Name: Yuying Zhang
Student #: V00924070
Class: CSC 360 Winter 2020
Date: Nov 6th 2020

Design Document Assignment 2: Airline Check-in System

1. **How many threads are you going to use? Specify the task that you intend each thread to perform.**

In terms of customer threads, the maximum number of registered customers at any run will be 500 and a customer thread will be created for each customer on the input file. So if there were 6 lines in the input file with 5 customers' data, 5 customer threads would be created. A customer thread simulates its respective customer's arrival time, position within the queue including entering and leaving the queue, console logging, processing time, determines overall waiting time, as well as service termination.

There will be 4 clerk threads created to handle customer threads, in accordance to specifications. A clerk thread is responsible for checking the business and economy queue, signaling waiting customers if they are available and updating their working status to let customers know which clerk had signaled them.

2. **Do the threads work independently? Or, is there an overall "controller" thread?**

The threads work independently and there is no overall controller thread as it is not necessary for this implementation design.

3. **How many mutexes are you going to use? Specify the operation that each mutex will guard.**

I will be utilizing 5 mutex locks, as follows:

pthread_mutex_t waitingTimeMutex;
To ensure accessing shared data from the waiting time array is an atomic operation.

pthread_mutex_t economyQueue;
To ensure entering and leaving the economy queue occurs as an atomic operation.

pthread_mutex_t businessQueue;
To ensure entering and leaving the business queue occurs as an atomic operation.

pthread_mutex_t selectedQueueMutex;
To ensure a selected queue for processing by one clerk is not altered by another during signaling for customers.

pthread_mutex_t clerkStatusMutex;

Name: Yuying Zhang
Student #: V00924070
Class: CSC 360 Winter 2020
Date: Nov 6th 2020

To ensure only one clerk can update their status at one time, to avoid conflicts when signaling customers.

4. **Will the main thread be idle? If not, what will it be doing?**

The main thread will be idle after initiating mutex locks, condition variables, and creating customer as well as clerk threads. The main thread will return to being active when customer and clerk threads terminate and join. The main thread also destroys all mutex locks, condition variables, and other pthread objects. Finally, it prints out total waiting times and simulation time.

5. **How are you going to represent customers? What type of data structure will you use?**

Customers will be represented through a struct data structure like the following:
struct customer: int id; int class; float arrivalTime; float serviceTime;
Clerks will be represented through a struct data structure like the following:
struct clerk: int id; int status; int signaled;

6. **How are you going to ensure that data structures in your program will not be modified concurrently?**

Data structures on my program will not be modified concurrently through the use of mutex locks to ensure that shared data that will be accessed by many threads can only be modified by one thread at a particular time. Condition variables ensure that only when certain conditions are met for a particular thread will it be accepted by a clerk.

7. **How many convars are you going to use? For each convar:**

I intend to use 5 condition variables.

pthread_cond_t clerkAvailable;
a) A condition that represents whether a clerk is available to serve a customer.
b) &economyQueue and &businessQueue mutex locks are associated.
c) Determine which clerk signaled that it is available and update that clerk's status.

pthread_cond_t clerk0Finished;
a) A condition that represents whether clerk 0 is finished working with its customer.
b) &clerkStatusMutex is associated.
c) Changes the signaling clerk's status such that it becomes available for the next job.

pthread_cond_t clerk1Finished;
a) A condition that represents whether clerk 1 is finished working with its customer.
b) &clerkStatusMutex is associated.
c) Changes the signaling clerk's status such that it becomes available for the next job.

pthread_cond_t clerk2Finished;

a) A condition that represents whether clerk 2 is finished working with its customer.
b) &clerkStatusMutex is associated.
c) Changes the signaling clerk's status such that it becomes available for the next job.

pthread_cond_t clerk3Finished;

a) A condition that represents whether clerk 3 is finished working with its customer.
b) &clerkStatusMutex is associated.
c) Changes the signaling clerk's status such that it becomes available for the next job.

