CSE344 Homework 3 Report

Akif Safa Angi - 200104004079

1. Problem Definition and My Solution Approach

The task required the implementation of a synchronization mechanism for a parking lot using semaphores and shared memory in C. The parking lot has separate spaces for pickups and automobiles, with a capacity of four spots for pickups and eight spots for automobiles. The challenge was to manage the parking process using two types of threads: `carOwner()` and `carAttendant()`. The `carOwner()` threads simulate vehicle owners, and the `carAttendant()` threads simulate parking attendants. The synchronization between these threads is achieved using semaphores.

My Solution

I've based my solution on the Sleeping Barber problem. Also use one more semaphore for ensure only one vehicle can enter to free parking area.

I created two carAttendant instances in my solution. Each is responsible for transferring vehicles from the free parking to the permanent parking. I created a sufficient number of carOwner threads, each representing a vehicle owner. Each vehicle waits in line to enter the free parking space and, if there is space available, parks there. Then, if there is space available in the permanent parking, it waits in line to be parked by the valet. If there is no space, it waits for a certain period and then leaves the free parking area. If there is no space available in the free parking area, the vehicles leave before entering the parking area.

I used four counters (**mFreeAutomobile**, **mFree**P**ickup**, **parkedAutomobile**, **parkedPickup**) to control the number of vehicles in the free and parking areas. Additionally, I used semaphores (**parkEntrance**) to ensure that each vehicle can enter the free parking in turn. To prevent synchronization issues while vehicles are parking in the free parking or being transferred to the permanent parking, I used separate semaphores for each type of vehicle (**controlParking** semaphores). This prevented issues such as increasing the **mFreeAutomobile** slots before the vehicle was transferred and allowing other vehicles to enter the free parking.

I used semaphores (**newPickup** and **newAutomobile**) to alert the valet when a vehicle parked in the free parking and to ensure that it is transferred to the permanent parking if space is available. This way, when a new vehicle parks in the free parking, it alerts the valet and the valet moves the vehicle if space is available.

Additionally, I used semaphores (**inChargeforPickup** and **inChargeforAutomobile**) for vehicles to wait for the completion of the process of being parked in the permanent parking so that they can exit the thread.

2. Detailed Implementation

Semaphore Initialization

- **newPickup** and **newAutomobile**: Notify attendants that a new vehicle is parked in the free parking.

- **inChargeforPickup** and **inChargeforAutomobile**: Ensure the valet parks the vehicle from the free spot to a permanent spot.

- **controlAutomobileParking** and **controlPickupParking**: Prevent race conditions while moving cars from free parking to regular parking.

- **parkEntrance**: Prevent race conditions while waiting for entrance to free parking.

Shared Memory

Two counter variables are used to track the availability of temporary parking spots:

- **mFree_automobile**: Free spots for automobiles

- **mFree_pickup**: Free spots for pickups

- **parkedAutomobile**: Permanent spots for automobiles

- **parkedPickup**: Permanent spots for pickups

- **waitingQueueAutomobile**: Controls the queue for waiting to park in permanent area. Prevents more than 8 cars wait for permanent parking. Otherwise some cars wait forever for parking

- **waitingQueuePickup**: Controls the queue for waiting to park in permanent area. Prevents more than 8 cars wait for permanent parking. Otherwise some cars wait forever for parking

Threads

1. carOwner() Thread

- Represents the vehicle owner.

- Confirms the availability of temporary parking space.

- Waits for the parking attendant if space is available, otherwise exits.

2. carAttendant() Thread

- Represents the valet.

- Waits at the temporary parking lot to park vehicles to permanent part from free temporary parking.

- Updates the availability status of parking spaces upon parking a vehicle.

3. System Requirements

The Makefile includes commands to compile the program (`make`), run the program (`make run`), and clean up the compiled files (`make clean`).

4. The Parts I Have Completed

I have completed all the requirements of the homework and tested the implementation rigorously.

5. Detailed Code Explanation

carOwner() Function

The `carOwner()` thread represents a vehicle owner. For automobiles, it works as follows:

1. Decreases the `parkEntrance` semaphore to ensure only one vehicle can enter at a time.

2. Waits on the `controlAutomobileParking` semaphore to prevent race conditions. Because there may be issues while increasing and decreasing `mFreeAutomobile` and `parkedAutomobile` between automobile threads and valet.

3. If there is no space in the parking area, it increases the `controlAutomobileParking` and `parkEntrance` semaphores. By posting `parkEntrance`, new vehicles can enter the parking area. With `controlAutomobileParking`, if there are waiting new automobiles or automobile valet, they can now proceed.

4. If there is space available, it decrements the `mFreeAutomobile` counter, indicating that a new automobile (itself) has entered the parking area. It prints a message confirming parking in the free parking area. Then, it posts `newAutomobile`, which wakes up the valet waiting for a vehicle to park. With `parkEntrance` posted, new vehicles can now enter the parking area.

