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

**Subject - Unix Operating System**

**Name – Hemant Sharma**

**PRN – 22610001 Class – TYIT**

**Assignment No – 4(h)**

**Title-** Write program using semaphore to ensure that function f1 should executed after f2. (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:**

In Java, **semaphores** are used to control access to a shared resource by multiple threads. A semaphore maintains a set of permits, and threads can acquire or release these permits to regulate their access.

To ensure that **function f1** executes after **function f2**, we can use a **binary semaphore** (with a permit count of 1) to signal between the two functions. f2 will release the semaphore, and f1 will wait for the semaphore permit to be released before it starts executing. This guarantees the correct order of execution.

**Program:**

import java.util.concurrent.Semaphore;

class OrderedExecution {

    private final Semaphore semaphore;

    // Constructor: Initialize semaphore with 0 permits (f1() must wait)

    public OrderedExecution() {

        this.semaphore = new Semaphore(0);

    }

    // Function f1 (Executes AFTER f2)

    public void f1() {

        try {

            semaphore.acquire(); // Wait until f2() releases the permit

            System.out.println("Function f1 executed.");

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

    }

    // Function f2 (Executes FIRST and allows f1 to proceed)

    public void f2() {

        System.out.println("Function f2 executed.");

        semaphore.release(); // Unlock f1()

    }

}

// Thread for f1

class F1Thread extends Thread {

    private final OrderedExecution orderedExecution;

    public F1Thread(OrderedExecution orderedExecution) {

        this.orderedExecution = orderedExecution;

    }

    public void run() {

        orderedExecution.f1();

    }

}

// Thread for f2

class F2Thread extends Thread {

    private final OrderedExecution orderedExecution;

    public F2Thread(OrderedExecution orderedExecution) {

        this.orderedExecution = orderedExecution;

    }

    public void run() {

        orderedExecution.f2();

    }

}

// Main class

public class SemaphoreExecutionOrder {

    public static void main(String[] args) {

        OrderedExecution execution = new OrderedExecution();

        // Creating threads

        F1Thread thread1 = new F1Thread(execution);

        F2Thread thread2 = new F2Thread(execution);

        // Start threads (f1() starts first but waits)

        thread1.start();

        thread2.start();

        // Ensure proper execution order

        try {

            thread1.join();

            thread2.join();

        } catch (InterruptedException e) {

            e.printStackTrace();

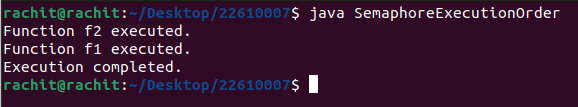
        }

        System.out.println("Execution completed.");

    }

}

**Output:**

****

**Conclusion:**Using semaphores for synchronization in Java allows us to control the order in which functions execute. By utilizing a binary semaphore, we ensure that f1 waits for f2 to complete before proceeding, demonstrating how semaphores can manage dependencies between threads