**4.8** Write program using semaphore to ensure that function f1 shouldexecuted after f2 in 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:**

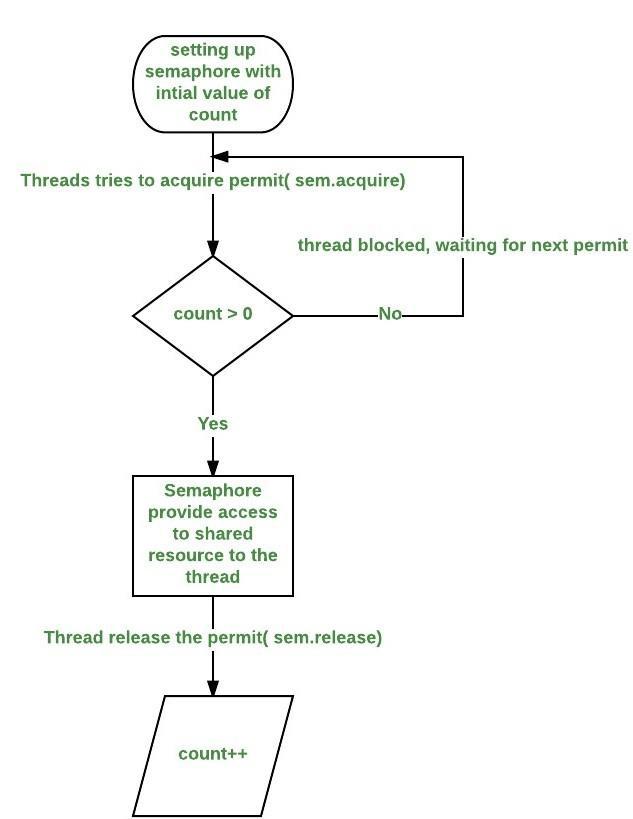
A Semaphore is a thread synchronization construct that can be used either to send signals between threads to avoid [missed signals,](http://tutorials.jenkov.com/java-concurrency/thread-signaling.html#missedsignals) or to guard a [critical section](http://tutorials.jenkov.com/java-concurrency/race-conditions-and-critical-sections.html) like you would with a [lock.](http://tutorials.jenkov.com/java-concurrency/locks.html) Methods in Semaphore Class:

* **void acquire()** : This method acquires a permit, if one is available and returnsimmediately, reducing the number of available permits by one.
* **void release()** : This method releases a permit, increasing the number of availablepermits by one. If any threads are trying to acquire a permit, then one is selected and given the permit that was just released.

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.



**Data Dictionary:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr Number | Variable/Function | Datatype | Use |
|  |  |  |  |
| 1 | sem | Semaphore | Used to synchronize between the threads. |
|  |  |  |  |
| 2 | mt1 | MyThread | Thread 1. |
|  |  |  |  |
| 3 | mt2 | MyThread | Thread 2. |
|  |  |  |  |
| 4 | threadName | String | Name of the thread. |
|  |  |  |  |

**Program:**

import java.util.concurrent.\*;

class Shared

{

static int count = 0;

}

class MyThread extends Thread

{

Semaphore sem;

String threadName;

public MyThread(Semaphore sem, String threadName)

{

super(threadName);

this.sem = sem;

this.threadName = threadName;

}

@Override

public void run() {

if(this.getName().equals("A"))

{

System.out.println("Starting " + threadName);

try

{

System.out.println(threadName + " is waiting for a permit."); sem.acquire();

System.out.println(threadName + " gets a permit.");

for(int i=0; i < 5; i++)

{

Shared.count++;

System.out.println(threadName + ": " + Shared.count);

Thread.sleep(1000);

}

} catch (InterruptedException exc) { System.out.println(exc);

}

System.out.println(threadName + " releases the permit."); sem.release();

}

else

{

System.out.println("Starting " + threadName); try

{

System.out.println(threadName + " is waiting for a permit."); sem.acquire();

System.out.println(threadName + " gets a permit.");

for(int i=0; i < 5; i++)

{

Shared.count--;

System.out.println(threadName + ": " + Shared.count);

Thread.sleep(1000);

}

} catch (InterruptedException exc) { System.out.println(exc);

}

System.out.println(threadName + " releases the permit."); sem.release();

}

}

}

public class H4

{

public static void main(String args[]) throws InterruptedException

{

Semaphore sem = new Semaphore(1);

MyThread mt1 = new MyThread(sem, "A");

MyThread mt2 = new MyThread(sem, "B");

mt1.start();

mt2.start();

mt1.join();

mt2.join();

System.out.println("count: " + Shared.count);

}

}

**Output:**

Starting B

B is waiting for a permit.

Starting A

A is waiting for a permit.

B gets a permit.

1. -1
2. -2
3. -3
4. -4
5. -5

B releases the permit. A gets a permit.

1. -4
2. -3
3. -2
4. -1

A: 0

A releases the permit.

count: 0

**Conclusion:**

Synchronization of functions using semaphores in Java by acquire and release of semaphores studied and implemented.

**References:**

https://www.geeksforgeeks.org/multithreading-in-java/