5. Then, it checks the ` waitingQueueAutomobile ` counter. If it's not zero, the current vehicle can proceed to permanent parking. Therefore, it decreases waitingQueueAutomobilevalue, posts `controlAutomobileParking`, allowing waiting new automobiles or valets to enter the critical region. It waits with `inChargeForAutomobile` for the parking process from free parking to permanent parking to finish. After the valet parks the vehicle, it posts this semaphore to wake up this car thread.

6. If `waitingQueueAutomobile` is zero, it means there are no automobiles parked in the regular parking area. In this case, the `carOwner()` thread posts the `controlAutomobileParking` semaphore, allowing waiting new automobiles or valets to enter the critical region. After posting the semaphore, the thread sleeps for a random time. When it wakes up, it waits on `controlAutomobileParking` again. This ensures that the increment of `mFreeAutomobile` and the associated print statement occur within the critical region, preventing race conditions or unsynchronized behavior.

Once inside the critical region, the thread increases `mFreeAutomobile`, prints relevant information, and finally posts the semaphore to allow other threads to proceed. This ensures that the modification of `mFreeAutomobile` is synchronized and consistent across all threads accessing it.

```
void* carOwner(void* arg) {
    int vehicleType = rand() % 2; // 0 for automobile, 1 for pickup
    // int vehicleType = 0;
    if (vehicleType == 0) { // Automobile
        sem_wait(&parkEntrance);
        sem_wait(&controlAutomobileParking);
        if (mFree_automobile > 0) {
            mFree_automobile--;
            printf("Automobile parked in free parking. Remaining spots in free parking: %d. Remaining Spots in regular parking: %d\n", mFree_automobile, parkedAutomobile);
            sem_post(&newAutomobile);
            sem_post(&parkEntrance);
            if(waitingQueueAutomobile == 0) {
                sem_post(&controlAutomobileParking);
                int randTime = rand() % 3;
                sleep(randTime);
                sem_wait(&controlAutomobileParking);
                mFree_automobile++;
                printf("Regular parking is still full. Automobile is leaving from free parking. Remaining spots in free parking %d\n", mFree_automobile);
                sem_post(&controlAutomobileParking);
            } else {
                waitingQueueAutomobile--;
                sem_post(&controlAutomobileParking);
                sem_wait(&inChargeforAutomobile);
            }
        } else {
            sem_post(&controlAutomobileParking);
            sem_post(&parkEntrance);
            printf("No space for another automobile in free parking.\n");
        }
    } else {
```

Same code for pickup

```
    } else {
        sem_wait(&parkEntrance);
        sem_wait(&controlPickupParking);
        if (mFree_pickup > 0) {
            mFree_pickup--;
            printf("Pickup parked in free parking. Remaining spots in free parking: %d. Remaining Spots in regular parking: %d\n", mFree_pickup, parkedPickup);
            sem_post(&newPickup);
            sem_post(&parkEntrance);
            if(waitingQueuePickup == 0) {
                sem_post(&controlPickupParking);
                int randTime = rand() % 3;
                sleep(randTime);
                sem_wait(&controlPickupParking);
                mFree_pickup++;
                printf("Regular parking is still full. Pickup is leaving from free parking. Remaining spots in free parking. %d\n", mFree_pickup);
                sem_post(&controlPickupParking);
            } else {
                waitingQueuePickup--;
                sem_post(&controlPickupParking);
                sem_wait(&inChargeforPickup);
            }
        } else {
            sem_post(&controlPickupParking);
            sem_post(&parkEntrance);
            printf("No space for another pickup in free parking.\n");
        }
    }
```

At the end, if all cars left, wake valets, so they won't wait for more vehicles.

```
    finishThreads++;
    if(finishThreads == MAX_VEHICLES - 2) {
        finish = 1;
        sem_post(&newAutomobile);
        sem_post(&controlAutomobileParking);
        sem_post(&newPickup);
        sem_post(&controlPickupParking);
    }
    pthread_exit(NULL);
}
```

carAttendant() Function

The `carAttendant()` thread represents the valet. It is in a while loop and will continue indefinitely. For automobiles, it works as follows:

1. Waits on the `newAutomobile` semaphore to be notified of a new vehicle in free parking.

2. Uses the `controlAutomobileParking` semaphore to synchronize parking operations.

3. Checks for space in permanent parking:

   - If no space is available or all cars finished, posts semaphores and breaks the loop.

- If space is available, moves the vehicle from free to permanent parking. Increases the `mFreeAutomobile` counter because it moves from free to permanent parking. Thus, new space becomes available in the free parking area. Decreases `parkedAutomobile` because a vehicle has been parked in the permanent parking area. Prints a message indicating that a vehicle has been parked.

