faculty of computers & Artificial intelligence

Operating system2

Project documentation

1) Project description:

- **Project no:1**
- **Project name: The Sleeping Teaching Assistant**

In a university computer science department, there's a Teaching Assistant (TA) who assists students with programming assignments during office hours. The TA's office is small and can only accommodate one student at a time. There are three chairs in the hallway for students to wait if the TA is busy.

When there are no students, the TA takes a nap at the desk. If a student comes in and finds the TA sleeping, they need to wake the TA up to get help. If the TA is already assisting another student, the new student waits on one of the hallway chairs. If all chairs are taken, the waiting student comes back later.

---

2) what you have actually did

we make 4 classes in addition to a gui class

1) TeachingAssistant.java:that has two semaphores,one to lock the TA,the other is to lock the student,making two integers that shows how many TAsleeping,and how many TAwakingup,An integer to the number of TAs An constructor taking an arguments(TAlock,StudentLock,num of TA),fuctions (Get TAsleeping), (Get TAwakingup) returns their outputs to the gui class,RUN function,TAlock acquire ,this let the TA to get the CS,once the TA take the lock the number of wakingup TAs increamented by 1 and the number of sleeping TAs decreamented by 1,
Help student Function to let the TA helping student for 5 sec.

2) office.java

this class manages the queuing system for students seeking help from TAs. It ensures synchronization using semaphores and locks, controls access to shared resources, and provides a mechanism for handling late-arriving

students. The class is designed to simulate a scenario where students and TAs interact in an office environment.

3) student.java

this class represents the behavior of a student in the simulation. The student waits for a random duration before entering the office (walkIntoOffice). The waiting time is simulated using thread sleeps, and the arrival is printed to the console. The Student class collaborates with the Office class, where the actual interaction with teaching assistants is handled.

4) osp.java

this class represents a simplified simulation of an office hours scenario, where students seek help from TAs in a controlled environment. Semaphores are used for synchronization, and threads are employed to model the behavior of TAs and students.

---

3)Team Members and Roles:

TeachingAssistant: Mohamed salah &Ahmed Mahmoud

Both Ahmed and Mohamed were commissioned to create the TeachingAssistant class and modeling TA behavior ,making synchronization between student and TAs , Part of the gui class has been worked on .

Students:Ayaat ahmed&Alaa atef

Both ayaat and alaa were commissioned to create the student class And handeling student behaviour within the simulation and making the logic code for the waiting students , Part of the gui class has been worked on .

Office:Alaa magdy&Enjy mousa

Both alaa and enjy were commissioned to create the office class with handling the office status(full and not full) and manage the number of accept students,managing QueueCount, Part of the gui class has been worked on .

Osp :this simulation class all team members shared with implementing it by their Knowledge for their classes**.**

-this documentation descriptions is made by Ayaat ahmed &Alaa atef

---

4) code documentation:

- **for student class**:

1) Constructor (Student):
* Takes an Office object as a parameter during instantiation, indicating the office environment where the student will seek help.

2) Method (run - from the Runnable interface):
* Entry point for the student's behavior when a thread is started. Calls the waitForSomeTime method and then walkIntoOffice.

3)Method (walkIntoOffice):
* Invokes the acceptStudent method of the associated Office object, indicating that the student is walking into the office and needs assistance.

4)Method (waitForSomeTime):
* Generates a random duration (up to 10 seconds) for the student to wait before entering the office.
* Sleeps the thread for a fixed duration (2 seconds) plus the random duration, simulating the time it takes for a student to arrive and wait.
* Prints a message indicating the student's arrival after the waiting period.

- **For osp class:**

1) Simulation Parameters:

- The class OSP has static variables (noOfStudents, Studentthreads, nofTAS, TASthreads, nofChairs) representing the number of students, an array to store student threads, the

number of TAs, an array to store TA threads, and the number of chairs in the office.
2) Constructor (OSP):

- The constructor allows setting initial values for the number of students, TAs, and chairs. However, the constructor is not utilized in the code.
3) Method (run):

- The run method is the main method that orchestrates the simulation.
4) Semaphore Initialization:

- Two semaphores (TALock and StudentLock) are initialized with zero initial permits. Semaphores are used to control access to shared resources, in this case, the office and TAs.
5) Office Initialization:

- An Office object is created, representing the office environment. The Office class is assumed to take semaphores and chair capacity as parameters.
6) TeachingAssistant Initialization:

