



Operating Systems 2 (Fall 2023)
Project Discussion



Project No. 1

THE SLEEPING TEACHING ASSISTANT

ID	Name
202000904	مصطفى عادل مصطفى ابو العز
201900990	يوسف علي شتا
201900041	احمد سيد فتحي عبدالوهاب
202000027	احمد حسن عبدالجليل الصغير
202000019	احمد تامر السيد السيد احمد الحلواني
201900207	ايمن انور عبدالمنعم
20180012	ابراهيم صفوت عبدالله محمد

DOCUMENTATION CONTENT :

- 1) Project Description-----pg.3
- 2) What has been done-----pg.3
- 3) Code Documentation -----pg.4
 - `TeachingAssistant` class-----pg.4
 - `Student` class -----pg.4
 - `SignalController` class -----pg.5
 - `sleeping_gui` class -----pg.5
- 4) The main components of the project -----pg.6
- 5) What is the purpose of the Semaphore in the project? ----- pg.7
- 6) Student Class ----- pg.8
- 7) TeachingAssistant class ----- pg.10
- 8) SIGNAL CONTROLLER CLASS ----- pg.12
- 9) SLEEPING GUI CLASS ----- PG.13
- 10) Graphical user interface ----- pg.15
- 11) Team Members Role ----- pg.17



Operating Systems 2 (Fall 2023) Project Discussion



PROJECT DESCRIPTION

The project is a simulation of a scenario where multiple students seek help from a teaching assistant (TA). The TA can only help one student at a time, and there are a limited number of chairs outside the TA's office where students can wait. The project uses multithreading and semaphores to manage the interactions between students and the TA.

WHAT HAS BEEN DONE

The project is implemented in Java and uses the `java.util.concurrent.Semaphore` class for synchronization. The main classes in the project are `Student`, `TeachingAssistant`, and `SignalController`

1. `TeachingAssistant`: The `TeachingAssistant` class represents a TA who alternates between helping students and sleeping. The TA waits for a signal from a student to wake up. When they receive a signal, they work with the student for a certain amount of time. If there are other students waiting in chairs, they then work with those students. This class implements the `Runnable` interface and represents a teaching assistant.
2. `Student`: The `Student` class represents a student who alternates between programming and seeking help from the TA. When a student needs help, they check if the TA is available. If the TA is available, the student wakes up the TA and works with them for a certain amount of time. If the TA is not available, the student checks if there are any chairs available. If a chair is available, the student waits for the TA to become available. If no chairs are available, the student goes back to programming.
3. `SignalController`: The `SignalController` class is a custom class that is used to coordinate between the `Student` and `TeachingAssistant` threads. The `sendSignal ()` method is used to send a signal, and the `waitForSignal ()` method is used to wait for a signal.
4. `sleeping_gui`: This class provides a graphical user interface for the simulation. It allows the user to specify the number of TAs, students, and chairs, and then run the simulation.

CODE DOCUMENTATION

- `TeachingAssistant` class

- `SignalController signalTrigger`: A semaphore used to wake up the TA.
- `Semaphore chairs`: A semaphore representing the chairs where students wait.
- `Semaphore TeacherAvailable`: A semaphore used to check if the TA is available.
- `Thread t`: A reference to the current thread.
- `int numberOfTeacher`: The number of the current TA.
- `int numberOfchairs`: The number of chairs available for waiting students.
- `run()`: This method is the main logic of the TeachingAssistant thread. It is an infinite loop where the TA alternates between helping students and sleeping.

- `Student` class

- `int waitToAsk`: The time a student programs before asking for help.
- `int studentNum`: The number of the student.
- `SignalController signalTrigger`: A semaphore used to wake up the TA.
- `Semaphore chairs`: A semaphore representing the chairs where students wait.
- `Semaphore TeacherAvailable`: A semaphore used to check if the TA is available.
- `Thread t`: A reference to the current thread.
- `run()`: This method is the main logic of the Student thread. It is an infinite loop where the student alternates between programming and seeking help from the TA.



Operating Systems 2 (Fall 2023)
Project Discussion



- `SignalController` class

- `boolean signal`: A boolean value used to signal the TA to wake up.
- `sendSignal ()`: This method is used to send a signal. It sets the signal field to true and then calls notify().
- `waitForSignal ()`: This method is used to wait for a signal. It enters a loop that continues until signal is true. Inside the loop, it calls wait().