4. Posts semaphores to allow other threads to proceed.

```
void* carAttendant(void* arg) {
    intptr_t argInt = (intptr_t)arg;
    if(argInt == 0) {
        while(1) {
            sem_wait(&newAutomobile);
            sem_wait(&controlAutomobileParking);
            if(parkedAutomobile == 0 || finish == 1) {
                sem_post(&inChargeforAutomobile);
                sem_post(&controlAutomobileParking);
                break;
            }
            mFree_automobile++;
            parkedAutomobile--;
            printf("Automobile parked in regular parking. Remaining spots in free parking: %d. Remaining Spots in regular parking: %d\n", mFree_automobile, parkedAutomobile);
            sem_post(&inChargeforAutomobile);
            sem_post(&controlAutomobileParking);
        }
    } else {
```

Same code for pickup

```
    } else {
        while(1) {
            sem_wait(&newPickup);
            sem_wait(&controlPickupParking);
            if(parkedPickup == 0 || finish == 1) {
                sem_post(&inChargeforPickup);
                sem_post(&controlPickupParking);
                break;
            }
            mFree_pickup++;
            parkedPickup--;
            printf("Pickup parked in regular parking. Remaining spots in free parking: %d. Remaining Spots in regular parking: %d\n", mFree_pickup, parkedPickup);
            sem_post(&inChargeforPickup);
            sem_post(&controlPickupParking);
        }
    }
    pthread_exit(NULL);
}
```

6. Testing and Validation

Leak Check

No leaks. Just warning for still reachable memory.

```
==28== LEAK SUMMARY:
==28==    definitely lost: 0 bytes in 0 blocks
==28==    indirectly lost: 0 bytes in 0 blocks
==28==      possibly lost: 0 bytes in 0 blocks
==28==    still reachable: 1,654 bytes in 4 blocks
==28==         suppressed: 0 bytes in 0 blocks
==28==
==28== For lists of detected and suppressed errors, rerun with: -s
==28== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Test Results with 28 vehicles

```
Automobile parked in free parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 8
Automobile parked in free parking. Remaining spots in free parking: 6. Remaining Spots in regular parking: 8
Automobile parked in free parking. Remaining spots in free parking: 5. Remaining Spots in regular parking: 8
Automobile parked in free parking. Remaining spots in free parking: 4. Remaining Spots in regular parking: 8
Automobile parked in free parking. Remaining spots in free parking: 3. Remaining Spots in regular parking: 8
Pickup parked in free parking. Remaining spots in free parking: 3. Remaining Spots in regular parking: 4
Automobile parked in regular parking. Remaining spots in free parking: 4. Remaining Spots in regular parking: 7
Automobile parked in regular parking. Remaining spots in free parking: 5. Remaining Spots in regular parking: 6
Automobile parked in regular parking. Remaining spots in free parking: 6. Remaining Spots in regular parking: 5
Automobile parked in regular parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 4
Pickup parked in regular parking. Remaining spots in free parking: 4. Remaining Spots in regular parking: 3
Automobile parked in regular parking. Remaining spots in free parking: 8. Remaining Spots in regular parking: 3
Pickup parked in free parking. Remaining spots in free parking: 3. Remaining Spots in regular parking: 3
Pickup parked in regular parking. Remaining spots in free parking: 4. Remaining Spots in regular parking: 2
Automobile parked in free parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 3
Automobile parked in regular parking. Remaining spots in free parking: 8. Remaining Spots in regular parking: 2
Pickup parked in free parking. Remaining spots in free parking: 3. Remaining Spots in regular parking: 2
Pickup parked in regular parking. Remaining spots in free parking: 4. Remaining Spots in regular parking: 1
Automobile parked in free parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 2
Pickup parked in free parking. Remaining spots in free parking: 3. Remaining Spots in regular parking: 1
Automobile parked in regular parking. Remaining spots in free parking: 8. Remaining Spots in regular parking: 1
Pickup parked in regular parking. Remaining spots in free parking: 4. Remaining Spots in regular parking: 0
Pickup parked in free parking. Remaining spots in free parking: 3. Remaining Spots in regular parking: 0
Pickup parked in free parking. Remaining spots in free parking: 2. Remaining Spots in regular parking: 0
Regular parking is still full. Pickup is leaving from free parking. Remaining spots in free parking. 3
Pickup parked in free parking. Remaining spots in free parking: 2. Remaining Spots in regular parking: 0
Regular parking is still full. Pickup is leaving from free parking. Remaining spots in free parking. 3
Automobile parked in free parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 1
Automobile parked in regular parking. Remaining spots in free parking: 8. Remaining Spots in regular parking: 0
Pickup parked in free parking. Remaining spots in free parking: 2. Remaining Spots in regular parking: 0
Automobile parked in free parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 0
Pickup parked in free parking. Remaining spots in free parking: 1. Remaining Spots in regular parking: 0
Automobile parked in free parking. Remaining spots in free parking: 6. Remaining Spots in regular parking: 0
Regular parking is still full. Automobile is leaving from free parking. Remaining spots in free parking 7
Automobile parked in free parking. Remaining spots in free parking: 6. Remaining Spots in regular parking: 0
Regular parking is still full. Automobile is leaving from free parking. Remaining spots in free parking 7
Pickup parked in free parking. Remaining spots in free parking: 0. Remaining Spots in regular parking: 0
Automobile parked in free parking. Remaining spots in free parking: 6. Remaining Spots in regular parking: 0
No space for another pickup in free parking.
No space for another pickup in free parking.
No space for another pickup in free parking.
No space for another pickup in free parking.
Automobile parked in free parking. Remaining spots in free parking: 5. Remaining Spots in regular parking: 0
No space for another pickup in free parking.
Regular parking is still full. Pickup is leaving from free parking. Remaining spots in free parking. 1
Regular parking is still full. Automobile is leaving from free parking. Remaining spots in free parking 6
Regular parking is still full. Automobile is leaving from free parking. Remaining spots in free parking 7
Regular parking is still full. Pickup is leaving from free parking. Remaining spots in free parking. 2
Regular parking is still full. Pickup is leaving from free parking. Remaining spots in free parking. 3
Regular parking is still full. Pickup is leaving from free parking. Remaining spots in free parking. 4
Regular parking is still full. Automobile is leaving from free parking. Remaining spots in free parking 8
```

Test Results with 20 vehicles