- A TeachingAssistant object is created with semaphores, and static information from the TeachingAssistant class is retrieved (e.g., sleeping and walking-up states).
7) Creating TA Threads:

- A loop creates threads for each TA, assigns a name, and adds them to the TASthreads array. The TeachingAssistant instance is shared among all TA threads.
8) Starting TA Threads:

- Another loop starts all the TA threads concurrently.
9) Creating Student Threads:

- A loop creates threads for each student, assigns a name, and adds them to the Studentthreads array. Each student is associated with the same Office instance.
10) Starting Student Threads:

- Another loop starts all the student threads concurrently.
11) Waiting for Student Threads to Complete:

- The program waits for all student threads to finish their execution before proceeding to the next steps.
12) Interrupting TA Threads:

- (Commented out in the code) There is a commented-out line that suggests interrupting the TA thread (TAthread.interrupt()). It's currently disabled.
13) Waiting for TA Threads to Complete:

- The program waits for all TA threads to finish their execution before concluding the simulation.

- **For office class:**

1) Class Variables:

- TAChair: A semaphore representing the available chairs where TAs can meet with students.

- TALock: A semaphore used to control access to the TA resource (e.g., ensuring only one student meets with a TA at a time).

- studentLock: A semaphore used for synchronization during student meetings with TAs.

- studentCount: An AtomicInteger to keep track of the number of students currently in the office.

- noOfWaitingChar: An integer representing the maximum number of students allowed to wait in the office.

- lock: A ReentrantLock used for synchronization in the acceptStudent method.

- LateStudents: A static variable counting the number of students who arrived late.

- queueCount: A static variable representing the current number of students in the queue.

- arr: An array that is declared but not used in the provided code.

2) Constructor (Office):

- Initializes the Office object with semaphores, the maximum number of waiting chairs, and the number of TAs. The initial values are provided as parameters during object creation.

3) Method (acceptStudent):

- Controls the process of a student entering the office, waiting, meeting with a TA, and completing the meeting.

- Uses semaphores and locks to ensure proper synchronization and to avoid race conditions.

- Checks if the office is full (maximum number of waiting students reached), and if so, prints a message indicating that the student needs to come later after waiting for 5 seconds.

- Acquires a chair (TAChair semaphore) and increments/decrements the student count accordingly.

- Prints messages indicating the number of students in the queue, students coming later, and the student having a meeting with a TA.

- Releases semaphores after the meeting.

4) Helper Method (isOfficeFull):

- Checks if the office is full, i.e., the maximum number of waiting students has been reached.

5) Static Method (getQueueCount):

- Returns the static variable queueCount, representing the current number of students in the queue.

- **For TeachingAssistant class:**

1) Class Variables:

- TALock: Semaphore for controlling access to the TA resource, ensuring that only one student can meet with a TA at a time.
- studentLock: Semaphore for synchronization during the TA-student meeting.
- TASLeeping: Static variable representing the number of sleeping TAs (initialized with the total number of TAs).
- TAWalkingUp: Static variable representing the number of TAs currently waking up.
- numTA: Added variable to store the total number of TAs.

2) Constructor (TeachingAssistant):

- Initializes the TA instance with the provided semaphores and the total number of TAs.
- Sets the initial value of TASLeeping to the total number of TAs.

3) Static Methods (getTASLeeping and getTAWalkingUp):

- Provide access to the TASLeeping and TAWalkingUp static variables.

4) Method (run):

- Represents the main behavior of TAs in a continuous loop.
- Acquires the TALock semaphore, indicating that a student is waiting for assistance.
- Prints messages indicating the TA waking up, decrementing TASLeeping and incrementing TAWalkingUp.

- Calls the HelpStudents method, simulating the TA helping students by sleeping for 5 seconds.
- Releases the studentLock semaphore, indicating the end of the TA-student meeting.
- Prints messages indicating the TA going to sleep, incrementing TASLeeping and decrementing TAWalkingUp.
- Checks for termination conditions: If all TAs have finished working and are sleeping, terminate the simulation.

5) Method (HelpStudents):

- Simulates the TA helping students by making the thread sleep for 5 seconds.

Project Implementation: Tools and Technologies Used:

Programming Language: Java

Concurrency Handling: Java Threads, Mutex Locks, Semaphores

GUI Reference

Inputs		Output	
#TAs	2	#TAs working	2
#Chairs	3	#TAs Sleeping	0
#Students	20	#Students Waiting on chairs	3
		#Students that will come later	15