- `sleeping_gui` class

- `int numberOfStudents`: The number of students in the simulation.
- `int numberOfTA`: The number of TAs in the simulation.
- `int numberOfchairs`: The number of chairs available for waiting students.
- `RunActionPerformed()`: This method is called when the "RUN" button is clicked. It reads the number of TAs, students, and chairs from the text fields in the GUI, creates the semaphores and the SignalController, and starts the Student and TeachingAssistant threads.



Operating Systems 2 (Fall 2023) Project Discussion



THE MAIN COMPONENTS OF THE PROJECT ARE:

1. 'SignalController' Class: This class is responsible for controlling signals between threads. It has two methods: 'sendSignal' and 'waitForSignal'. 'sendSignal' is used to send or trigger a signal, and 'waitForSignal' makes the thread wait until it receives a signal.
2. 'TeachingAssistant' Class: This class represents a teaching assistant (TA). It implements the 'Runnable' interface and has a 'run' method that defines the behavior of the TA when they are awake and when they are sleeping.
3. 'Student' Class: This class represents a student. It also implements the 'Runnable' interface. The 'run' method defines the behavior of the student when they are programming and when they are asking for help.
4. 'sleeping_gui' Class: This class is responsible for the graphical user interface of the application. It allows the user to input the number of TAs, students, and chairs, and then run the simulation.
5. Semaphores: These are used to manage the chairs outside the office and to determine if the TA is available. They are key to synchronizing the interaction between the TA and the students.



Operating Systems 2 (Fall 2023) Project Discussion



WHAT IS THE PURPOSE OF THE SEMAPHORE IN THE PROJECT?

The 'Semaphore' class in the project is used as a mechanism for controlling access to shared resources by multiple threads. In this case, the shared resources are the Teaching Assistant's (TA's) time and the chairs outside the TA's office.

There are three 'Semaphore' instances used in the project:

1. 'TeacherAvailable': This semaphore represents the availability of the TA. If a permit is available from this semaphore, it means the TA is available to help a student. If not, it means the TA is currently busy.
2. 'chairs': This semaphore represents the chairs outside the TA's office. The number of permits in this semaphore is equal to the number of chairs. If a student finds that the TA is busy and a permit is available from this semaphore, they can wait in a chair until the TA is available.
3. 'available': This semaphore is used in the 'sleeping_gui' class to control the number of TAs that are created and started when the "RUN" button is clicked.

the 'Semaphore' class is used in this project to control access to shared resources in a multithreaded environment, ensuring that threads do not interfere with each other when accessing these resources.

STUDENT CLASS

The Student class in the provided Java code represents a student in a scenario where students are programming and occasionally need help from a Teaching Assistant (TA). The class implements the Runnable interface, which means instances of this class can be executed as threads. The class has several instance variables, including a SignalController object, two Semaphore objects, a Thread object, and two integers. The SignalController object, wakeup, is used to wake up the TA. The Semaphore objects, chairs and TeacherAvailable, are used to manage the chairs outside the office and to determine if the TA is available, respectively. The Thread object, t, represents the current thread, and the integers, waitToAsk and studentNum, represent the time a student waits to ask for help and the student number, respectively.

// Time to program before asking for help (in seconds).

`private int waitToAsk;`

// Student number.

`private int studentNum;`

// Semaphore used to wakeup TA.

`private SignalController wakeup;`

// Semaphore used to wait in chairs outside office.

`private Semaphore chairs;`

// Mutex lock (binary semaphore) used to determine if TA is available.

`private Semaphore TeacherAvailable;`

// A reference to the current thread.

`private Thread t;`

The run method is the main method that is executed when the thread for the Student object is started. The method contains a while loop that continues until the thread is interrupted. Inside the loop, the student first programs for a certain amount of time, then checks if the TA is available. If the TA is available, the student wakes up the TA and starts working with the TA. If the TA is not available, the student checks if any chairs are available. If a chair is available, the student sits and waits for the TA. If no chairs are available, the student goes back to programming.

- The method starts with an infinite loop that continues until the thread is interrupt

`while(!Thread.currentThread().isInterrupted())`

- Inside the loop, the student first programs for a certain amount of time, which is simulated by making the thread sleep for waitToAsk seconds:

// Program first.

