|  |
| --- |
| **Project 2: Threads and Semaphores** |
| CSCI 474: Operating Systems Concepts |
|  |
| The summary report about the hotel simulation project which utilized semaphores and multithreading for the course of CSCI 474: Operating Systems Concepts |
|  |
| **Akshat Sharma** |
| **12/5/2013** |
|  |

Akshat Sharma

CSCI 474: Operating Systems Concepts

Project 2: Semaphores and Multithreading

12/05/2013

**SUMMARY TABLE**

|  |  |  |
| --- | --- | --- |
| Semaphore | Function | Initial Value |
| availableReservationist\_CheckInCounter | To check if a reservationist is available at the check-in counter to serve the guest, or if the reservationist is already serving another guest | 1 |
| availableReservationist\_CheckOutCounter | To check if a reservationist is available at the check-out counter to serve the guest, or if the reservationist is already serving another guest | 1 |
| greet\_checkInReservationist | To ensure that the reservationist at the check-in counter first greets the guest and then moves on with the check-in process.  Enforces event ordering | 0 |
| greet\_checkOutReservationist | To ensure that the reservationist at the check-out counter first greets the guest and then moves on with the check-out process.  Enforces event ordering | 0 |
| availableRoom | To ensure that a maximum of 5 rooms are assigned to the guests. If all the rooms are occupied, and then a guest would have to check-out, and only after that can a guest check-in | 5 |
| guest\_getsRoom | To inform the guest method that guest has received the room and can now complete the check-in and perform any random activity.  Enforces event ordering | 0 |
| calculateBalanceForGuest | Semaphore to make sure that the balance for the guest is calculated first, and only then can a guest make the payment  Enforces event ordering | 0 |
| guest\_Payment | Semaphore to ensure that the guest makes a payment only after the balance is calculated, and ensures that only after making the payment can the guest complete the check-out process.  Enforces event ordering | 0 |
| roomAssignment\_mutex | Semaphore to protect the shared resource which is accessed by the checkIn and the checkOut method  Enforces Mutual Exclusion. | 1 |

**What was learnt in this Project?**

**Multithreading**

This project involved 2 main concepts, namely, multithreading and semaphores. Both of which I have implemented for the first time in this project. Through this project, I have gained knowledge about threads and their implementation. By using certain online resources and gaining knowledge about threads from the class discussion, implementation of multiple threads got much more understandable and easier to perform. For this project, we needed to create a total of 12 threads, which included 10 guest threads, 1 thread for check-in and 1 thread for check-out. Next, after successful creation of these threads, we needed to make sure that these threads are synchronized and not running randomly. The next thing I learned about multithreading was about joining the threads. After successful creation of threads we need to join these threads. By using the join method, we can pause the current thread while another thread completes its required tasks. This also helps in synchronization of the program. Then at the end of the program, we should call the method pthread\_exit() in order to exit the threads.

**Semaphores**

Along with the concept of multithreading, I learnt another concept in this Project, which was semaphores. Semaphores are used for mutual exclusion, which is protecting the critical section, and also used for event ordering. With this project, I gained practical knowledge of using semaphores. This project utilized a total of 9 semaphores. I learned how to declare, initialize, and destroy semaphores by completing this project. The most important step is to initialize the semaphores correctly, as if the semaphores are initialized with a wrong value, a number of problems can arise in the execution sequence of the code.

**Problems encountered in the Project**

Since this was the first time I have a done a project using semaphores and multithreading, I did encounter a few problems while completing the project. The problems I encountered were mainly related to the semaphores used in this project.

* At first, I did have problems synchronizing the threads, and to solve this problem, I created a pseudo code of the project code, to check where the semaphores would be implemented. This helped me solve a lot of problems and my project was thus synchronized correctly.
* Secondly, I encountered certain problems with the initial values of the semaphores. There were a couple of places where I had initialized the semaphores with a wrong value. This was thus resulting in a deadlock in my program. To solve this, I did a test run of the semaphores on a sheet of paper, and thus found out that the values of these semaphores were going into negative, which thus resulted in blockage of the threads.

**Results**

At the end of the Project, I finished the project by getting the correct results. The code is synchronized correctly by using event ordering with semaphores, and all the shared resources are protected by implementing mutual exclusion. Lastly, at the end, the program displays the total number of guests, and then further splits them down based on the activity they did.

**REFERENCES**

http://www.youtube.com/watch?v=jPcXuKhMC6I

http://www.youtube.com/watch?v=EVNCIjySBDU

https://computing.llnl.gov/tutorials/pthreads/ http://www.csc.villanova.edu/~mdamian/threads/posixsem.html

http://www.amparo.net/ce155/sem-ex.html

http://www.daniweb.com/software-development/c/threads/330806/semaphores-in-c