Eren Torlak 210104004090

Parking Lot Simulation Program

GENERAL SUMMARY :

There are temp parking lots for 4 pickup and 8 car .

In random time and random kind of vehicles(car or pickup) cames .

If temp parking lots are full owners immediately leave.

If not full places car to temp car lot.

Then if it is car, car valet cames and takes the car from temp lot . This process lasts 2 seconds.
Then if it is pickup, pickup valet cames and takes the pickup from temp lot . This process lasts 2 seconds.

1. **Constants and Shared Variables**: Defines parking lot capacities and shared memory counters.
2. **Semaphores and Mutex**: Initializes synchronization primitives.
3. **Thread Functions**: Defines behavior for vehicle owners and attendants.
4. **Main Function**: Orchestrates thread creation, synchronization, and cleanup.

Semaphores

Semaphores are used to synchronize vehicle owners and attendants:

1. **newPickup**: Signaled when a new pickup arrives.
2. **inChargeforPickup**: Signaled when an attendant has parked a pickup.
3. **newAutomobile**: Signaled when a new automobile arrives.
4. **inChargeforAutomobile**: Signaled when an attendant has parked an automobile.

Initialization:

```c
sem_init(&newPickup, 0, 0);
sem_init(&inChargeforPickup, 0, 0);
sem_init(&newAutomobile, 0, 0);
sem_init(&inChargeforAutomobile, 0, 0);
```

Mutex

A mutex is used to protect the shared counters `mFree_automobile` and `mFree_pickup`:

Mutex Initilazed like :
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

Eren Torlak 210104004090

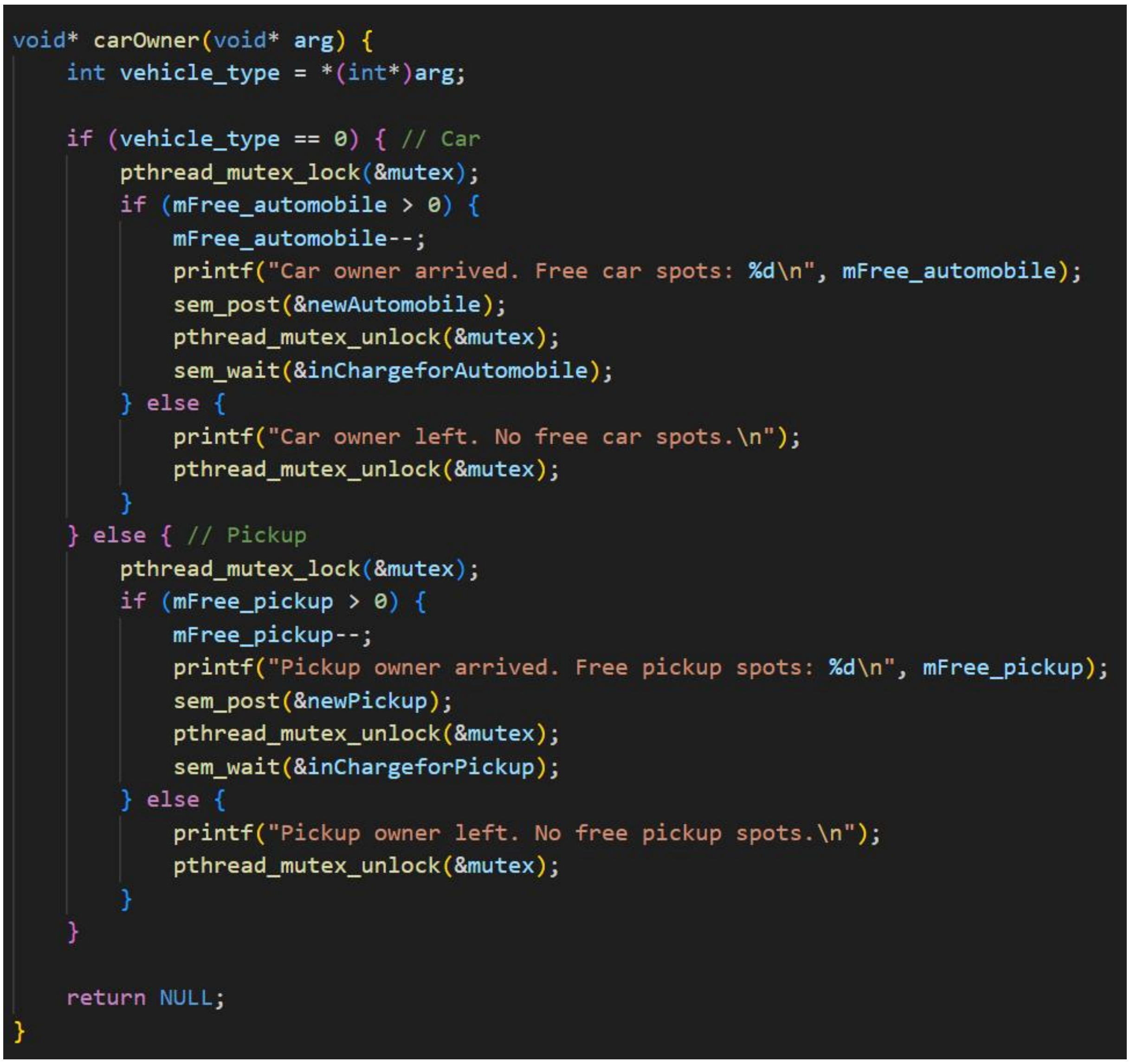
Shared Counters

The shared counters `mFree_automobile` and `mFree_pickup` track the available spots in the parking lot for automobiles and pickups, respectively.