`System.out.println("Student " + studentNum + " has started programming for " + waitToAsk + " seconds.");`

`t.sleep(waitToAsk * 1000);`



Operating Systems 2 (Fall 2023)
Project Discussion



- After programming, the student checks if the TA is available by trying to acquire the TeacherAvailable semaphore:

```
System.out.println("Student " + studentNum + " is checking to see if TA is available.");  
if (TeacherAvailable.tryAcquire())
```

- If the TA is available (the tryAcquire method returns true), the student wakes up the TA by calling the sendSignal method of the wakeup object, and then starts working with the TA:

```
wakeup.sendSignal();  
System.out.println("Student " + studentNum + " has woke up the TA. ");  
System.out.println("Student " + studentNum + " has started working with the TA. ");  
t.sleep(5000);  
System.out.println("Student " + studentNum + " has stopped working with the TA. ");
```

- If the TA is not available (the tryAcquire method returns false), the student checks if any chairs are available by trying to acquire the chairs semaphore:

```
System.out.println("Student " + studentNum + " could not see the TA. Checking for available chairs.");  
if (chairs.tryAcquire())
```

- In case of an InterruptedException, the method catches the exception, interrupts the current thread, and breaks the loop:

```
catch (InterruptedException e)  
{  
    Thread.currentThread().interrupt();  
    System.err.println("Student thread interrupted: " + e.getMessage());  
  
    break;  
}
```

he run method in the Student class represents the behavior of a student who alternates between programming and asking for help from the TA, waiting for the TA if necessary.



TEACHING ASSISTANT CLASS

The TeachingAssistant class in the provided Java code represents a teaching assistant (TA) in a scenario where students are programming and occasionally need help from a TA. The class implements the Runnable interface, which means instances of this class can be executed as threads. The class has several instance variables, including a SignalController object, two Semaphore objects, a Thread object, and two integers. The SignalController object, signalTrigger, is used to wake up the TA. The Semaphore objects, chairs and TeacherAvailable, are used to manage the chairs outside the office and to determine if the TA is available, respectively. The Thread object, t, represents the current thread, and the integers, numberOfTeacher and numberOfchairs, represent the number of the TA and the number of chairs, respectively.

// Semaphore used to wakeup TA.

```
private SignalController signalTrigger;
```

// Semaphore used to wait in chairs outside office.

```
private Semaphore chairs;
```

// Mutex lock (binary semaphore) used to determine if TA is available.

```
private Semaphore TeacherAvailable;
```

// A reference to the current thread.

```
private Thread t;
```

```
private int numberOfTeacher;
```

```
private int numberOfchairs;
```

- The run method is the main method that is executed when the thread for the TeachingAssistant object is started. The method contains a while loop that continues until the thread is interrupted. Inside the loop, the TA goes to sleep if there are no students left.

```
System.out.println("No students left. The TA "+numberOfTeacher+ " is going to nap.");
```

- The TA waits for a signal from a student. If a student arrives and finds the TA sleeping, the student wakes up the TA

```
signalTrigger.waitForSignal();
```

```
System.out.println("The TA "+numberOfTeacher+ " was awake by a student.");
```

- The TA then helps the students who are waiting. The TA continues to help students until there are no more students waiting in the chairs.

```
int permitsAcquired = numberOfchairs - chairs.availablePermits();
```

```
while (permitsAcquired > 0) {
```

```
    t.sleep(5000);
```

```
    if (chairs.availablePermits() < numberOfchairs) {
```

```
        chairs.release();
```

```
    permitsAcquired--;
```

```
    }
```

```
}
```

- If the thread is interrupted, the method catches the InterruptedException, interrupts the current thread, and breaks the loop.

`catch (InterruptedException e)`

```
{  
    Thread.currentThread().interrupt();  
    System.err.println("TeachingAssistant thread interrupted: " + e.getMessage());  
    break;  
}
```

- the TeachingAssistant class represents a TA's behavior in the scenario, including sleeping, waiting for a student, and helping students. The use of the SignalController and Semaphore objects allows the TA to synchronize its interaction with the students and the chairs.



SIGNAL CONTROLLER CLASS

The SignalController class in the code is used to manage signals between threads. It has two methods: sendSignal and waitForSignal.

