Project Title : Multiple Sleeping Barber Problem

ID	Name
202000531	عبدالرحمن محمد علي مصطفى
202000486	عائشة احمد فتحي محمد
202000939	منة الله اشرف عبدالحليم السيد
202000644	فاطمة ناصر عبدالغني
202000277	تغريد عبدالباسط عبدالعزيز
202001008	نوران ياسر مصطفى
202000394	سلمى محمد سيد احمد

project description

This problem is based on a hypothetical barbershop with k barbers. When there are no customers, all barbers sleep in their chairs. If any customer enters, he will wake up the barber and sit in the customer chair.If there are no chairs empty, they wait in the waiting queue.

What we have actually did

1. First, we create 2 classes : (Customer , Barber) which implements Runnable interface to implements run function
2. Then create class of (sleepingBR) which creates BR and customer threads and execute them.
3. we create class of (hall) which represent interactions between customer and barbers by using multithreading and synchronization techniques (ReentrantLock mutex, Semaphore (Available) , synchronized (function – blocks) and linkedlist(Customerlist, Backlater)).
4. Finally represent results on gui by creating thread runs function sleepingBR.start(form)

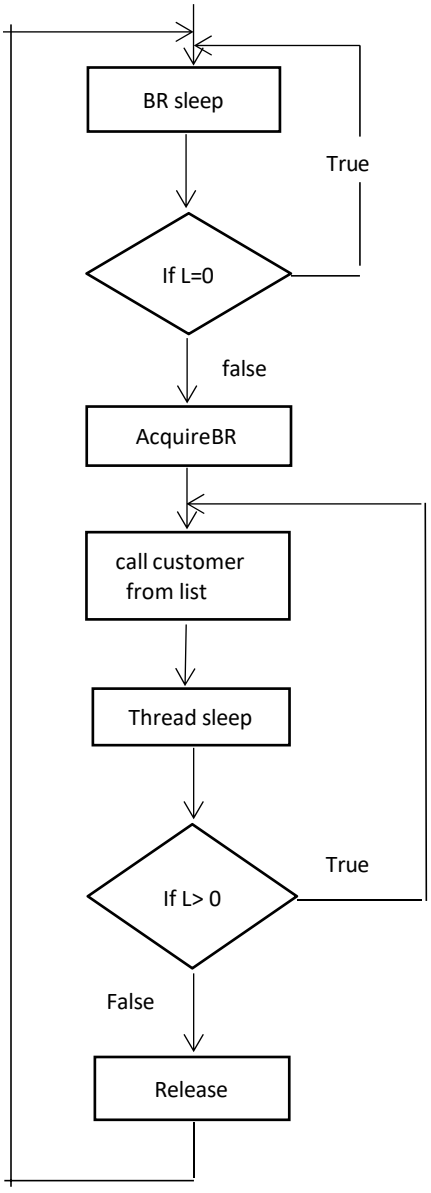
GitHub Link

https://github.com/NouranYasser/project.git

Code documentation.

1. Overall algorithm

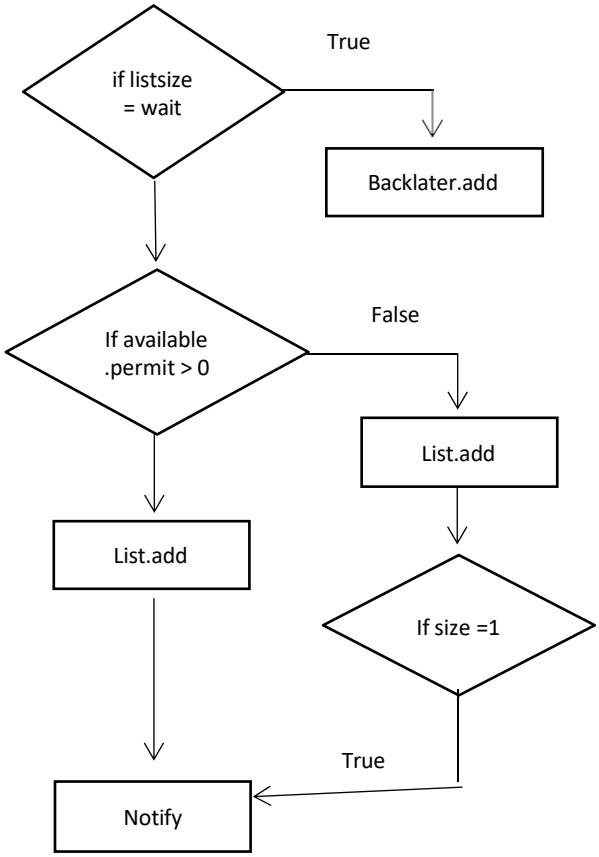
A. Barber



we made this class that implements Runnable interface for creating a new thread and then pass that instance to Thread constructor

