International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 256-263 www.ijaem.net ISSN: 2395-5252

A Theory of Synchronisation using Semophores

Suveetha.V, Sree Dharshni.V.S, Akshaya.A, Dr.M. Sujithra M.C.A,M.Phil.,PhD, Dr.A.D. Chitra M.C.A, M.Phil.,PhD,

2nd year , m.sc. Software systems (integrated), coimbatore institute of technology, coimbatore
Assistantprofessor, , department of software systems, coimbatore institute of technology,

Date of Submission: 10-11-2020 Date of Acceptance: 24-11-2020

ABSTRACT: In a multi process system when two or more process running at a same time accessing the same shared resource may lead to the inconsistency of data. synchronization between more than one process is needed so that there will no collision between two processes . process synchronization is used to handle concurrent access to shared data. Semaphore is one of the basic synchronization primitive where it has only two operations wait and signal .it is initialized to non-negative value. This paper introduces a formal definition of semaphore and illustrates a general theory of synchronization an various problem and solutions that come under semaphoresynchronization.

Keywords: process , mutex , wait , signal , synchronization, critical section.

I. INTRODUCTION

In the operating system, process is a task that is currently under execution.During the execution it undergoes few states like new, ready, running, waiting, terminate etc. A process is said to be co-operating if the execution of a process can affect or be affected by the execution of other processes. Process synchronization is a method or way to coordinate two or more processes running at same time to avoid data collision. Data synchronization which is important during the process execution, data synchronization means a way to keep multiple data copies in coherence with one another. Process synchronization is commonly used to execute data synchronization. The need arises especially in the simulation of many process at same time. It is been used frequently when multiple process need to execute simultaneously. The other purpose is the coordination of process interactions in an operating system.

Exit section

Critical section:
In a program, there are four important section like entry section, critical section , exit section, remainder section. A critical section is segment of code which can be accessed by a signal process at a same point of time.

entry section

```
do {

                critical section

}while(TRUE);
```

Fig-1 critical section

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 256-263 www.ijaem.net ISSN: 2395-5252

The entry to the critical section and the exit from the critical section is handled by the wait() and the signal() function . Here only a single process can be executed and the other process has to wait for its turn.

II. METHODOLOGY

Three main condition to solve the critical section problem

Mutual exclusion:Out of a group of cooperating processes, only one process can be in its critical section at a specific time.

Process:If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.

Bounded waiting:After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, system must grant the process permission to get into its critical section.

Mutex lock:A concept related to the semaphore, mutex is a programming flag used to grab and release the an object. Mutexes are just simple locks obtained before entering its critical section and then releasing it. Since only one thread is in its critical section at any given time, there are no race conditions, and data always remain consistent.It has some disadvantage like If a thread obtains a lock and goes to sleeporitis pre-empted, then the other thread may not able to move forward.

This may lead to starvation. It can't be locked or unlocked from a different context than the one thatacquiredit. Only one thread should be allowed in the critical section at a time. Thenormal implementation may lead to busy waiting state, which wastes CPUtime.

III. SEMAPHORE

SEMAPHORES : For signalling all means of communication (execution of process)a special variable is used , and this special variable is called semaphores .(i.e) an integer value ..The two main atomic operations that can be performed on semaphores are :Initialize**,**Decrement (sem wait),Increment (sem signal)here is no way to manipulate semaphores other than these two operations .A Semaphore can be initialised to an non-negative integer value . The Decrement operations may result in the blocking of process .The Increment operations may result in unblocking of process.

Wait and Signal are the two operations used for process synchronization .

Wait : The wait operation decrements the value of the semaphore (S). If the value of the semaphore S is negative , then no operation is performed , else the process continues execution.

Signal : The **signal** operation increments the semaphore value S .

Wait(S)	signal (S)

{
S--;
}
	{
	S++ ;
	}

Fig-2 wait and signal

TYPES OF SEMAPHORES :

Counting Semaphores :

This type of Semaphores uses a count that helps tasks to be acquired or released various times . If initially count is 0 , then the semaphore is created in the unavailable state and if count value is greater than 0 , then the semaphore is created in the available state.

