Advanced Operating System: COEN 383

Project 3

Group No. 6

Varun Toshniwal (W1620487)

Maharsh Suryawala (W1628800)

Orion Sun (W1361233)

Malavika Sreedhar Tankala (W1628801)

Yulin Zeng (W1629103)

Software Design:

We implemented a C program that uses the Pthread library to create threads in order to simulate ticket sellers who sell tickets based on their category for a concert of 100 seats.

For each of the 10 sellers a customer queue is created having N customers. Customer arrival times are in terms of minutes and are randomly generated. Customers arrive at the beginning of the minute.Customers are served based on the seller type (H,M,L). Once a seller begins to serve a customer it will look for a seat in the AVAILABLE state which is denoted by a locked mutex to ensure only one seller can access a given seat at a given time. Once the seller reserves the seat the mutex is unlocked. Similarly all customers are served until 1 hour unless the seats are sold out. If some customers are not served within the 1 hour time frame, they are turned away. At the end of the simulation, the average response time, average turnaround time and average throughput is calculated for each seller type. The number of customers served and turned away are also calculated.

- What parameters did you adjust to make the simulation run realistically?

The seat_struct is an important parameter where the state of a seat is modified between Available, Sold and Processing states.

The customer parameter determines the arrival time of the customer and identifies each customer with a unique customer id.

The process_time parameter is used along with the rand() function to make sure the H type seller uses randomly exactly 1-2 minutes to complete a ticket sale, the M type seller uses randomly exactly 2,3,or 4 minutes to complete a ticket sale and the L type seller uses randomly exactly 4,5,6 or 7 minutes to complete a ticket sale.

The theater parameter is used throughout the simulation to realistically manage and portray the seat allocation.

- What process synchronization was required?

In this multi-threaded Tickets' Sellers project, different tickets sellers may want to sell the same ticket. So it may cause problems, like generating dirty data or overwriting the previous records. To avoid this situation, we need to choose one process synchronization method to ensure that two sellers won't assign the same seat to two different customers.

There are three methods of process synchronization, including mutex, semaphore, and event. In this project, we use the **Pthread's mutex** to make sure **t**he tickets won't be sold again.

- What data was shared and what were the critical regions?

All the ticket sellers shared the same seat table. And in the real code, the data is an Array of seats. We also use sear_manager to manage the seats. Which are shown in the following picture.

```
// generating 100 seats
seat theater[100];
```

```
typedef struct seat_struct
{
    int id;
    int counter;
    enum seat_state state;
    customer *cust;
    pthread_args *p_args;
} seat;
```

Shared data and data manager

And the critical region of our program is shown in the following picture. The critical region has three parts which are inside the get_M_seat_to_sell, get_H_seat_to_sell(), and get_L_seat_to_sell() function. And the exact part of the code is shown in the following pictures.

```c
// sells h seat
seat *get_h_seat_to_sell()
{
    seat *allocated_seat_to_sell;
    pthread_mutex_lock(&seat_lock);
    if (seat_manager.free_seats > 0)
    {
        // If the seat pointed to by the seat_manager is not free, get a free seat.
        if (seat_manager.h_seat->state != AVAILABLE)
            h_next_seat();
        allocated_seat_to_sell = seat_manager.h_seat;
        allocated_seat_to_sell->state = PROCESSING;
        seat_manager.free_seats--;
        h_customers++;
    }
    else
    {
        allocated_seat_to_sell = NULL;
    }
    pthread_mutex_unlock(&seat_lock);
    return allocated_seat_to_sell;
}
```

```c
// sells m seat
seat *get_m_seat_to_sell()
{
    seat *allocated_seat_to_sell;
    pthread_mutex_lock(&seat_lock);
    if (seat_manager.free_seats > 0)
    {
        // If the seat pointed to by the seat_manager is not free, get a free seat.
        if (seat_manager.m_seat->state != AVAILABLE)
            m_next_seat();
        allocated_seat_to_sell = seat_manager.m_seat;
        allocated_seat_to_sell->state = PROCESSING;
        seat_manager.free_seats--;
        m_customers++;
    }
    else
    {
        allocated_seat_to_sell = NULL;
    }
    pthread_mutex_unlock(&seat_lock);
    return allocated_seat_to_sell;
}
```

```c
// sells l seat
seat *get_l_seat_to_sell()
{
    seat *allocated_seat_to_sell;
    pthread_mutex_lock(&seat_lock);
    if (seat_manager.free_seats > 0)
    {
        // If the seat pointed to by the seat_manager is not free, get a free seat.
        if (seat_manager.l_seat->state != AVAILABLE)
            l_next_seat();
        allocated_seat_to_sell = seat_manager.l_seat;
        allocated_seat_to_sell->state = PROCESSING;
        seat_manager.free_seats--;
        l_customers++;
    }
    else
    {
        allocated_seat_to_sell = NULL;
    }
    pthread_mutex_unlock(&seat_lock);
    return allocated_seat_to_sell;
}
```

The critical region

Simulation result:

-The simulation result depicts the concert seating allocation that shows which customer was assigned to each seat and also represents the unsold seats using *.

- It also shows how many customers were assigned seats based on H,M,L seller type and how many customers were turned away.

Statistics:

We run the simulation for 5, 10, and 15 customers per ticket seller, and we get average response time, average turnaround time, and average throughput of H1, M1 to M3, and L1 to L6. Depending on those results, we calculate those data and get the final statistics of the whole simulation project. And the statistics are shown in the following table.

Seller Type	N	Average turnaround time	Average response time	Average throughput
H	5	63	32	1
M	5	471	244	26
L	5	622	272	46
H	10	65	33	1
M	10	1187	584	48
L	10	1314	636	92
H	15	51	23	1
M	15	1497	702	60
L	15	984	462	88

Basic Statics per seller type

Finally, we can get the average response time per customer for a given seller type, average turn-around time per customer for a given seller type, average throughput per seller type. And the result is shown in the following table.

Seller Type	Average turnaround time	Average response time	Average throughput
H	59.67	29.33	1
M	1051.67	510	44.67
L	973.33	456.67	75.33

Statistics