Introduction to Semaphore in Operating Systems (OS)

In thi tutorial, we are about to learn about the most important topic called Semaphores. There is a 100% surety that there is going to be a question about the topic named Semaphores in Viva, Interviews, Exams, and even Placement Exams. So, please understand this topic with utmost care and preference.

In this topic, we are going to learn about Semaphore definition, Types of Semaphores, Operations of Semaphores, Advantages and Disadvantages in Semaphores, Process of Solving Classical Synchronization Problems using Semaphores and the usage of these types of Semaphores in solving these Classical Synchronization Problems.

Semaphore Definition

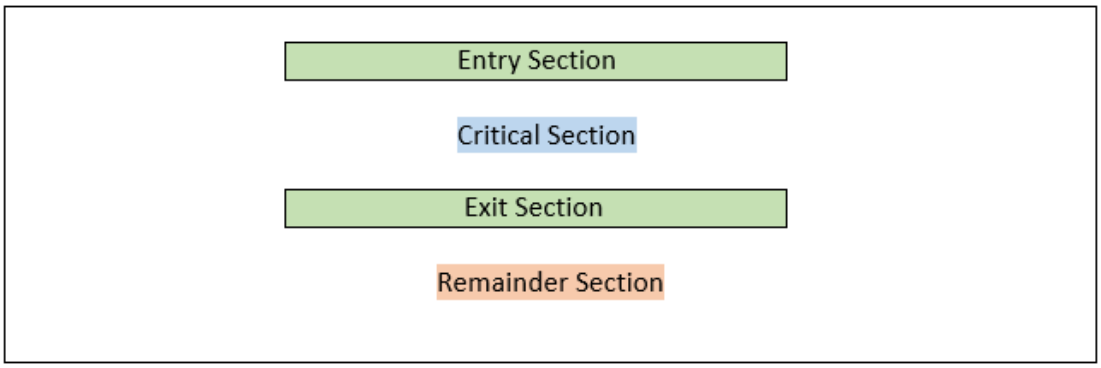
Semaphore is a Hardware Solution. This Hardware solution is written or given to critical section problem.

What is a Critical Section Problem?

The Critical Section Problem is a Code Snippet. This code snippet contains a few variables. These variables can be accessed by a few processes. There is a condition for these processes.

The condition is that only one process can only enter the critical section. Remaining Processes which are interested to enter the critical section have to wait for the process to complete its work and then enter the critical section.

Critical Section Representation



Problems in Critical Section Problems

There may be a state where one or more processes try to enter the critical state. After multiple processes enter the Critical Section, the second process try to access variable which already accessed by the first process.

Explanation

Suppose there is a variable which is also known as shared variable. Let us define that shared variable.

Here, x is the shared variable.

```
int x = 10;
```

Process 1

```
// Process 1
int s = 10;
int u = 20;
x = s + u;
```

Process 2

```
// Process 2
```

```
int s = 10;
int u = 20;
x = s - u;
```

If the process is accessed the x shared variable one after other, then we are going to be in a good position.

If Process 1 is alone executed, then the value of x is denoted as x = 30;

The shared variable x changes to 30 from 10

If Process 2 is alone executed, then the value of x is denoted as x = -10;

The shared variable x changes to -10 from 30

If both the processes occur at the same time, then the compiler would be in a confusion to choose which variable value i.e. -10 or 30. This state faced by the variable x is Data Inconsistency. These problems can also be solved by Hardware Locks

To, prevent such kind of problems can also be solved by Hardware solutions named Semaphores.

Semaphores

The Semaphore is just a normal integer. The Semaphore cannot be negative. The least value for a Semaphore is zero (0). The Maximum value of a Semaphore can be anything. The Semaphores usually have two operations. The two operations have the capability to decide the values of the semaphores.

The two Semaphore Operations are:

1. Wait ( )
2. Signal ( )

Wait Semaphore Operation

The Wait Operation is used for deciding the condition for the process to enter the critical state or wait for execution of process. Here, the wait operation has many different names. The different names are:

1. Sleep Operation
2. Down Operation
3. Decrease Operation
4. P Function (most important alias name for wait operation)

The Wait Operation works on the basis of Semaphore or Mutex Value.

Here, if the Semaphore value is greater than zero or positive then the Process can enter the Critical Section Area.

If the Semaphore value is equal to zero then the Process has to wait for the Process to exit the Critical Section Area.

This function is only present until the process enters the critical state. If the Processes enters the critical state, then the P Function or Wait Operation has no job to do.

If the Process exits the Critical Section we have to reduce the value of Semaphore

Basic Algorithm of P Function or Wait Operation

```
P (Semaphore value)
{
Allow the process to enter if the value of Semaphore is greater than zero or positive.
Do not allow the process if the value of Semaphore is less than zero or zero.
Decrement the Semaphore value if the Process leaves the Critical State.

}
```

Signal Semaphore Operation

The Signal Semaphore Operation is used to update the value of Semaphore. The Semaphore value is updated when the new processes are ready to enter the Critical Section.

