Thread Concept

Subject:- Unix Operating System

System Lab Class :- TYIT

Name PRN

Aditi Sudhir Ghate 2020BTEIT00044

Assignment No - 4C

Title- Write program to synchronize threads using construct – monitor/serialize/semaphore of Java.

Objectives –

1. To learn about threading in Linux/Unix and Java and difference between them..
2. Use of system call/library to write effective programs

Theory-

A semaphore controls access to a shared resource through the use of a counter. If the counter is greater than zero, then access is allowed. If it is zero, then access is denied. What the counter is counting are permits that allow access to the shared resource. Thus, to access the resource, a thread must be granted a permit from the semaphore.

Working of semaphore : In general, to use a semaphore, the thread that wants access to the shared resource tries to acquire a permit.

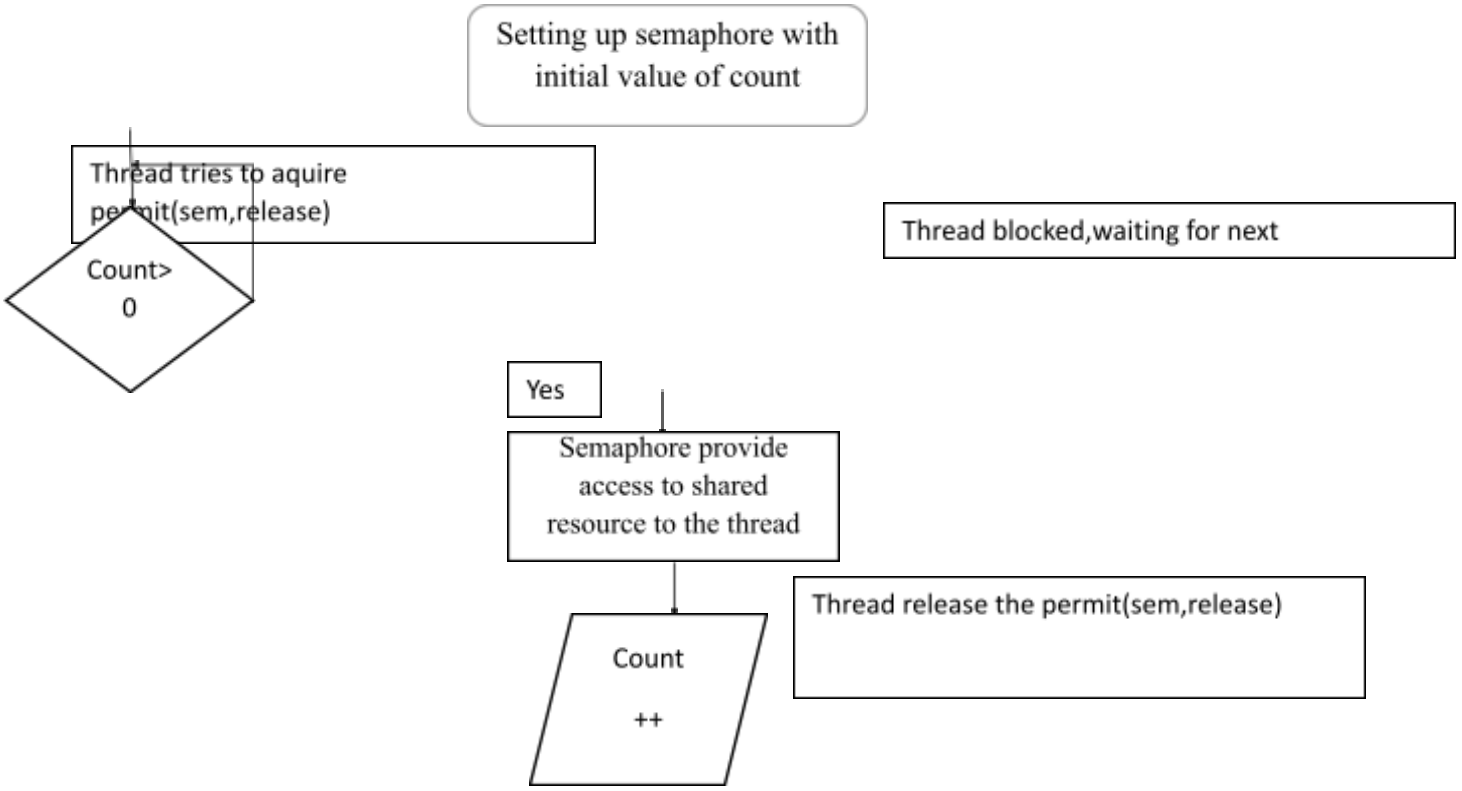
• If the semaphore's count is greater than zero, then the thread acquires a permit, which causes the semaphore's count to be decremented.

• Otherwise, the thread will be blocked until a permit can be acquired.

• When the thread no longer needs an access to the shared resource, it releases the permit, which causes the semaphore's count to be incremented.

• If there is another thread waiting for a permit, then that thread will acquire a permit at that time. Java provide Semaphore class in java.util.concurrent package that implements this mechanism, so you don't have to implement your own semaphores.

Flowchart-



Program-

```
import java.util.concurrent.*;
public class sem extends Thread
{
public static Semaphore semaphore = new Semaphore(1);
 public static void main(String args[])
throws Exception {
Thread1 t1 = new Thread1();
Thread2 t2 = new Thread2();
sem tm = new sem();
tm.start();
t1.start();
t2.start();
}
public void run()
{
try
{
semaphore.acquire();
for(int i=0;i<=20;i=i+2)
{
System.out.println("Even: "+i);
}
semaphore.release();
```

```
}
catch(InterruptedException exc){}
}
static class Thread1 extends Thread
{
public void run()
{
try
{
semaphore.acquire();
for(int i=1;i<=21;i=i+2)
{
System.out.println("Odd: "+i);
}
semaphore.release();
}
catch(InterruptedException exc){}
}
}
static class Thread2 extends Thread
{
public void run()
{
try
{
semaphore.acquire();
for(int i=2;i<50;i++)
{
int count = 0;
for(int j=2;j<i;j++)
{
if(i%j==0)
{
count++;
}
}
if(count==0)
System.out.println("Prime: "+i);
}
semaphore.release();
}
catch(InterruptedException exc){}
}
}
}
```

Output-

```
aditi@aditi-Lenovo-ideapad-330S-14IKB-U:~/ADnOR/Assignments/4C$ javac sem.java
aditi@aditi-Lenovo-ideapad-330S-14IKB-U:~/ADnOR/Assignments/4C$ java sem
Odd: 1
Odd: 3
Odd: 5
Odd: 7
Odd: 9
Odd: 11
Odd: 13
Odd: 15
Odd: 17
Odd: 19
Odd: 21
Prime: 2
Prime: 3
Prime: 5
Prime: 7
Prime: 11
Prime: 13
Prime: 17
Prime: 19
Prime: 23
Prime: 29
Prime: 31
Prime: 37
Prime: 41
Prime: 43
Prime: 47
Even: 0
Even: 2
Even: 4
Even: 6
Even: 8
Even: 10
Even: 12
Even: 14
Even: 16
Even: 18
Even: 20
```

Conclusion:

Synchronization of multiple threads using semaphore to let threads work synchronously to produce desirable outputs learned and implemented in Java

References:

[1] https://www.geeksforgeeks.org/multithreading-in-java