8. Briefly sketch the overall algorithm you will use.

```
typedef struct customer
{
   int id;
   int class;
   float arrivalTime;
   float serviceTime;
} customer;

typedef struct waitEntry
{
   int class;
   double waitTime;
} waitEntry;

typedef struct clerk
{
   int id;
   int status;
   int signaled;
} clerk;

double getTimeDifference(struct timeval startTime)
{
   Returns the system time difference in seconds between a passed start time to function call
}

void fileReader(char *path, char fileContents, customer customerList)
{
   Reads input file and parses the information into customer type entries
}

void removeFromQueue(int class)
{
   Removes the head of the queue, class determines which queue
}
```

Name: Yuying Zhang
Student #: V00924070
Class: CSC 360 Winter 2020
Date: Nov 6th 2020

```
int findClerk(int class)
{
    A function for finding the clerk that signaled
}

int clerkNotTaken()
{
    A function that checks the clerk status to know it's not taken
}

waitForClerk(customer)
{
    clerkSignal;

    if (currCustomer->class == 0)
    {
        mutex_lock(&economyQueue)
        Adds currCustomer to economy queue
        Increase length of queue

        while (TRUE)
        {
            if clerk is available cond_wait(&clerkAvailable) then
            if (currCustomer->id == head of queue && selectedQueue == economy queue &&
clerkNotTaken())
            {
                Find clerk that signaled
                Update clerk status
            }
        }
        Remove currCustomer from existing queue
        mutex_unlock(&economyQueue)
    }
    else //  This is the business class queue
    {
        Same process as above for business class, change mutex as &businessQueue...
    }
}

void *customerEntry(void *customerInfo)
{
    struct timeval waitingTime // Starts waiting time

    usleep(currCustomer->arrivalTime) // Simulates arrival time

    Starts waitingTime

    waitForClerk(currCustomer); // Enters into the queue till clerk signals
```

Name: Yuying Zhang
Student #: V00924070
Class: CSC 360 Winter 2020
Date: Nov 6th 2020

```
    double overallWaitingTime = getTimeDifference(waitingTime) // Ends waiting time

    usleep(10);

    Prints when clerk starts serving the customer

    usleep(processing time)

    Prints when clerk finishes serving the customer

    Based on clerk id, signal which clerk just finished serving the customer
    case 0:
        (pthread_cond_signal(&clerk0Finished)
        break;
    case 1:
        (pthread_cond_signal(&clerk1Finished)
        break;
    case 2:
        (pthread_cond_signal(&clerk2Finished)
        break;
    case 3:
        (pthread_cond_signal(&clerk3Finished)
        break;
    }

    mutex_lock(&waitingTimeMutex)
    Inputs waiting time into waiting time records
    mutex_unlock(&waitingTimeMutex)
}

void *clerkEntry(void *clerkInfo)
{

    while (TRUE)
    {
        /* The clerk will continue working until the number of wait entries equals total customers served */
        if both business and economy queues are empty
            if number of entries in waiting times array is equal to number of customers
                All customers are served so return to main

        if (currClerk->status == 0) // If clerk is idle...
        {
            pthread_mutex_lock(&selectedQueueMutex)

            if (businessQueueLength > 0)
            {
                selectedQueue = 1;
                mutex_lock(&clerkStatusMutex)
                currClerk->signaled = selectedQueue
                mutex_unlock(&clerkStatusMutex)

                mutex_lock(&businessQueue)
                pthread_cond_broadcast(&clerkAvailable)
                mutex_unlock(&businessQueue)
            }
            else
            {
                selectedQueue = 0;
                mutex_lock(&clerkStatusMutex)
                currClerk->signaled = selectedQueue;
                mutex_unlock(&clerkStatusMutex)

                mutex_lock(&economyQueue)
                pthread_cond_broadcast(&clerkAvailable);
                mutex_unlock(&economyQueue)
            }
            (pthread_mutex_unlock(&selectedQueueMutex)
        }
        else // If clerk is busy...
        {
            pthread_cond_wait on all clerk finished variables and update the clerk's status accordingly
        }
}

void printWaitingTimes()
{
    Prints and calculates all waiting times
}

int main(int argc, char *argv[])
{
    Initialize all the condition variables and mutex locks that will be used

            Read customer information from txt file and store them in the customer structure

    Create customer threads

    Create clerk threads

    Join threads

    Destroy pthread objects
}
```