writer threads(S)$$
$$L = \{(s, e, i, o, f, DD, NDD, success, failure)\}$$

s= Initial state
e= End of state
i=input set i.e.number of reader and writer thread
o=output set i.e execution of the threads
DD=deterministic data
NDD=Non deterministic data
success=system reaches desired state i.e executes reader and writer operation successfully
failure=desired outcome not generated. i.e threads are not assigned properly

Venn diagram:



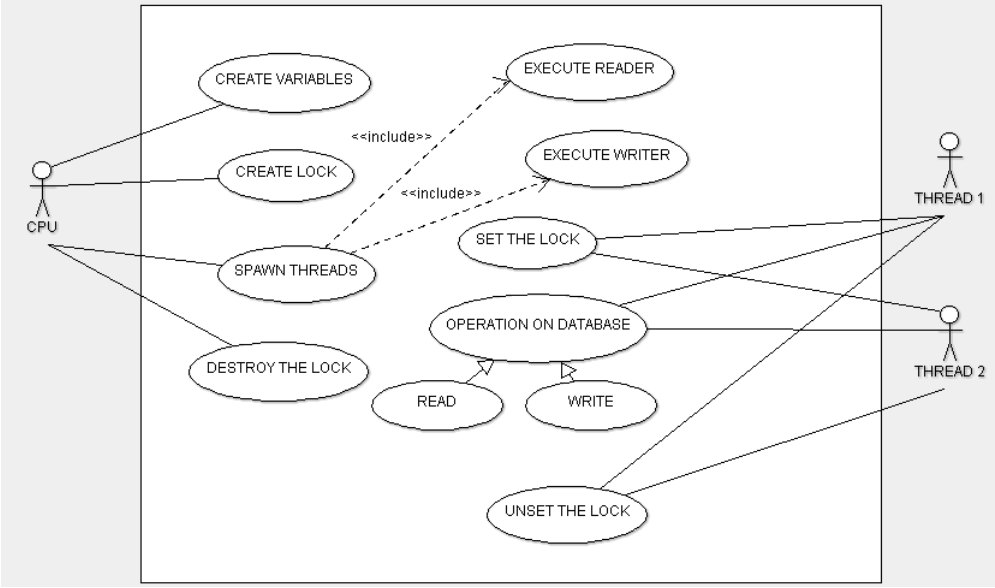


6. SRS:

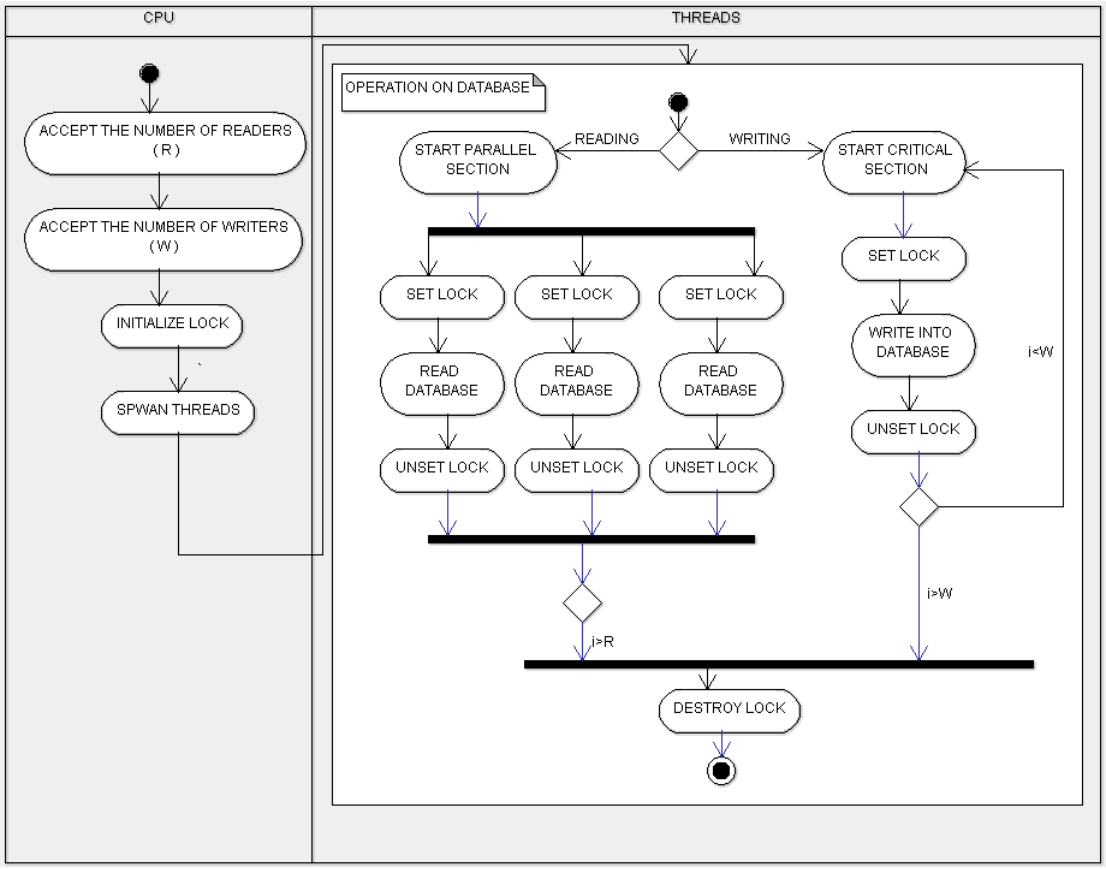
Specification	Description
Project Scope	To understand shared memory concept for reader and writer operation
Functional Requirement	To allow multiple readers to read and single writer to write.
Design and implementation	User gives the number of reader/writer threads and CPU assigns them
Machine Specification	32/64 bit CPU ,128 bit RAM

7. UML Diagrams:

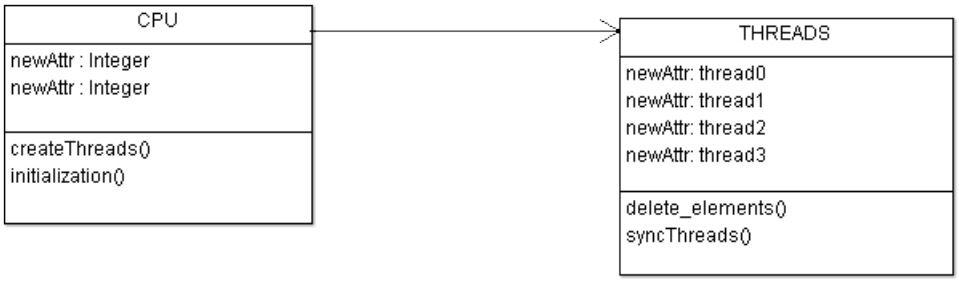
Use case Diagram:



Sequence Diagram:



Class Diagram:



- **Input:**
  Input to this program is reader and writer threads number.
- **Output:**
  Output of this program is execution of the threads.
- **Conclusion :**
  Thus we have implemented the reader and writer problem in openmp.

Course Outcomes	Achieved Outcome
CO I : Ability to perform multi-core, Concurrent and Distributed Programming.	√
CO II : Ability to perform Embedded Operating Systems Programming using Beaglebone	
CO III :Ability to write Software Engineering Document.	√
CO IV :Ability to perform Concurrent Programming using GPU.	

FAQ:

- What are the advantages of openmp?
- What are the applications of Openmp?
- Which omp functions are called in reader writer problem?
- Which omp call is needed for iniialiazation of the resource lock?
- Which omp call is needed for destroying of the resource lock?