Binary Semaphores :

The Binary Semaphore is quite similar to the Counting Semaphores , but it may only take values 0 and 1 .In this type of Semaphore , the **wait** operation works only when the semaphore value is equal to 1 (S=1) and the **signal** operation works when semaphore value is equal to 0 (S=0).

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 256-263 www.ijaem.net ISSN: 2395-5252

Pseudocode for counting semaphore :

```
void semWait ( semaphore s )
{
        s.count ++ ;
        if ( s.count < 0 ) {
          // place the process in s.queue ;
          // block the process ;
        }
}
void semSignal ( semaphore s )
{
  s.count ++ ;
  if ( s.count <= 0 )
      //remove process from s.queue ;
      //place process on ready list ;
   }
}
```

Fig-3 semwait and semsignal

Pseudocode for binary semaphore :

```
void semWaitB(binary_semaphore s)
{
        if (s.value == one)
            s.value=zero ;
        else {
          // place the process in s.queue ;
          // block the process ;
        }
}
void semSignalB(semaphore s)
{
        if (s.queue is empty ())
             s.value = one ;
        else {
              //remove process from s.queue ;
              //place process on ready list ;
        }
}
```

Fig-4 semwait and semsignal

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 256-263 www.ijaem.net ISSN: 2395-5252

IV. PROPOSED SOLUTION TO SYNCHRONIZATION

Producer Consumer Problem : It is also known as Bounded Buffer problem.We have a buffer of N fixed size. One or more producer are generating data and placing it in the buffer .A consumer can take items out of the buffer and can consume them .The producer should produce data only when the buffer is not full. The consumer should consume data only when the buffer is not empty. The producer and consumer should not access the buffer at the sametime.

```
void producer()
{
   mutex=wait(mutex);
   full=signal(full);
   empty=wait(empty);
   x++;
   printf("\nProducer produces theitem
%d",x);
   printf("\nRemaining slots in the buffer
=%d ",empty);
   mutex=signal(mutex);
}
```

Fig-5 producer

```
void consumer()
{
   mutex=wait(mutex);
   full=wait(full);
   empty=signal(empty);
printf("\nConsumer consumes item
%d",x);
   x--;
   mutex=signal(mutex);
}
```

Fig-6 consumer

Inference: When producer produces an item then the value of "empty" is reduced by 1 because one slot willbefillednow.Thevalueofmutexisalsoreducedtopreventconsumertoaccessthebuffer. Now,theproducerhasplacedtheitemandthusthevalue of"full"is increasedby1.Thevalue ofmutexisalsoincreased by1becausethetaskofproducerhasbeencompletedand consumer can access thebuffer. Astheconsumerisremovinganitemfrombuffer,therefore thevalueof"full"is reducedby1 and the value is mutex is also reduced so that the producer cannot access the buffer at this moment. Now, the consumer has consumed the item, thus increasing the value of "empty"by

1. The value of mutex is also increased so that producer can access the buffernow.

Readers Writers Problem :

If two readers access the object at the same time there is no problem. However if two writers or a reader and writer access the object at the same time, there may be problems. To solve this situation, a writer should get exclusive access to an object i.e. when a writer is accessing the object, no reader or writer may access it. However, multiple readers can access the object at the sametime.

