PROJECT 3 DOCUMENT

In this project, the goal was to implement a payment system using a multi-threaded algorithm. Customers tried to reach ticket machines and the machines tried to reach shared variables. To prevent race-condition between these threads, mutex locks should be used.

Firstly, the input file is read line by line. Then each line is parsed into data segments. Each line inside the file represents a customer and the parsed data segments corresponds to the data about the customers. Each customer obtains an id according to the position of their line in the input file. Then sleep time, machine instance and the company are gathered from the parsed data segments. Later, these data are combined in a **customer** construct and then they are put into **customers** list.

After filling the **customers** list, main thread starts creating the machine threads using **machine** struct which only contains the id of the machine. Then the creation of the machine threads, customer threads are created using the **customer** structs gathered from the **customers**. The main thread is also responsible for writing the output file. Therefore, it waits until all the customer threads are joined to the main thread.

Upon creation, customer threads read the **customer** struct that was given to them by a void pointer. Right here, the threads are supposed to sleep for a given sleep time. However, putting the threads to sleep unfortunately does not determine which thread would come out of the sleep time and take a lock. I added a barrier which ensures that the threads will start sleeping at the same time. But this does not always solve the problem.

When the customer threads wake up, they start grabbing machine locks. Machine locks are unique to each machine, and they ensure that each machine is being used by a single customer at a time. Machine locks are stored inside **machine_locks** and each machine's lock can be obtained by subtracting 1 from the **machine_id** and getting the lock form that index. The reason to the -1 is that the machine index range goes from 1 to 10 however the list indexes go from 0 to 9.

After acquiring the lock, the customer starts writing the order to the **current_orders** list. This order will be read by the machine, and then it will start executing the order. The machine will check the existence of a new order by checking the id section of its place in the **current_orders** list. So, the id section of the order is filled lastly and then the customer starts waiting for the machine. When the machine is done with the order, it will rechange the id of the **current_orders**. So, the customer checks this part and if it gets the signal from the machine, it will unlock the machine lock and exit.

From the machine's perspective, it firstly collects it data from the **machine** struct. Then starts waiting for a change in the id section of the order inside the **current_orders** list. When it detects the change, it gathers the order. Then increases the balance of the company told by the customer. This balance part is locked because the **balance_list** shared between the machines. Each company has its own balance account, so all accounts have a separate lock. When the machine is done with the balance account, it releases the balance lock of that company and then proceeds to result writing.

Results are stored in a separate array because they will be written to an output file by the main thread when the payment part is done. The result should be sorted according to their prepayment time, and the machine is done with the prepayment. Therefore, each machine can write to result array one at a time, continuing from where the last machine left. This way the earlier prepayments can be written to smaller indexes in the result compared to the late prepayments. When the machine grabs the **result_lock**, it writes prepayment data to **results** list. When it is done it releases the **result_lock**, it starts waiting for another order until the main thread exits the program.

Lastly the main thread writes the output log file and the algorithm finishes. Main thread starts the writing the output file when the customer threads join the main thread. Then main thread creates the output log file and fills it with the data from the **results** list.

Emre Batuhan Göç - 2020400090