The Signal Operation is also known as:

1. Wake up Operation
2. Up Operation
3. Increase Operation
4. V Function (most important alias name for signal operation)

We know that the semaphore value is decreased by one in the wait operation when the process left the critical state. So, to counter balance the decreased number 1 we use signal operation which increments the semaphore value. This induces the critical section to receive more and more processes into it.

The most important part is that this Signal Operation or V Function is executed only when the process comes out of the critical section. The value of semaphore cannot be incremented before the exit of process from the critical section

Basic Algorithm of V Function or Signal Operation

```
V (Semaphore value)
{
If the process goes out of the critical section then add 1 to the semaphore value
Else keep calm until process exits
}
```

Types of Semaphores

There are two types of Semaphores.

They are:

1. Binary Semaphore

Here, there are only two values of Semaphore in Binary Semaphore Concept. The two values are 1 and 0.

If the Value of Binary Semaphore is 1, then the process has the capability to enter the critical section area. If the value of Binary Semaphore is 0 then the process does not have the capability to enter the critical section area.

2. Counting Semaphore

Here, there are two sets of values of Semaphore in Counting Semaphore Concept. The two types of values are values greater than and equal to one and other type is value equal to zero.

If the Value of Binary Semaphore is greater than or equal to 1, then the process has the capability to enter the critical section area. If the value of Binary Semaphore is 0 then the process does not have the capability to enter the critical section area.

This is the brief description about the Binary and Counting Semaphores. You will learn still more about them in next articles.

Advantages of a Semaphore

- Semaphores are machine independent since their implementation and codes are written in the microkernel's machine independent code area.
- They strictly enforce mutual exclusion and let processes enter the crucial part one at a time (only in the case of binary semaphores).
- With the use of semaphores, no resources are lost due to busy waiting since we do not need any processor time to verify that a condition is met before allowing a process access to the crucial area.
- Semaphores have the very good management of resources
- They forbid several processes from entering the crucial area. They are significantly more effective than other synchronization approaches since mutual exclusion is made possible in this way.

Disadvantages of a Semaphore

- Due to the employment of semaphores, it is possible for high priority processes to reach the vital area before low priority processes.

- Because semaphores are a little complex, it is important to design the wait and signal actions in a way that avoids deadlocks.
- Programming a semaphore is very challenging, and there is a danger that mutual exclusion won't be achieved.
- The wait ( ) and signal ( ) actions must be carried out in the appropriate order to prevent deadlocks.

Now, let us solve the Classical Synchronization Problems using the concept of Semaphore.

Solving Classical Synchronization Problems using Semaphore Concept

1) Solving Readers - Writers Problem using Semaphore

Definition:

First of all let us all know about the Readers - Writers Problem.

In Readers - Writers Problems there are two Actors for the problem. They are Reader and Writer. The Reader - Writer Problem tries to access or change the value of the Shared Variable. Due, to this parallel execution of accessing and changing operation data faults take place. Due, to this expected outputs are not the actual outputs of the problem. This is the problem of Readers - Writers

The duties of Readers and Writers are different from each other.

The Reader duty is to read the value of the Shared Variable.

The Writer duty is to change the value of the Shared Variable.

Now, let us solve this problem with the help of a Semaphore.

Required Data

First of all let us consider there is Critical Section which contains a shared variable. Let us consider that there are two processes. The first process is the one which always tries to access the shared variable. The second process is the one which always tries to change the value of the variable.

Solution

Now, let us understand the solution to the Readers and Writers Problem using

Now, our task is to perform the operations of both Readers and Writers using Semaphores without any possibility of occurrence of Deadlock.

Solving Readers - Writers Problem with the help of Binary Semaphore

We know that the Binary Semaphore allows the process to enter the critical section if the value of Binary Semaphore is 1.

Now, we are going to allow the writer to work or execute its task if the value of Binary Semaphore value is 1. This allows that the data changes are error free.

We can allow the reader to work or execute its task if the value of Binary Semaphore is 1 or 0. This is because the Reader just reads the value of Shared Variable. But according to the rules laid by the Semaphore. We can only enter the critical section if the value of Semaphore is greater than or equal to 1.

So, we can allow the reader if the value of semaphore 1.

Algorithm for Solving Readers - Writers Problem with the help of Binary Semaphore

```
/* Process 1 - - - > Reader
Process 2 - - - > Writer */
Solving (int n)
{
if (n==0)
{
//You cannot enter the critical section area.
}
if (n==1)
{
```

```
Enter the process which you want to perform
scan (op)
if (op==1)
{
Read ( )
} // read
else if (op==2)
{
Write ( )
} // write
} // if condition for entering critical section
} // Solving

Main ( )
{
do
{
{
Process (Semaphore)
// Enter the Entry section if the Semaphore value is one.
Entry Section
Critical Section
Exit Section
} // Entry Section
} // main
```

Solving Readers - Writers Problem with the help of Counting Semaphore

We know that the Counting Semaphore allows the process to enter the critical section if the value of Counting Semaphore is greater than or equal to 1.