We start as BR is sleeping in his shop then we check if the list of customers equal zero he will remain sleeping, if not equal zero then the BR semaphore acquired, and we call a customer from the list (the customer awake the BR) then when the BR finishes we check if the list is greater than zero we call another customer else the BR releases and go back to sleep

B. Customer



checks if the list size not equal waiting chairs then add the customer in the back later list, if equal then check again if the available semaphore permits greater than zero then add the customer to the list and notify else add customer to tail of list only but if the list was empty before adding customer then notify

2. Main functions

1. SleepingBR.Start()

- First Function takes parameters (**Session**)
- It creates an object **hall** from *Hall* class sending number of Brs, customers and waiting chairs and send gui form and creates an object **r** from *Random* class
- It creates BRs threads
- After that it creates a customer 's threads and sets ID to each customer and sets a start time to each customer randomly when enter to BR.
- If no customers back later , function will terminate else it runs customers in back later then terminate

2. Hall.AproveRequest

- it takes one parameter (BR_ID)
- first synchronized (customerLists) and put BR sleeping in loop breaks when customer added in list
- Get customer id which is waiting for BR to be empty and print customer id finds BR id available and ask the BR.
- Then check if customer list .size() equal waiting chairs and available try acquire() then make available acquire.
- Call BusyBR function to represent it on gui window
- After that create thread.sleep as delay to return time of each customer to ask and
- check if customer list size isn't empty print BR_ID calls a customer id to enter Hall after all
  end then make available release.

3. Hall.EnterHall

- It takes parameter customer which object of class customer.
- Print customer id tries to enter hall to aprove request at that time
- Check if customer list size is full and equal waiting chairs then print number of chairs available for customer id so customer leaves and will come back later ,then call CustomerBackLater.add(customer)
- Check if available permits > 0 then notify customer list
- customer call method take chair so gui represent customer takes a chair in the waiting room then check if customer list equal 1 then notify customer

4. Backlater

which type of List<customer> return customer back later list

Problem1 :Deadlock

There might be a scenario in which the customer ends up waiting on the barber and a barber waiting on the customer ,which would result a deadlock.

Scenario:

A customer may arrive to find the barber cutting hair so they return to the waiting room to take a seat but while walking back to the waiting room the barber finishes the haircut and goes to the waiting room, which he finds empty (because the customer walks slowly or went to the restroom) and thus goes to sleep in the barber chair.

Solution:

There are several possible solutions , all solutions require a mutex, which ensures that only one of the participants can change state at once. The barber must acquire the room status mutex before checking for customers and release it when they begin either to sleep or cut hair; a customer must acquire it before entering the shop and release it once they are sitting in a waiting room or barber chair, and also when they leave the shop because no seats were available. This would take care of both of the problems mentioned above. A number of semaphores is also required to indicate the state of the system. For example, one might store the number of people in the waiting room , second one to check status of the barber either idle or not , and third semaphore to keep count of the available seats , so customer either wait if there free seats or leave if there are none

---

Problem2 :starvation

Starvation might happen to customer who is waiting for a long time when the customer don't follow order or when barber calls out customer randomly

Solution:

To handle this problem in the code, insert the customers in a linked list which follows the first in first out property. So, every time a customer sits in a waiting room, they will be selected by the barber in first come first serve basis

Example description

A university computer science department has a teaching assistant (TA) who helps undergraduate students with their programming assignments during regular office hours. The TA 's office is rather small and has room for only one desk with a chair and computer. There are three chairs in the hallway outside the office where students can sit and wait if the TA is currently helping another student. When there are no students who need help during office hours, the TA sits at the desk and takes a nap. If a student arrives during office hours and finds the TA sleeping, the student must awaken the TA to ask for help. If a student arrives and finds the TA currently helping another student, the student sits on one of the chairs in the hallway and waits. If no chairs are available, the student will come back later.

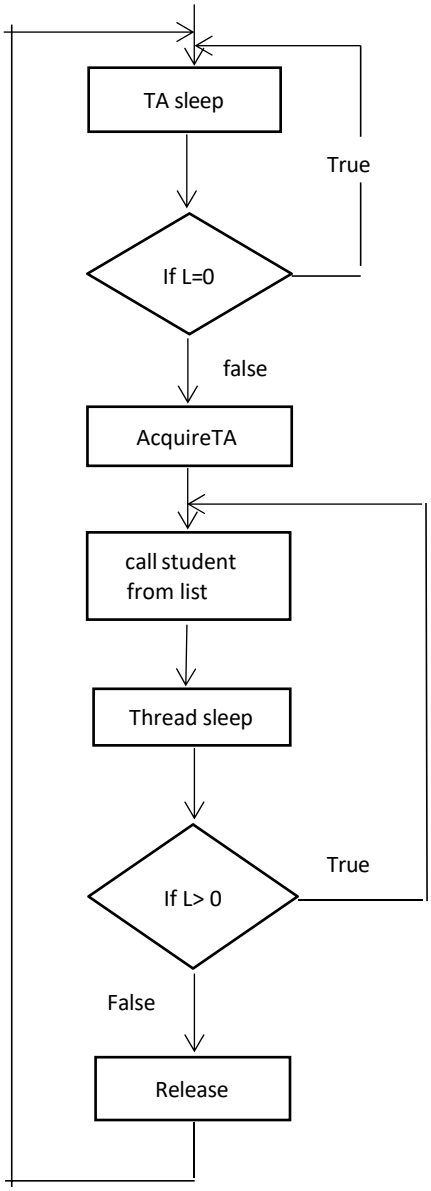
1. First, we create 2 classes : (Students , Teacher assistant) which implements Runnable interface to implements run function

