**Thread concept: clone, threads of java**

**Subject - Unix Operating System**

**Name – Hemant Sharma**

**PRN – 22610001 Class – TYIT**

**Assignment No – 4(g)**

**Title-** Write program to synchronize threads using construct –monitor/serialize/semaphore of Java (In java only).

**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.

**Program:**

import java.util.concurrent.Semaphore;

class NumberPrinterSemaphore {

    private Semaphore evenSem = new Semaphore(1);

    private Semaphore oddSem = new Semaphore(0);

    private Semaphore primeSem = new Semaphore(0);

    public void printEven(int num) {

        try {

            evenSem.acquire();

            System.out.println("Even: " + num);

            oddSem.release();

        } catch (InterruptedException e) {}

    }

    public void printOdd(int num) {

        try {

            oddSem.acquire();

            System.out.println("Odd: " + num);

            primeSem.release();

        } catch (InterruptedException e) {}

    }

    public void printPrime(int num) {

        try {

            primeSem.acquire();

            System.out.println("Prime: " + num);

            evenSem.release();

        } catch (InterruptedException e) {}

    }

}

class EvenThreadSem extends Thread {

    private NumberPrinterSemaphore printer;

    EvenThreadSem(NumberPrinterSemaphore printer) { this.printer = printer; }

    public void run() {

        for (int i = 2; i <= 20; i += 2) {

            printer.printEven(i);

            try { Thread.sleep(500); } catch (InterruptedException e) {}

        }

    }

}

class OddThreadSem extends Thread {

    private NumberPrinterSemaphore printer;

    OddThreadSem(NumberPrinterSemaphore printer) { this.printer = printer; }

    public void run() {

        for (int i = 1; i <= 19; i += 2) {

            printer.printOdd(i);

            try { Thread.sleep(500); } catch (InterruptedException e) {}

        }

    }

}

class PrimeThreadSem extends Thread {

    private NumberPrinterSemaphore printer;

    PrimeThreadSem(NumberPrinterSemaphore printer) { this.printer = printer; }

    private boolean isPrime(int num) {

        if (num < 2) return false;

        for (int i = 2; i \* i <= num; i++)

            if (num % i == 0) return false;

        return true;

    }

    public void run() {

        int count = 0, num = 2;

        while (count < 10) {

            if (isPrime(num)) {

                printer.printPrime(num);

                count++;

                try { Thread.sleep(500); } catch (InterruptedException e) {}

            }

            num++;

        }

    }

}

public class SemaphoreThreads {

    public static void main(String[] args) {

        NumberPrinterSemaphore printer = new NumberPrinterSemaphore();

        EvenThreadSem even = new EvenThreadSem(printer);

        OddThreadSem odd = new OddThreadSem(printer);

        PrimeThreadSem prime = new PrimeThreadSem(printer);

        even.start();

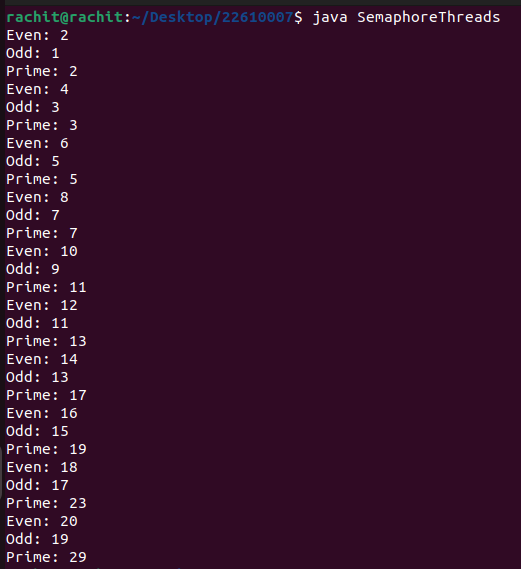
        odd.start();

        prime.start();

    }

}

**Output:**

****

**Conclusion:**

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