Documentation

I implemented a multi-threaded application for simulating a fun fair payment system in this project. Synchronization and the data consistency is taken into the consideration throughout the implementation because prepayments can be made simultaneously through different ticket vending machine instances.

You can run the program with "./simulation <input path>" command after executing the makefile. The output file name will be "<input path>_log.txt" after execution.

I implemented a class named Customer. Customer class have 6 public fields, which are id, sleepTime, machineId, companyName, paymentAmount and companyId. All of these variables are integers except companyName. CompanyName variable is a string.

I implemented a split function to make taking input easier. Split function takes three parameters. The first parameter is the string we want to split. The second parameter is character that we want to split from. The last parameter is the vector that we want to store the output. For example, if a user wants to split the sentence "I like ice cream" from its spaces and store the output inside of the "result" vector, the user should call the split function like split("I like ice cream", ' ', result). The content of the result vector after the execution of the split function is "I", "like", "ice" "cream".

The program starts with taking inputs from specified input file. Program reads the input file line by line. A customer object is created before reading each line. After reading the line, program splits the line from commas. When the line is splitted, the fields of the customer object is initialized with the information that extracted from the line. The customer object is pushed to customerArray vector after it's fields are initialized.

The vending machine threads are created after taking the input file. Machine threads takes their id's as parameter. Machine threads loops idle until a customer comes. It starts to execute if a customer comes. I will mention how the machine thread executes int the following parts of the report. The customer threads are created after machine threads. They take customers as parameter.

Customer thread starts executing by converting the void* argument to Customer*. The thread reads the sleepTime field of the customer and sleeps that amount. The customer wants to acquire the lock of the vending machine that the customer wants to operate on. If the lock is acquired by another customer thread, it waits until the other customer to release the lock. After acquiring the lock, customer thread adds the customer to the customerArrayVending. The customer thread also sets the customerCheckArray[machineId] to 1. The customer thread makes busy waiting until the operation is done by the vending machine.

The vending machine thread understands that a new customer comes by checking the customerCheckArray[machineId]. If a new customer arrives, the vending machine thread starts to execute. The vending machine first checks which company that user wants to pay. The machine thread tries to acquire the lock of the company that the user wants to pay. The machine thread waits until the lock is released if the lock is acquired by another thread. The thread acquires write lock to write the file. The write lock is required in order to prevent the writing collisions. The thread writes the customer id, payment amount and the name of the company to the output file. The write lock is released after the writing operation. The vending machine thread also makes the given amount payment to the proper company. The machine sets the customerCheckArray[machineId] to 0 and releases the company lock after these operation. Making customerCheckArray[machineId] 0 ensures the vending machine waits idle until a new customer arrives.

Customer thread quits busy waiting when the customerCheckArray[machineId] is 0. This also means the customer finished its job. Customer releases the machine lock after finishing its job.

The main thread collects the customer threads. The main thread makes sets the check variable to 1. The vending machine threads quit from looping and exit if the check variable is not equal to 0. The main thread also collects the vending machine threads after these operations.

The main thread prints the “All payments are completed” statement and the total payments received by the companies to the output file and finishes the execution.

Ali Başaran

2020400357