2019400057-Yunus Emre Altuğ

# Project#3 - Simulation of a Fun Fair Payment System

In this project, I implemented a multi-threaded application for simulating a fun fair payment system, where the customers can make the prepayments of the various rides through the available ticket vending machines. Since the prepayments can be made simultaneously through different ticket vending machine instances, the synchronization and the data consistency are taken into the consideration throughout the implementation.

The program can work with “make” command or “g++ main.cpp -o simulation -pthread” command. Both commands will create a “simulation” file which executes with this command: “./simulation <input file>”. There will be “<inputfile>_log.txt” file after execution. I excluded “.txt” from the input file name and added “_log.txt” to name output file.

The program consists of three parts: Customer function, Machine function and main part. There are 16 mutex locks and 10 of them are for machines, 5 of them are for companies and the other one is for counting total customers transactions.

In the main part, the program firstly creates machine threads and then receives the inputs. After putting input into an array of array with specific format, the program starts to create customer threads. Each customer thread receives part of the array with information about it. I assumed the number of customers does not pass 300, so the size of array is 300 and each array in that array has 5 integer values. Then, program joins all threads and prints own output into the text file.

In the customer part, the function receives its information list as parameter. After sleeping, transaction can be started. There is a mutex lock for both ordering the customers and protect the information about customers. I implemented an array with 10 elements of which name is “machineArray”, for vending machines. In this array for each machine, customer id is kept. If there is a customer id in that index of array, a machine works with this customer id. If machine is idle, the value in the array is 0. After mutex lock, the value in index of machine is changed and involved machine starts to work. In machine part when the transaction ends, the value will be again 0. Then, the mutex is unlocked and customer exits.

In the machine part, the function receives its vending machine number as parameter. There is a while loop in that part so that machine must be kept awake for whole payment process. If the transaction counts are equal to number of total customers, the machines end working, and the threads join. In the loop each machine checks if any transaction with related it is ready. A machine basically does three things: printing the output, incrementing number of transactions by 1 and depositing the gaining of each company. Each machine merges the output and prints into the file. In this case I implemented a mutex lock to print only one customer’s output and I merged output before printing, so there is not any problem with output. After printing, there is a lock for number of transactions. Finally, company receives the payment with mutex lock and the value in “machineArray” for the machine turns 0 again.