**Name:** Tejal Chavan.

**Class:** SECOMPs

**Batch:** E

**Roll No.:** 67

**Mini Project**

**Problem Statement:**

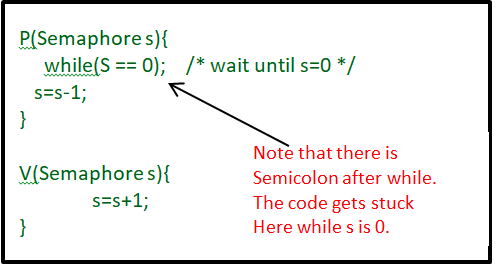
Teaching Assistant (TA) that holds office hours where they provide help to students with their programming assignments. The TA's office only has room for one desk with a chair and a computer. There are three chairs outside the office where students may sit and wait if the TA is currently helping another student. If there are no available chairs in the waiting area, the student shows up at a later time and go for the programming. When the TA has finished helping a student. He takes the next student and begins to help them. If there are no waiting students, he returns to his chair in the office and takes a nap. If a student shows up and sees the TA sleeping, they sit in his chair and wake him up.

Using threads and semaphores, implement a solution in JAVA that coordinates the activities of the TA and the students.

**Theory:**

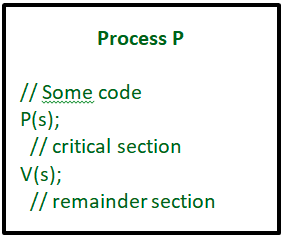
**Semaphore** is a simply a variable. This variable is used to solve critical section problem and to achieve process synchronization in the multi processing environment. The two most common kinds of semaphores are Counting Semaphores and Binary Semaphores. Binary semaphore can take the value 0 & 1 only. Counting semaphore can take nonnegative integer values.

Two standard operations, wait and signal are defined on the semaphore. Entry to the critical section is controlled by the wait operation and exit from a critical region is taken care by signal operation. The wait, signal operations are also called P and V operations. The manipulation of semaphore (S) takes place as following:



The wait command P(S) decrements the semaphore value by 1. If the resulting value becomes negative then P command is delayed until the condition is satisfied.

The V(S) i.e. signals operation increments the semaphore value by 1.



Mutual exclusion on the semaphore is enforced within P(S) and V(S). If a number of processes attempt P(S) simultaneously, only one process will be allowed to proceed & the other processes will be waiting. The semaphore operation are implemented as operating system services and so wait and signal are atomic in nature i.e. once started, execution of these operations cannot be interrupted. Thus semaphore is a simple yet powerful mechanism to ensure mutual exclusion among concurrent processes.

**Example:**

we have semaphore s, and two processes, P1 and P2 that want to enter their critical sections at the same time. P1 first calls wait(s). The value of s is decremented to 0 and P1 enters its critical section. While P1 is in its critical section, P2 calls wait(s), but because the value of s is zero, it must wait until P1 finishes its critical section and executes signal(s). When P1 calls signal, the value of s is incremented to 1, and P2 can then proceed to execute in its critical section (after decrementing the semaphore again). Mutual exclusion is achieved because only one process can be in its critical section at any time.

**Flow of Project:**

1) Create n students, each will run as a separate thread.

2) Student threads will alternate between programming for a period of time and seeking help from the TA.

3) If the TA is available, students will obtain help

4) If the TA is not available, students will sit in a chair outside office

5) If no chairs are available, students will resume programming and seek help at a later time

6) If the TA is sleeping, the student will notify TA with a semaphore

7) Use a semaphore to indicate a student in help mode

8) When a TA finishes helping a student, the TA must check to see if there are students waiting for help outside office.

9) The TA must help the students waiting outside office in the order that they started waiting.

10) If no students are waiting, then the TA can resume napping.

The best option for simulating students programming as well as the TA providing help to a student is to have appropriate threads sleep for a random period of time.

Methods of semaphore used in program-

boolean tryAcquire() : This method acquires a permit, if one is available and returns immediately, with the value true, reducing the number of available permits by one. If no permit is available then this method will return immediately with the value false.

int availablePermits() : This method returns the current number of permits available in this semaphore.This method is typically used for debugging and testing purposes.

**Conclusion:**

We studied about Semaphore,which is integer variable to achieve synchronization for two or more process having comman shared memory area using two methods called P() and V() i.e. wait() and signal() which achieves mutual exclusion and bounded waiting.