Design

List of Semaphores

mutex1: mutex1 block used for enqueueing and dequeueing from info desk queue, initialized to 1

mutex2: mutex2 block used for enqueueing and dequeueing from waiting room queue, initialized to 1

mutex3: mutex3 block used for enqueueing and dequeueing from agent line queue, initialized to 1

coord: coordination semaphore used for ensuring correct order of agent/customer steps, initialized to 0

customer_ready_at_info_desk: used to let the info desk know there is a customer in the info desk line, initialized to 0

number_assigned: used to let the customer know that the info desk has assigned it a number, initialized to 0

customer_in_waiting_room: used to let the announcer know there is a customer in the waiting room, initialized to 0

announced: used to let the customer know that the announcer has called its ticket number, initialized to 0

agent_line_capacity: used to ensure only 4 customers are in the agent line at a time, initialized to 4

customer_in_agent_line: used to let the agents know that there is a customer in the agent line, initialized to 0

available_agent: used to make sure at most 2 customers are processed at a time because there are only 2 agents, initialized to 2

customer_being_served: used to let the customer know it is being served by an agent, initialized to 0

customer_acknowledgement: used to let the agent know that its customer knows it is being served so it can ask the customer to take photo and eye exam, initialized to 0

photo_and_eye_exam_request: used to let the customer know that an agent is asking it to take a photo and eye exam, initialized to 0

completed_photo_and_eye_exam: used to let the agent know that its customer has taken a photo and eye exam, initialized to 0

finished[20]: used to let each individual customer know that it has been given a license, each semaphore in the array is initialized to 0

Thread Psuedocode

Customer

```
void customer_thread(arg) {
    int tid = arg;
    // customer gets created, enters DMV

    wait(mutex1);
        enqueue(info_desk_queue, customer);
        signal(customer_ready_at_info_desk);
    signal(mutex1);

    wait(number_assigned);
    // customer gets number, enters waiting room

    wait(mutex2);
        enqueue(waiting_room_queue, customer);
        signal(customer_in_waiting_room);
    signal(mutex2);

    wait(announced);
    // customer moves to agent line

    wait(mutex3);
        enqueue(agent_line_queue, customer);
        signal(customer_in_agent_line);
    signal(mutex3);

    wait(customer_being_served);

    // customer is being served by agent
    signal(customer_acknowledgement);

    wait(photo_and_eye_exam_request);
    wait(coord);

    // customer completes photo and eye exam for agent
    signal(completed_photo_and_eye_exam);

    wait(finished[tid]);
    // customer gets license and departs
    // customer was joined
    return arg;
}
```

Info Desk

```
void info_desk_thread(arg) {
    // information desk created
    int customer_count = 1;
    while(true) {
        if(customer_count > 20) {
            return arg; // exit
        }

        wait(customer_ready_at_info_desk);

        wait(mutex1);
            customer = dequeue(info_desk_queue);
            customer->ticket_num = customer_count;
            signal(number_assigned);
        signal(mutex1);

        customer_count++;
    }
    return arg;
}
```

Announcer

```
void announcer_thread(arg) {
    // announcer created
    int customer_count = 1;
    while(true) {
        if(customer_count > 20) {
            return arg; // exit
        }

        wait(customer_in_waiting_room);

        wait(mutex2);
            wait(agent_line_capacity);
            customer = dequeue(waiting_room_queue);
            // announcer calls customer->ticket_num
        signal(mutex2);

        signal(announced);
        customer_count++;
    }
    return arg;
}
```

Agent

```
void agent_thread(arg) {
    tid = arg;
    // agent created
    while(true) {
        if(served_customers >= 20) {
            return arg; // exit
        }

        wait(customer_in_agent_line);

        wait(mutex3);
            wait(available_agent);
            served_customers++;
            customer = dequeue(agent_line_queue);
            customer->agent_num = tid;
            // agent is serving customer->threadid
            signal(customer_being_served);
        signal(mutex3);

        wait(customer_acknowledgement);

        // agent asks customer to take photo and eye exam
        signal(coord);
        signal(photo_and_eye_exam_request);

        wait(completed_photo_and_eye_exam);

        // agent gives license to customer
        signal(finished[customer->threadid]);
        signal(available_agent);
        signal(agent_line_capacity);
    }
    return arg;
}
```