- The sendSignal method is used to send or trigger a signal. It sets the signal field to true and then calls the notify method, which wakes up a single thread that is waiting on this object's monitor.

```
public synchronized void sendSignal() {  
    this.signal = true;  
    this.notify();  
}
```

- The waitForSignal method is used to make a thread wait until it receives a signal. It enters a loop that continues until the signal field is true. Inside the loop, it calls the wait method, which causes the current thread to wait until another thread invokes the notify method for this object. Once the signal is received (i.e., the signal field is true), it sets the signal field back to false.

```
public synchronized void waitForSignal() throws InterruptedException {  
    while(!this.signal) wait();  
    this.signal = false;  
}
```

In the TeachingAssistant and Student classes, the SignalController is used to synchronize the interaction between the TA and the students. When a student needs help from the TA, it uses the SignalController to wake up the TA. The TA, on the other hand, uses the SignalController to wait for a signal from a student.

SLEEPING GUI CLASS

The `sleeping_gui` class in the provided Java code is the main class of a desktop application that simulates a scenario where a teaching assistant (TA) helps students with their queries. This class extends `javax.swing.JFrame`, which means it represents a window in a desktop application. The class has several instance variables that represent the number of students, TAs, and chairs. These variables are used to configure the simulation based on user input.

```
int numberOfStudents;  
int numberOfTA;  
int numberOfchairs;
```

The `initComponents` method is called from the constructor to initialize the form. This method sets up the layout of the components in the window and configures their properties.

```
public sleeping_gui() {  
    initComponents();  
}
```

The `RunActionPerformed` method is an event handler that is called when the "RUN" button is clicked. This method reads the input from the text fields, creates a `SignalController` and several `Semaphore` objects, and starts the threads for the TAs and the students. It also starts a thread that updates the text fields with the current state of the simulation.

- At the beginning of the method, it reads the input from the text fields to get the number of TAs, students, and chairs:

```
numberOfTA = Integer.parseInt(taT.getText());  
numberOfStudents = Integer.parseInt(studentT.getText());  
numberOfchairs = Integer.parseInt(chairT.getText());
```

- Then, it creates a `SignalController` object and two `Semaphore` objects. The `SignalController` object is used to wake up the TA. The `Semaphore` objects are used to manage the chairs outside the office and to determine if the TA is available:

```
SignalController wakeup = new SignalController();  
Semaphore chairs = new Semaphore(numberofchairs,true);  
Semaphore available = new Semaphore(numberofTA,true);
```

- Next, it creates and starts a new thread for each student and TA. The `Student` and `TeachingAssistant` classes implement the `Runnable` interface, so they can be passed to the `Thread` constructor:

```
for (int i = 0; i < numberOfStudents; i++) {  
    Thread student = new Thread(new Student(studentWait.nextInt(20), wakeup, chairs, available, i + 1));  
    student.start();  
}
```

```

for (int i = 0; i < numberOfTA; i++) {

    // Create and start TA Thread.
    Thread ta = new Thread(new TeachingAssistant(wakeup, chairs, available, i + 1, numberOfchairs));
    ta.start();
}

```

- Finally, it starts a new thread that updates the text fields in the GUI with the current state of the simulation every 2 seconds. The SwingUtilities.invokeLater method is used to ensure that the GUI updates are performed on the Event Dispatch Thread, as required by Swing:

```

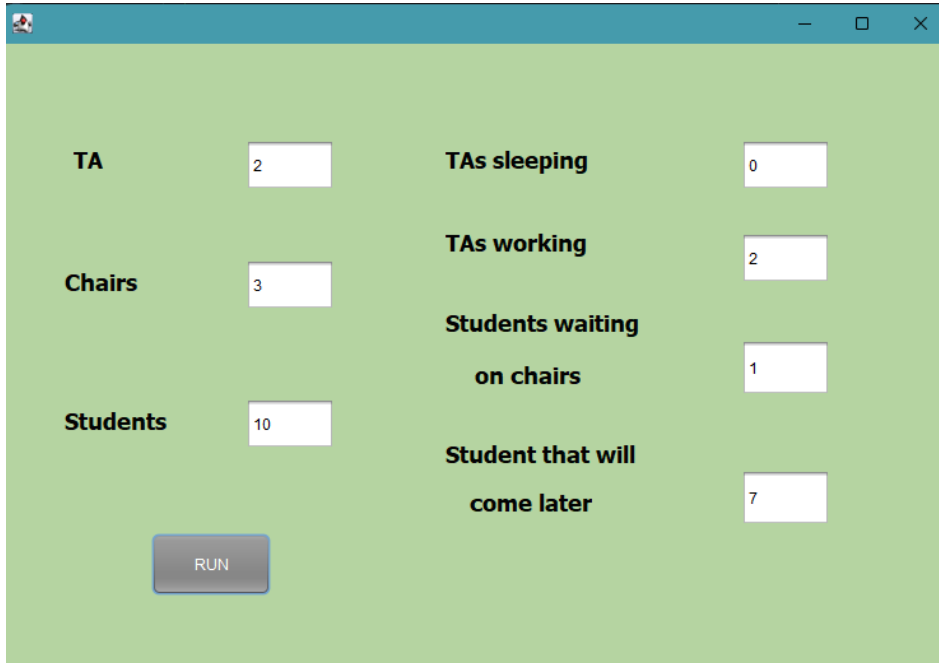
Thread print = new Thread(() -> {
    while (true) {
        try {
            Thread.sleep(2000);
            SwingUtilities.invokeLater(() -> {
                ta_workingT.setText(String.valueOf(numberofTA - available.availablePermits()));
                ta_sleepT.setText(String.valueOf(available.availablePermits()));
                student_watingT.setText(String.valueOf(numberofchairs - chairs.availablePermits()));
                student_laterT.setText(String.valueOf(numberofStudents - ((numberOfTA - available.availablePermits()) +
(numberofchairs - chairs.availablePermits()))));
            });
        } catch (InterruptedException ex) {
            break;
        }
    }
});
print.start();