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 256-263 www.ijaem.net ISSN: 2395-5252

```
void addReader(struct semaphore *s)
{
if (s->mutex == 0 && s->readcount == 0)
{
printf("\nSorry, File open in Write mode.\nNew Reader
added to queue.\n");
s->readwait++;
}
else
{
printf("\nReader Process added.\n");
s->readcount++;
s->mutex--;
}
return ;
}



void addWriter(struct semaphore *s)
{
if(s->mutex==1)
{
s->mutex--
; s-
>write=1;
printf("\nWriter Process added.\n");
}
else if(s->write) printf("\nSorry, Writer already
operational.\n");
else printf("\nSorry, File open in Read mode.\n");
return ;}
```

Fig-7 addReader and addWriter

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 256-263 www.ijaem.net ISSN: 2395-5252

```
void remReader(struct semaphore *s)
{
if(s->readcount == 0)
printf("\nNo readers to remove.\n");
else
{
printf("\nReader Removed.\n");
s->readcount--;
s->mutex++;
}
return ;
}


void remWriter(struct semaphore *s)
{
if(s->write==0)
printf("\nNo Writer to Remove");
else
{
printf("\nWriter Removed\n");
s->mutex++;
s->write=0;
if(s->readwait!=0)
{
s->mutex-=s->readwait;
s->readcount=s->readwait;
s->readwait=0;
printf("%d waiting Readers Added.",s->readcount);
}
}
}
   continue;
}
```

Fig-8 remReader and remWriter

Inference:

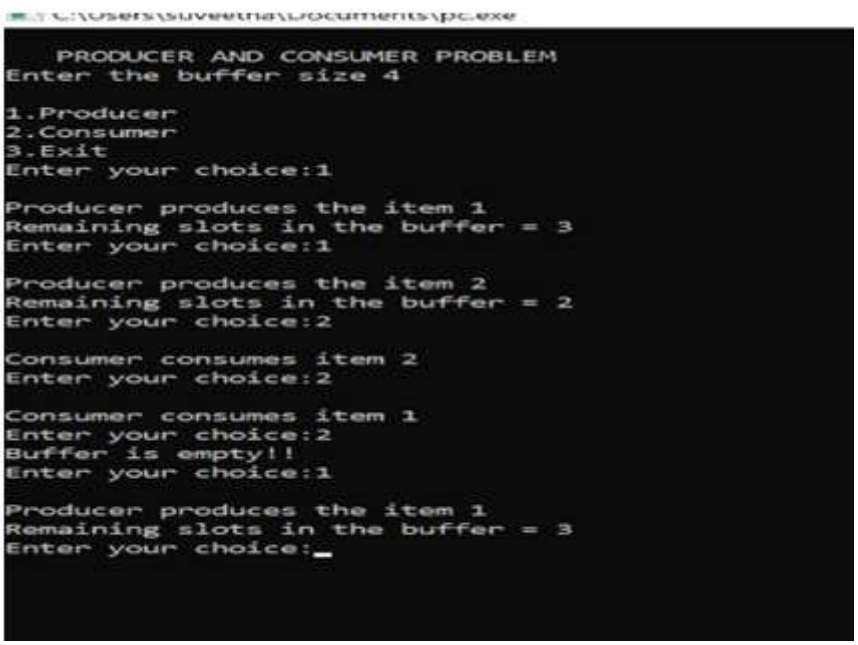
Aresourceissharedamong manyprocesses,eachbelongingtooneoftwoprocesses. Theyare either reader or writer. In this problem, any number of readers can read from the shared resource simultaneously, but at a time only one writer can write to the shared resource. Also, when a writer is writing data to the resource, at that time no other process can access the resource. Readers do not write, readers only read. If a process is writing, no other process can readit.

V. RESULT OBTAINED

Producer consumer problem:

