**Project #2**

***Design***

**Semaphores:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Initial Value** |
| sem\_main\_terminate | Called to end the Program | 0 |
| sem\_informationLine | Limits the Capacity of the Information Line | 10 |
| sem\_agentLine | Limits the Capacity of the Agent Line | 10 |
| sem\_customer\_arriveAtInformationLine | Called to let the Information Desk know that a Customer has arrived at the Information Line | 0 |
| sem\_customer\_arriveAtWaitingArea | Called to let the Announcer know that a Customer has arrived at the Waiting Area | 0 |
| sem\_customer\_arriveAtAgent[] | Called to let the Agents know that the Customer has arrived at their Desk | 0 |
| sem\_customer\_takeExam[] | Called to let the Agents know that the Customer has finished the Exam | 0 |
| sem\_customer\_takeLicense[] | Called to let the Agents know that the Customer has taken the License | 0 |
| sem\_customer\_leaveDMV[] | Called to let the Agents know that the Customer has left the DMV | 0 |
| sem\_informationDesk\_callCustomer[] | Called to let the Customer know that the Information Desk has called them | 0 |
| sem\_announcer\_callCustomer[] | Called to let the Customer know that the Announcer has called them | 0 |
| sem\_agent\_callCustomer[] | Called to let the Customer know that an Agent has called them | 0 |
| sem\_agent\_giveExam[] | Called to let the Customer know that the Agent has given them an Exam | 0 |
| sem\_agent\_giveLicense[] | Called to let the Customer know that the Agent has given them a License | 0 |
| sem\_mutex\_customerCount | Provides Mutual Exclusion for the Customer Count Variable | 1 |
| sem\_mutex\_agentCount | Provides Mutual Exclusion for the Agent Count Variable | 1 |
| sem\_mutex\_customersProcessed | Provides Mutual Exclusion for the Customers Processed Variable | 1 |
| sem\_mutex\_informationQueue | Provides Mutual Exclusion for the Information Queue Variable | 1 |
| sem\_mutex\_waitingAreaQueue | Provides Mutual Exclusion for the Waiting Area Queue Variable | 1 |
| sem\_mutex\_agentQueue | Provides Mutual Exclusion for the Agent Queue Variable | 1 |

**Pseudocode:**

//\* Description \*//

// Title: Pseudocode

// Author: Tyler Reed

// Defines the Pseudocode for Simulating the DMV

//\* Semaphores \*//

// Main Semaphores

**semaphore** sem\_main\_terminate = 0;

// Counting Semaphores

**semaphore** sem\_informationLine = 10;

**semaphore** sem\_agentLine = 10;

// Customer Semaphores

**semaphore** sem\_customer\_arriveAtInformationLine = 0;

**semaphore** sem\_customer\_arriveAtWaitingArea = 0;

**semaphore** sem\_customer\_arriveAtAgent[100] = {0};

**semaphore** sem\_customer\_takeExam[100] = {0};

**semaphore** sem\_customer\_takeLicense[100] = {0};

**semaphore** sem\_customer\_leaveDMV[100] = {0};

// Information Desk Semaphores

**semaphore** sem\_informationDesk\_callCustomer[100] = {0};

// Announcer Semaphores

**semaphore** sem\_announcer\_callCustomer[100] = {0};

// Agent Semaphores

**semaphore** sem\_agent\_callCustomer[100] = {0};

**semaphore** sem\_agent\_giveExam[100] = {0};

**semaphore** sem\_agent\_giveLicense[100] = {0};

// Mutual Exclusion Semaphores

**semaphore** sem\_mutex\_customerCount = 1;

**semaphore** sem\_mutex\_agentCount = 1;

**semaphore** sem\_mutex\_customersProcessed = 1;

**semaphore** sem\_mutex\_informationQueue = 1;

**semaphore** sem\_mutex\_waitingAreaQueue = 1;

**semaphore** sem\_mutex\_agentQueue = 1;

//\* Counting Variables \*//

**int** customer\_count = 0;

**int** customers\_processed = 0;

//\* Queues \*//

**queue**<Customer\*> informationQueue;

**queue**<int> waitingAreaQueue;

**queue**<int> agentQueue;

**void** Customer::simulate()

{

wait(sem\_mutex\_customerCount);

id = customer\_count++;

signal(sem\_mutex\_customerCount);

arriveAtDMV();

wait(sem\_informationLine);

wait(sem\_mutex\_informationQueue);

informationQueue.enqueue(**this**);

signal(sem\_mutex\_informationQueue);

arriveAtInformationLine();

signal(sem\_customer\_arriveAtInformationLine);

wait(sem\_informationDesk\_callCustomer[id]);

arriveAtWaitingArea();

signal(sem\_customer\_arriveAtWaitingArea);

signal(sem\_informationLine);

wait(sem\_announcer\_callCustomer[ticket\_number]);

arriveAtAgentLine();

signal(sem\_customer\_arriveAtAgentLine);

wait(sem\_agent\_callCustomer[ticket\_number]);

arriveAtAgent();

signal(sem\_customer\_arriveAtAgent[ticket\_number]);

signal(sem\_agentLine);

wait(sem\_agent\_giveExam[ticket\_number]);

takeExam();

signal(sem\_customer\_takeExam[ticket\_number]);

wait(sem\_agent\_giveLicense[ticket\_number]);

takeLicense();

signal(sem\_customer\_takeLicense[ticket\_number]);

wait(sem\_mutex\_customersProcessed);

customers\_processed++;

signal(sem\_mutex\_customersProcessed);

leaveDMV();

signal(sem\_customer\_leaveDMV[ticket\_number]);

}

**void** InformationDesk::simulate()

{

wait(sem\_customer\_arriveAtInformationLine);

wait(sem\_mutex\_informationQueue);

Customer\* customer = informationQueue.dequeue();

signal(sem\_mutex\_informationQueue);

customer.setTicketNumber(ticket\_number);

wait(sem\_mutex\_waitingAreaQueue);

waitingAreaQueue.enqueue(ticket\_number++);

signal(sem\_mutex\_waitingAreaQueue);

callCustomer();

signal(sem\_informationDesk\_callCustomer[customer.getID()]);

}

**void** Announcer::simulate()

{

wait(sem\_agentLine);

wait(sem\_customer\_arriveAtWaitingArea);

wait(sem\_mutex\_waitingAreaQueue);

**int** ticket\_number = waitingAreaQueue.dequeue();

signal(sem\_mutex\_waitingAreaQueue);

wait(sem\_mutex\_agentQueue);

agentQueue.enqueue(ticket\_number);

signal(sem\_mutex\_agentQueue);

callCustomer();

signal(sem\_announcer\_callCustomer[ticket\_number]);

}

**void** Agent::simulate()

{

wait(sem\_customer\_arriveAtAgentLine);

wait(sem\_mutex\_agentQueue);

int ticket\_number = agentQueue.dequeue();

signal(sem\_mutex\_agentQueue);

callCustomer();

signal(sem\_agent\_callCustomer[ticket\_number]);

wait(sem\_customer\_arriveAtAgent[ticket\_number]);

giveExam();

signal(sem\_agent\_giveExam[ticket\_number]);

wait(sem\_customer\_takeExam[ticket\_number]);

giveLicense();

signal(sem\_agent\_giveLicense[ticket\_number]);

wait(sem\_customer\_takeLicense[ticket\_number]);

wait(sem\_custoemr\_leaveDMV[ticket\_number]);

}