```
a@DESKTOP-5F6OAM7:/mnt/c/Users/akif_/Desktop/SystemPro/Homework 3/System-Programming-Semaphores-and-Threads$ make run
./main
Automobile parked in free parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 8
Pickup parked in free parking. Remaining spots in free parking: 3. Remaining Spots in regular parking: 4
Pickup parked in free parking. Remaining spots in free parking: 2. Remaining Spots in regular parking: 4
Automobile parked in regular parking. Remaining spots in free parking: 8. Remaining Spots in regular parking: 7
Automobile parked in free parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 7
Pickup parked in regular parking. Remaining spots in free parking: 3. Remaining Spots in regular parking: 3
Pickup parked in regular parking. Remaining spots in free parking: 4. Remaining Spots in regular parking: 2
Automobile parked in regular parking. Remaining spots in free parking: 8. Remaining Spots in regular parking: 6
Automobile parked in free parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 6
Automobile parked in free parking. Remaining spots in free parking: 6. Remaining Spots in regular parking: 6
Automobile parked in regular parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 5
Automobile parked in regular parking. Remaining spots in free parking: 8. Remaining Spots in regular parking: 4
Pickup parked in free parking. Remaining spots in free parking: 3. Remaining Spots in regular parking: 2
Pickup parked in regular parking. Remaining spots in free parking: 4. Remaining Spots in regular parking: 1
Pickup parked in free parking. Remaining spots in free parking: 3. Remaining Spots in regular parking: 1
Pickup parked in regular parking. Remaining spots in free parking: 4. Remaining Spots in regular parking: 0
Automobile parked in free parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 4
Automobile parked in regular parking. Remaining spots in free parking: 8. Remaining Spots in regular parking: 3
Pickup parked in free parking. Remaining spots in free parking: 3. Remaining Spots in regular parking: 0
Pickup parked in free parking. Remaining spots in free parking: 2. Remaining Spots in regular parking: 0
Pickup parked in free parking. Remaining spots in free parking: 1. Remaining Spots in regular parking: 0
Automobile parked in free parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 3
Automobile parked in regular parking. Remaining spots in free parking: 8. Remaining Spots in regular parking: 2
Automobile parked in free parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 2
Automobile parked in regular parking. Remaining spots in free parking: 8. Remaining Spots in regular parking: 1
Automobile parked in free parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 1
Automobile parked in regular parking. Remaining spots in free parking: 8. Remaining Spots in regular parking: 0
Pickup parked in free parking. Remaining spots in free parking: 0. Remaining Spots in regular parking: 0
Automobile parked in free parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 0
Regular parking is still full. Automobile is leaving from free parking. Remaining spots in free parking 8
No space for another pickup in free parking.
Automobile parked in free parking. Remaining spots in free parking: 7. Remaining Spots in regular parking: 0
Regular parking is still full. Automobile is leaving from free parking. Remaining spots in free parking 8
No space for another pickup in free parking.
Regular parking is still full. Pickup is leaving from free parking. Remaining spots in free parking. 1
Regular parking is still full. Pickup is leaving from free parking. Remaining spots in free parking. 2
Regular parking is still full. Pickup is leaving from free parking. Remaining spots in free parking. 3
Regular parking is still full. Pickup is leaving from free parking. Remaining spots in free parking. 4
```