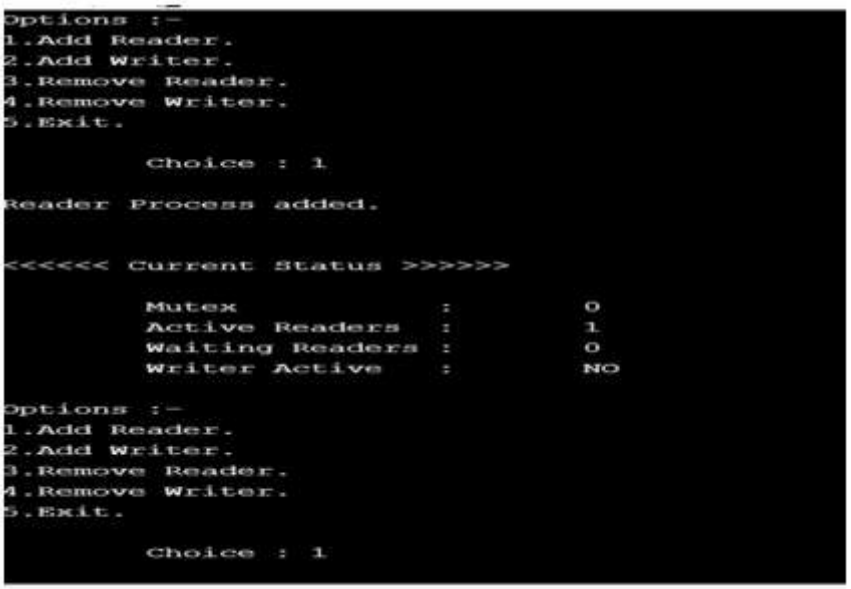
Here there is a buffer of ns lots which stores the data and there are two process namely producer the job of producer is to produce the item and place it in the buffer it cannot produce when the bufferis empty.The job of consumer is to take the items ou of the bufferitcanonlytakewhen the buffer has at least one item. It they only function one at a time.

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 9, pp: 256-263 www.ijaem.net ISSN: 2395-5252



Reader writer problem:

Readinginformationfromthedatabasewillno tcauseaproblemsincenodataischanged.The problem lies in writing information to the data base .If no constraints are put on access to the data base, data may change at any moment. By the time a reading process displays the result of a request for information to the user, the actual data in the data base may have changed. What if, for instance, a process reads the number of available seats on a flight, finds a valueof one,and reports it to the customer. Before the customer has a chance to make their reservation, another process makes a reservation for another customer, changing the number of available seats tozero.



VI. SUMMARY

In operating systems, when two or more process run at the same time . There is chance for data collision. To avoid this data synchronization plays an important role . It can be achieved using process synchronization. This process synchronization is used to manage the concurrent access to the shared data . One of the basic tool used is semaphore. Semaphore synchronization is the method or way to coordinate two or more processes running at the same time to avoid data collision. There are two operations used wait and signal and it is initialized to non negative value. Mainly there are two problems producer consumer problem and reader writer problem. The other purpose is the coordination of process interactions in the operatingsystem.

VII. CONCLUSION

Operating Systems play an important role in system performance. It is said to be a set of programs that manages the computer hardware and some application software. Here We have introduced a formal definition and the synchronization tool called semaphore, which recovers one of the main issues in operating systems (i.e.,) concurrent program executions.Our purpose is to synchronize the process so that each data gets a mutually exclusive access to the shared cresources. This will be achieved through a synchronization tool called semaphore, which allows only a single process at a time to access the data and send its data successfully without any collision with other data. Further our work includes the possible applications of the semaphore synchronization and optimization of thecode.

REFERENCES

[1]. Kartik Pandya, "Network Structure or Topology", International Journal ofAdvance Research in Computer Science and Management Studies, Volume 1, Issue 2, July 2013

[2]. Dhananjay M.DhamDhere, "Operating Systems A Concept – Based Approach", 3rdEdition, McGraw Hill Education (India) Private Limited, New Delhi,2003

[3]. David P. Reed and Rajendra K. Kanodia. Synchronization with eventcounts and sequencers. Communications of the ACM, 22(2), February 1979. Proposesdifferent synchronizationmechanism.

[4]. K. A. Sumitradevi, N. P. Banashree, "Operating Systems", second edition, SPD publications.

[5]. Brian P. Crow, Jeong Geun Kim, Prescott T. Sakai," IEEE 802.11 WirelessLocal Area Networks", IEEE Communications Magazine, September1997.

[6]. William Stallings, "Operating Systems: Internals and Design Principles", seventh Edition, Pearson Publications.