```

the RunActionPerformed method in the sleeping_gui class is responsible for reading user input, starting the simulation, and updating the GUI to reflect the current state of the simulation.

the sleeping_gui class is responsible for creating and managing the graphical user interface of the application, receiving input from the user, starting the simulation based on the user input, and updating the user interface to reflect the current state of the simulation.

GRAPHICAL USER INTERFACE





THE GUI OF THIS PROJECT, AS DEFINED IN THE 'SLEEPING_GUI.JAVA' FILE, CONTAINS SEVERAL COMPONENTS. IN ADDITION TO THE COMPONENTS MENTIONED EARLIER, THE GUI ALSO INCLUDES:

1. 'JLABEL JLABEL₃', 'JLABEL JLABEL₄', 'JLABEL JLABEL₅', 'JLABEL JLABEL₆', 'JLABEL JLABEL₁₄', 'JLABEL JLABEL₁₆': THESE ARE JLABEL COMPONENTS USED TO DISPLAY TEXT LABELS ON THE GUI. THEIR SPECIFIC ROLES ARE NOT CLEAR FROM THE PROVIDED CODE.
2. 'JBUTTON RUN': THIS IS A BUTTON THAT THE USER CAN CLICK TO START THE SIMULATION. THE ACTION PERFORMED WHEN THIS BUTTON IS CLICKED IS DEFINED IN THE 'RUNACTIONPERFORMED()' METHOD.
3. 'JPANEL JPANEL₁': THIS IS THE MAIN PANEL THAT CONTAINS ALL OTHER GUI COMPONENTS.
4. 'JTEXTFIELD TAT', 'JTEXTFIELD STUDENTT', 'JTEXTFIELD CHAIRT', 'JTEXTFIELD TA_WORKINGT', 'JTEXTFIELD TA_SLEPT', 'JTEXTFIELD STUDENT_WATINGT', 'JTEXTFIELD STUDENT_LATERT': THESE

ARE JTEXTFIELD COMPONENTS USED TO GET INPUT FROM THE USER OR DISPLAY CERTAIN VALUES DURING THE SIMULATION.

THE LAYOUT OF THESE COMPONENTS IS DEFINED USING THE GROUPLAYOUT LAYOUT MANAGER IN THE 'INITCOMPONENTS()' METHOD.

	<p>Operating Systems 2 (Fall 2023) Project Discussion</p>	 كلية الحاسبات والذكاء الاصطناعي Faculty of Computers & Artificial Intelligence
---	---	--

TEAM MEMBERS ROLE

مصطفى عادل مصطفى ابو العز : Implemented the SignalController class and managed the synchronization logic.

احمد تامر السيد السيد احمد الحلواني && يوسف علي شتا : Developed the TeachingAssistant class, defining the behavior of TAs.

ابراهيم صفوت عبدالله محمد && احمد سيد فتحي عبدالوهاب : Created the Student class, defining the behavior of students.

ايمن انور عبدالمنعم && احمد حسن عبدالجليل الصغير : Developed the sleeping_gui class, designing and implementing the user interface of the application.

احمد تامر السيد السيد احمد الحلواني : Documentation