

CS431: Programming
Languages Lab

Assignment 1 Report

Shivang Dalal
170101086

8th Oct 2020

Q1: Sock Matching Robot

Introduction

The task is to simulate an automatic sock matching process. The system has three moving parts, the robotic arms which pick socks from a sock pile and give it to the matching machine. Second is the Matching machine, which matches the same colored socks into pairs, and finally, the shelf manager, which puts the socks onto the shelf. Since there are multiple Robotic arms, the primary synchronization concern occurs in the interaction between the sock piles and the robotic arms and the interaction between the arms and the matching machine.


Concurrency

Concurrency is achieved when multiple components perform their tasks simultaneously. In this scenario, concurrency's primary method is having multiple robot arms for quickly delivering socks to the matching machine. Secondly, the matching machine should be able to process pairs as soon as a valid pair can be formed and pass it on to the shelf manager. Also, multiple arms should be able to deliver socks to the Matching machine; hence there will be a need for a buffer for the arms to put the socks into.

How it is handled: Multiple threads independent from each other have been created for running the different robot arms. Similarly, the Machine machine has its own thread. These are different from the main thread, which initializes the execution of the program and waits for all of its children thread to complete.

Synchronization

Since we have multiple components accessing the same resource, there is a need for synchronization. The first point of Synchronization is the Sock Pile; this is needed to ensure that when multiple robot arms try to pick socks, they don't pick the same sock and also to maintain coherence. The second point of



Synchronization is the buffer for socks in the matching machine. Since the robot arms are agnostic of the color of the sock, hence there is only one common buffer for all robots to put their socks. Also, to simulate the real world, the buffer has been given a fixed size; hence robots can't indefinitely add socks to the buffer. Once it is full, they have to wait for the matching machine to match socks and start emptying the buffer before the arms can add more socks.

How is it Handled: The main synchronization technique used is monitors. The Sock pile, which has a class of its own, has an associated monitor that is used to synchronize access to the sock pile. It allows each robot arm to pick a sock out of the remaining socks randomly. A similar monitor is also used for synchronizing access to the Shelf by the shelf manager. The other technique used is Java's BlockingLinkedQueue for the Matching machine buffer, which is a thread-safe Queue. This serves both our purposes, first of a limited size buffer, which automatically blocks new additions when the limit is reached. Secondly, it guarantees synchronization and also provides a certain amount of concurrency.

Q2: Data Modification System:

Introduction

We have to simulate a Data modification system that is used by three teachers, CC, TA1, and TA2. CC has a higher priority than the two TAs. In order to simulate concurrency we take requests from updates from all teachers in batches and update them simultaneously. The major reason for following this principle is that trying to run multiple instances of the program and executing them in close enough succession to observe the effects of the coded synchronization and concurrency is difficult. Also, since the input is a text file instead of a database, there is no index and trying to search directly in the file for all the updates is a very inefficient and impractical solution. Hence the solution adopted is to gather a batch of updates and then read the entire data into memory once and execute all the queries. This can be easily extended to support multiple instances of the program running simultaneously with the inclusion of file locks.



Need for Concurrency

Since the different teachers may need to perform updates in parallel, for example, multiple vivas/evaluation happening at the same time, or multiple teachers evaluating the same student. This calls for concurrency in the system.

How is it handled: Once a batch of updates has been collected, each Teacher has a thread created for them which is fed with the updates scheduled by the respective teacher. (the distributed system has been simulated using a single program this way) And then, these threads are run concurrently while ensuring that CC completes its updates before the other threads. This is to ensure that if a Student record has an update scheduled by both a TA and the CC, then only the CC's update gets registered.


Shared resource

The shared resource, in this case, is the shared student info file (stud_info.txt). However, as stated earlier, we read that file in the form of a map of student records into the memory, which is indexed by the roll numbers. This memory representation of the file is also a shared resource for our program.

Synchronization

Synchronization is important since this ensures that only one teacher is able to edit a student's record at a time. If not implemented the outcome of the same commands may not remain deterministic, and depend on the execution order of the commands by the processor. A simple example of this can be that if synchronization is not implemented in this question, and two teachers TA1 and TA2, wish to increase and decrease a certain student's marks by 10, respectively. Then they may read the student's current marks simultaneously as 80. Hence TA1 will want to increase the marks to 90 and TA 2 will want to decrease it to 70. The final value of the marks will depend on which teacher's thread gets executed later.

E.g., if TA2 executes later, then the final value will be 70. However, ideally, the final value should remain 80 itself, since the increase and decrease cancel each



other out. Synchronization can guarantee this since, irrespective of which teacher gets access to the record first, the other teacher will see only the updated values and hence change the updated values further, resulting in the final correct value.

How is it handled: This has been implemented using semaphores; each student record has an associated semaphore (incorporated into the student class). Any thread that wishes to edit the record has to acquire the semaphore first. And it is released post-editing the record.

Q3: Calculator for Differently Abled Persons:

Introduction

The task is to design a calculator which can be operated by a disabled person who may only be able to press a limited number of buttons on the keyboard without much movement. The basic idea is to highlight different buttons on the calculator periodically, which can then be pressed using limited buttons on the keyboard. For the first scheme, only the Enter key is used for the first scheme, and all buttons on the calculator, including the Clear and Calculate button, are highlighted based on the provided scheme. For the second implementation, the Enter key is used to select highlighted number keys, and the Space bar is used to select the highlighted operator or aforementioned Clear and Calculate buttons.

Implementation

The choice for the control scheme is given at the start of the program. The highlighting is done by two threads, one for the number buttons and one for the remaining operators and function buttons. Locks have been used for the first control scheme to ensure only one of the two highlighting threads is active at any given time. An additional feature of making all the buttons respond to mouse clicks has been implemented, irrespective of the initial control scheme.