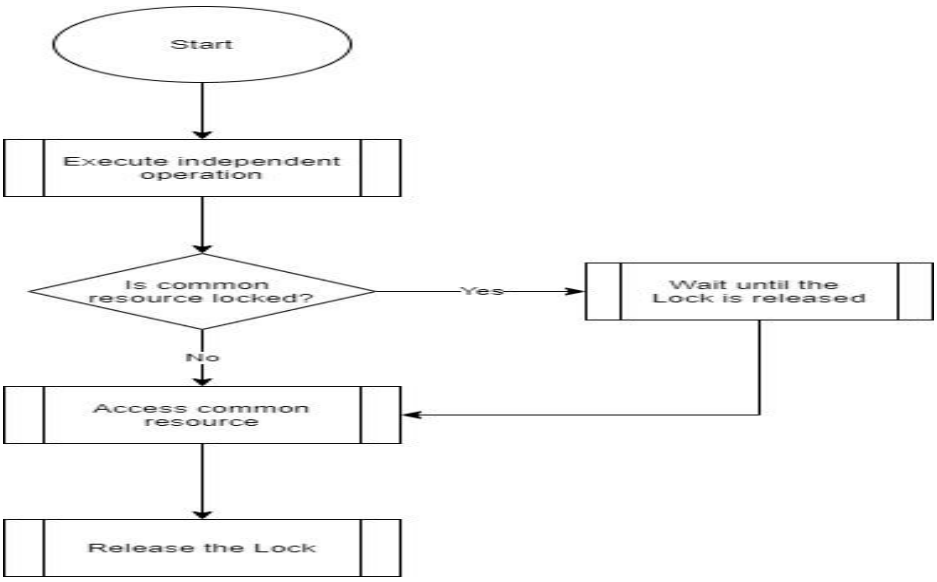
Assignment No. 4: Write a program to solve Classical Problems of Synchronization using Mutex and Semaphore.

Theory:

Mutex

- Mutex is used to ensure only one thread at a time can access the resource protected by the mutex. The process that look the mutex must be the one that unlock it. Mutex is good only for managing mutual exclusion to some shared resource.
- Mutex is easy and efficient for implement.
- Mutex can be in one of two states : locked or unlocked.
- Mutex is represented by one bit. Zero (0) means unlock and other value represents locked. It uses two procedures.
- When a process need access to a critical section, it checks the condition of mutex_locks. If the mutex is currently unlocked then calling process enters into critical section.
- If the mutex is locked , the calling process is enters into blocked state and wait until the process in the critical section finishes its execution .
- Mutex variable have only two states so they are simple implement. Their use is limited to guarding entries to critical resigns.
- Mutex variable is like a binary semaphore. But both are not same.

Algorithm:



Semaphore :

- Semaphore is described by Dijkstra . Semaphore is a non-negative integer variable that is used as a flag. Semaphore is an operating system abstract data type. If takes only integer value. Its used to solve critical section problem.
- Dijkstra introduces two operations (p and v) to operate on semaphore to solve process synchronization problem. A process calls the p operation when it wants to enter its critical section and calls v operation when it when to exit its critical section. The p operation and is called as signal operation.
- A wait operation on a semaphore decease its value by one.

```
Waits  : < 0
        Do loops;
   S: = S – 1;
```

- A signal operation increments its value:

```
Signal:
 S := S  +  1;
```

- A proper semaphore implementation requires that p and v indivisible operations. A semaphore operation is atomic. This may be possible by taking hardware support. The operation p and v are executed by the operating system in response to calls issued by any one process naming a semaphore as parameter.
- There is no guarantee that no two processes can execute wait and signal operations on the same semaphore at the same time.

Properties of semaphore :

1. Semaphores are machine independent.
2. Semaphores are simple to implement.
3. Correctness is easy to determine.
4. Semaphore acquire many resources simultaneously.

Types of Semaphores :

There are mainly two types of Semaphores, or two types of signaling integer variables:

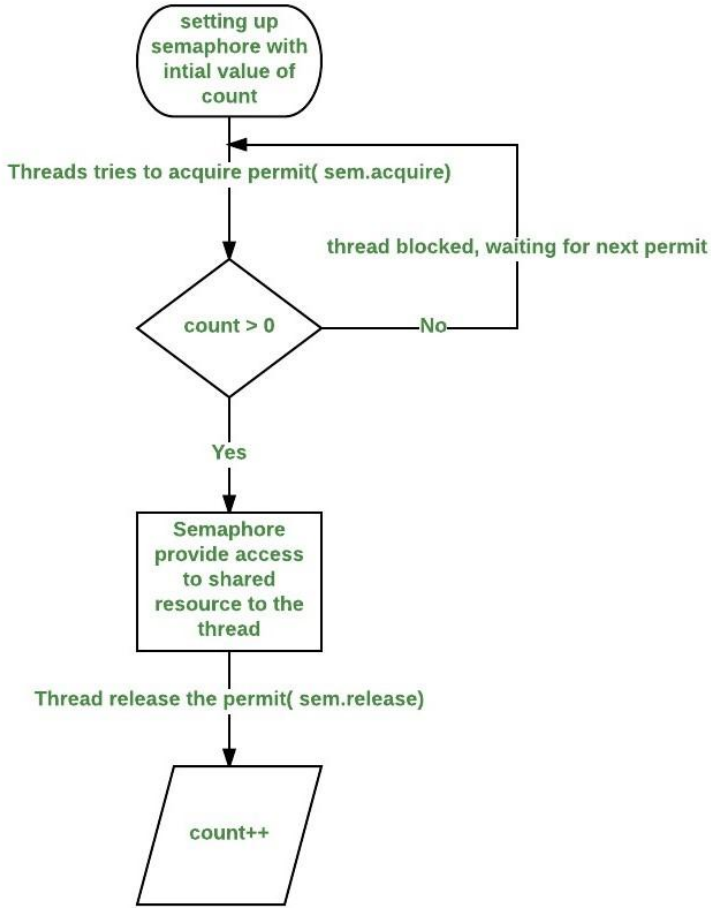
Binary Semaphores :

In these type of Semaphores the integer value of the semaphore can only be either 0 or 1. If the value of the Semaphore is 1, it means that the process can proceed to the critical section (the common section that the processes need to access). However, if the value of binary semaphore is 0, then the process cannot continue to the critical section of the code. When a process is using the critical section of the code, we change the Semaphore value to 0, and when a process is not using it, or we can allow a process to access the critical section, we change the value of semaphore to 1. Binary semaphore is also called mutex lock.

Counting Semaphores :

Counting semaphores are signaling integers that can take on any integer value. Using these Semaphores we can coordinate access to resources and here the Semaphore count is the number of resources available. If the value of the Semaphore is anywhere above 0, processes can access the critical section, or the shared resources. The number of processes that can access the resources / code is the value of the semaphore. However, if the value is 0, it means that there aren't any resources that are available or the critical section is already being accessed by a number of processes and cannot be accessed by more processes.

Algorithm:



Conclusion: In this practical we successfully solve classical problems of synchronization using Mutex and Semaphore.