Sleeping Barbers Problem

Name: Róisín Ní Bhriain
Student Number: 23269640
Email: roisin.nibhriain3@mail.dcu.ie
Program: MCM Secure Software Engineering
Module Code: CA670
Submission: Java Threads Assignment
Word Count: 632

Róisín Ní Bhriain - 23269640

I understand that the University regards breaches of academic integrity and plagiarism as grave and serious. I have read and understood the DCU Academic Integrity and Plagiarism Policy. I accept the penalties that may be imposed should I engage in practice or practices that breach this policy. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations, paraphrasing, discussion of ideas from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the sources cited are identified in the assignment references. Any use of generative AI or search will be described in a one page appendix including prompt queries.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work. By signing this form or by submitting this material online I confirm that this assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. By signing this form or by submitting material for assessment online I confirm that I have read and understood DCU Academic Integrity and Plagiarism Policy.

Signed Name: Róisín Ní Bhriain
Date: 28-02-2024

Program Design

The program uses three main characters in its design. Each of which I will explain in turn. Finally I will explain the main design and how it all functions together.

Barber

The barbers are a class implementing *Runnable*. They have an id and the sleeping barbers shop as class variables as well as the mean haircut time and the standard deviation. The run function prints that the barber is starting and has a continuous loop to continue calling the cut hair function in the shop.

Customer

The customers are also a class implementing *Runnable*. They have an id and the sleeping barbers shop as class variables. The run function prints that the customer is starting and attempts to add the customer to the waiting area in the shop by using the add customer function in the shop.

Customer Generator

The customer generator generates a thread for each new customer. The ID of the customer is decided by a count that begins when the customer generator thread is first started. The thread sleeps using a Gaussian generator from the Random class. This is multiplied by the standard deviation and the mean is added to it. The thread does this gaussian sleep until the next customer is ready to enter the shop.

Main Design

This has three class variables - the list of customers waiting, the number of barbers and the number of available waiting chairs. To ensure the concurrency works correctly my design uses synchronised statements in two functions: cutHair and addCust. I will discuss the functions more in depth below.

cutHair

This function has a synchronised statement using the arraylist of customers as a monitor. If the list is empty then the barber waits until there is a free customer. A customer is removed from the head of the list once the list is not empty. Then the length of time to cut the hair is calculated using the standard deviation and the mean to cut hair. Then the thread sleeps for that length of time. Before exiting the function.

addCust

The customer enters the shop. There is also a synchronised statement here with a monitor of the arraylist used to check if there are too many customers sitting in the waiting chairs. If the list size is larger than the max number of chairs then the customer leaves. Else the

customer is added to the list and the list is used to notify the barber that there is a customer waiting.

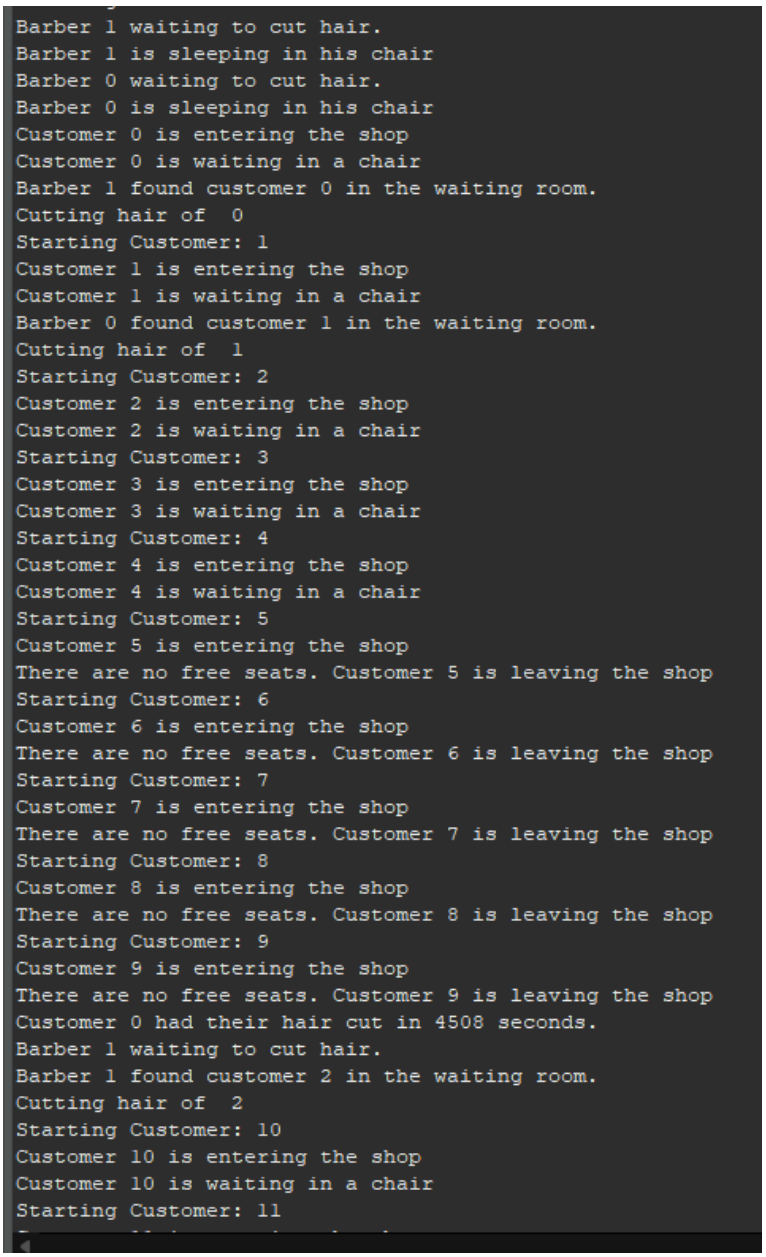
main

The main initialises the arraylist of customers - which will be empty. It initialises its own class (for use in the various threads). It prints the number of cores available and then starts a fixed thread pool based on the number of cores available. It also initialises a customer generator. The executor is then executed and a number of barber threads are created with consecutive IDs and each of these barbers are started. Then a thread with the customer generator is created and started. Finally the shutdown is initiated.

Correctness and Fairness Justifications

In terms of the customers each customer will get added to the queue if there is space and as it is a synchronised section only one customer can be added at a time. The barber takes the customer to cut hair from the head of the arraylist every time which means that there will be no customer who sits down in a waiting chair who will be left behind and missed.

The barbers section creates fairness with a synchronised section which waits until there are customers available to cut hair. Each barber must sleep for a certain amount of time after the synchronised statement which ensures that there will be no deadlock as eventually a barber will be able to choose a customer from the list.



Sleeping Barbers Problem In Practice

The solution to this problem is used in practice when doing interprocess communication or any other queuing problems in operating systems. Any processes which use shared memory to communicate will find the sleeping barbers problem useful. The customers here will be like the processes giving instructions and the barbers receiving instructions.