Now, we are going to allow the writer to work or execute its task if the value of Counting Semaphore value is greater than or equal to 1. This allows that the data changes are error free.

We can allow the reader to work or execute its task if the value of Counting Semaphore is greater than or equal to 1 or Counting Semaphore is equal to 0. This is because the Reader just reads the value of Shared Variable. But according to the rules laid by the Semaphore. We can only enter the critical section if the value of Counting Semaphore is greater than or equal to 1.

So, we can allow the reader if the value of Counting Semaphore is greater than or equal to 1.

Algorithm for Solving Readers - Writers Problem with the help of Binary Semaphore

```
/* Process 1 - - - > Reader
Process 2 - - - > Writer */
Solving (int n)
{
if (n==0)
{
//You cannot enter the critical section area.
}
if (n>=1)
{
Enter the process which you want to perform
scan (op)
if (op==1)
{
Read ( )
} // read
else if (op==2)
{
Write ( )
} // write
} // if condition for entering critical section
} // Solving

Main ( )
```

```
{
do
{
Process (Semaphore)
// Enter the Entry section if the Semaphore value is greater or equal to one.
Entry Section
Critical Section
Exit Section
Remainder Section
} // do
while (true);
} // Entry Section
} // main
```

2. Solving Bound Buffer Problem using the concept of Semaphores

Definition:

The Producer-Consumer problem is another name for Bound Buffer Problem. In this issue, there are n slots in a buffer, and each slot may hold one data unit. Producer and Consumer are the two operations that are using the buffer. Both the producer and the consumer attempt to enter and delete data.

Solution:

Now, we are going to solve the problem faced in the Bound Buffer or Producer Consumer Problem.

We already know that the duty of the Producer is to enter data in which ever area possible.

The duty of Consumer is to remove the data produced by the Producer or already present data where ever possible.

Now, let us understand how these two Producers and Consumers are going to work along with the concepts of Binary and Counting Semaphore.

Solving Bound Buffer or Producer Consumer Problem with the help of Binary Semaphore

We know that the Binary Semaphore allows the process to enter the critical section if the value of Binary Semaphore is 1.

Now, we are going to allow the writer to work or execute its task if the value of Binary Semaphore value is 1. This allows that the data changes are error free.

Now, let us assume that the Producer is creating a few shared variables inside the critical section.

Now, let us allow the process named Producer and produce new value inside the critical section after entering the Critical Section. The entry is accepted only if the value of Binary Semaphore is equal to 1.

Now, let us allow the process named Consumer and delete the processes created after entering the Critical Section. The entry is accepted only if the value of Binary Semaphore is equal to 1.

Now, it is our duty to prevent the Producer and Consumer gets executed at the same time. So, we give the user or the computer an option to choose which operation to be performed at the time of Entering the Entry Section of the Critical Section

Solving Bound Buffer or Producer Consumer Problem with the help of Counting Semaphore

We know that the Binary Semaphore allows the process to enter the critical section if the value of Counting Semaphore is 1 or greater than 1.

Now, we are going to allow the writer to work or execute its task if the value of Counting Semaphore is 1 or greater than 1.. This allows that the data changes are error free.

Now, let us assume that the Producer is creating a few shared variables inside the critical section.

Now, let us allow the process named Producer and produce new value inside the critical section after entering the Critical Section. The entry is accepted only if the value of Counting Semaphore is equal to or greater than 1.

Now, let us allow the process named Consumer and delete the processes created after entering the Critical Section. The entry is accepted only if the value of Counting Semaphore is 1 or greater than 1.

Now, it is our duty to prevent the Producer and Consumer gets executed at the same time. So, we give the user or the computer an option to choose which operation to be performed at the time of Entering the Entry Section of the Critical

Section

3. Solving Dining Philosopher's Problem using the concept of Semaphores

Definition:

The dining philosopher's dilemma, also known as the classical synchronization issue, has five philosophers seated around a circular table who must alternate between thinking and eating. At the center of the table sits a bowl of noodles and five chopsticks, one for each of the philosophers. A philosopher must use both their right and left chopsticks in order to eat. A philosopher can only eat if both of their right and left chopsticks are within reach. If the philosopher's left and right chopsticks are not immediately available, the philosopher sets down one of the chopsticks (either the left or right) and resumes pondering.

The Dining Philosopher is a classic synchronization problem because it illustrates a broad class of concurrency control problems.

Now, you might have understood how Semaphores are useful in solving the problems created by the Synchronization Process.

This is all about Semaphores in Operating Systems.