2. Then create class of (sleepingTA) which creates TA and student threads and execute them.

3. we create class of (hall) which represent interactions between student and teaching assistants by using multithreading and synchronization techniques (ReentrantLock mutex, Semaphore (Available) , synchronized (function – blocks) and linkedlist(Studentlist, Backlater)).

4. Finally represent results on gui by creating thread runs function sleepingTA.start(form)

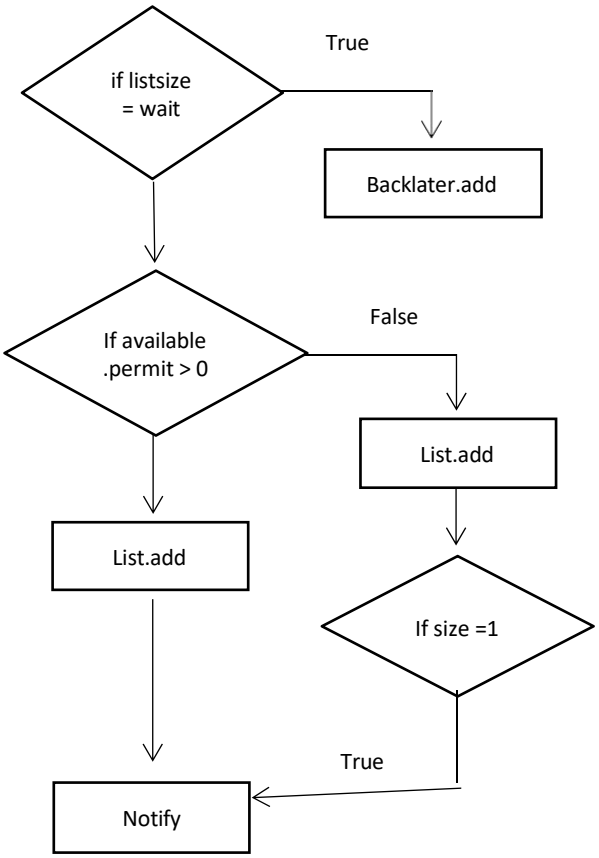
A. Teacher Assistant



we made this class that implements Runnable interface for creating a new thread and then pass that instance to Thread constructor

We start as TA is sleeping in his office then we check if the list of students equal zero he will remain sleeping, if not equal zero then the TA semaphore acquired, and we call a student from the list (the student awake the TA) then when the TA finishes we check if the list is greater than zero we call another student else the TA releases and go back to sleep

B. Student



checks if the list size not equal waiting chairs then add the student in the back later list, if equal then check again if the available semaphore permits greater than zero then add the student to the list and notify else add student to tail of list only but if the list was empty before adding student then notify

Main functions

1. SleepingTA.Start()
   - First Function takes parameters (**Session**)
   - It creates an object **hall** from *Hall* class sending number of Tas, student and waiting chairs and send gui form and creates an object **r** from *Random* class
   - It creates TAs threads
   - After that it creates a student's threads and sets ID to each Student and sets a start time to each student randomly when enter to TA.
   - If no students back later , function will terminate else it runs students in back later then terminate

2. Hall.AnswerQuestion
   - it takes one parameter (TA_ID)
   - first synchronized (studentLists) and put TA sleeping in loop breaks when student added in list
   - Get student id which is waiting for TA to be empty and print student id finds TA id available and ask the TA.
   - Then check if student list .size() equal waiting chairs and available try acquire() then make available acquire.
   - Call BusyTA function to represent it on gui window
   - After that create thread.sleep as delay to return time of each student to ask and
   - check if student list size isn't empty print TA_ID calls a student id to enter Hall after all end then make available release.

3. Hall.EnterHall
   - It takes parameter student which object of class student.
   - Print student id tries to enter hall to ask question at that time
   - Check if student list size is full and equal waiting chairs then print number of chairs available for student id so Student leaves and will come back later ,then call StudentBackLater.add(student)
   - Check if available permits > 0 then notify student list
   - Student call method take chair so gui represent student takes a chair in the waiting room then check if student list equal 1 then notify student

4. Backlater
   - which type of List<Student> return student back later list