Thread Functions

Car Owner

The `carOwner` function simulates a vehicle owner trying to park:

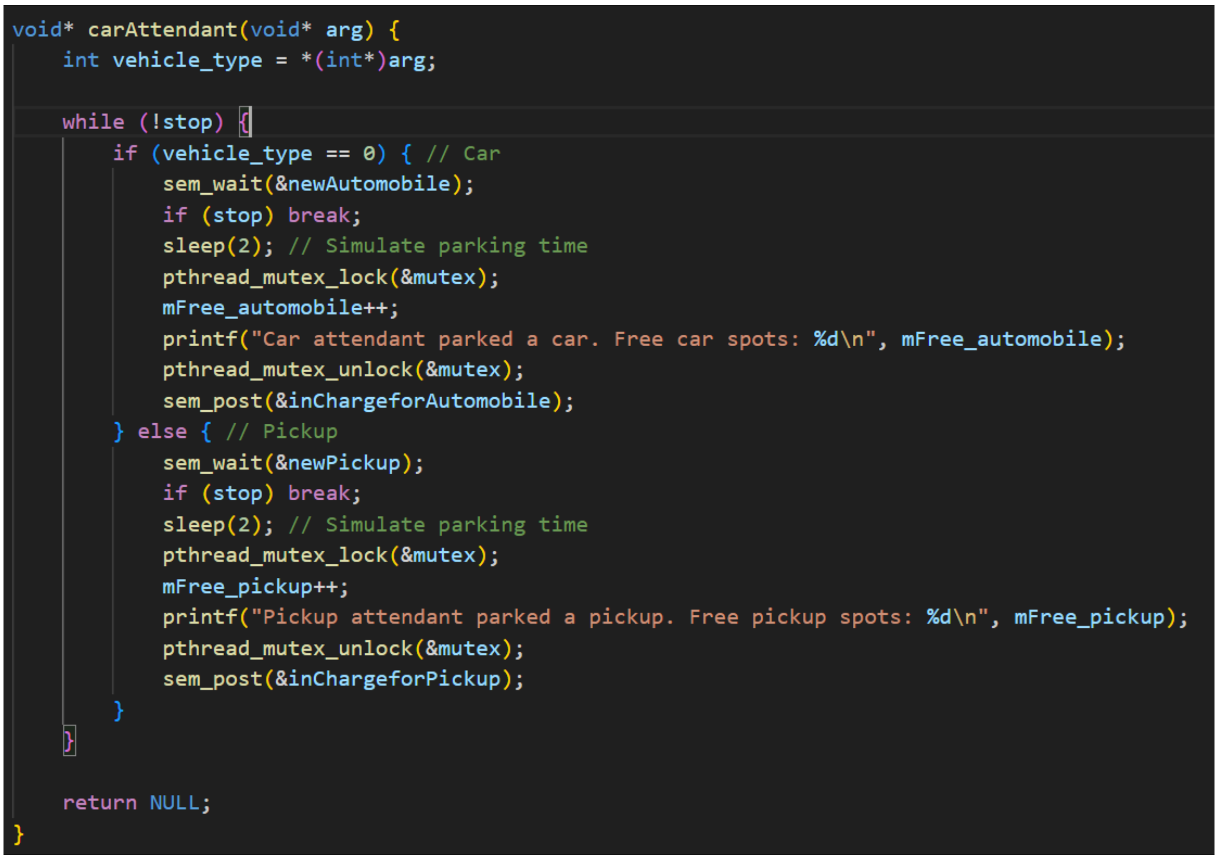


```c
void* carOwner(void* arg) {
    int vehicle_type = *(int*)arg;

    if (vehicle_type == 0) { // Car
        pthread_mutex_lock(&mutex);
        if (mFree_automobile > 0) {
            mFree_automobile--;
            printf("Car owner arrived. Free car spots: %d\n", mFree_automobile);
            sem_post(&newAutomobile);
            pthread_mutex_unlock(&mutex);
            sem_wait(&inChargeforAutomobile);
        } else {
            printf("Car owner left. No free car spots.\n");
            pthread_mutex_unlock(&mutex);
        }
    } else { // Pickup
        pthread_mutex_lock(&mutex);
        if (mFree_pickup > 0) {
            mFree_pickup--;
            printf("Pickup owner arrived. Free pickup spots: %d\n", mFree_pickup);
            sem_post(&newPickup);
            pthread_mutex_unlock(&mutex);
            sem_wait(&inChargeforPickup);
        } else {
            printf("Pickup owner left. No free pickup spots.\n");
            pthread_mutex_unlock(&mutex);
        }
    }

    return NULL;
}
```

Car Attendant

The `carAttendant` function simulates an attendant parking vehicles:

Eren Torlak 210104004090



```c
void* carAttendant(void* arg) {
    int vehicle_type = *(int*)arg;

    while (!stop) {
        if (vehicle_type == 0) { // Car
            sem_wait(&newAutomobile);
            if (stop) break;
            sleep(2); // Simulate parking time
            pthread_mutex_lock(&mutex);
            mFree_automobile++;
            printf("Car attendant parked a car. Free car spots: %d\n", mFree_automobile);
            pthread_mutex_unlock(&mutex);
            sem_post(&inChargeforAutomobile);
        } else { // Pickup
            sem_wait(&newPickup);
            if (stop) break;
            sleep(2); // Simulate parking time
            pthread_mutex_lock(&mutex);
            mFree_pickup++;
            printf("Pickup attendant parked a pickup. Free pickup spots: %d\n", mFree_pickup);
            pthread_mutex_unlock(&mutex);
            sem_post(&inChargeforPickup);
        }
    }

    return NULL;
}
```

Main Function

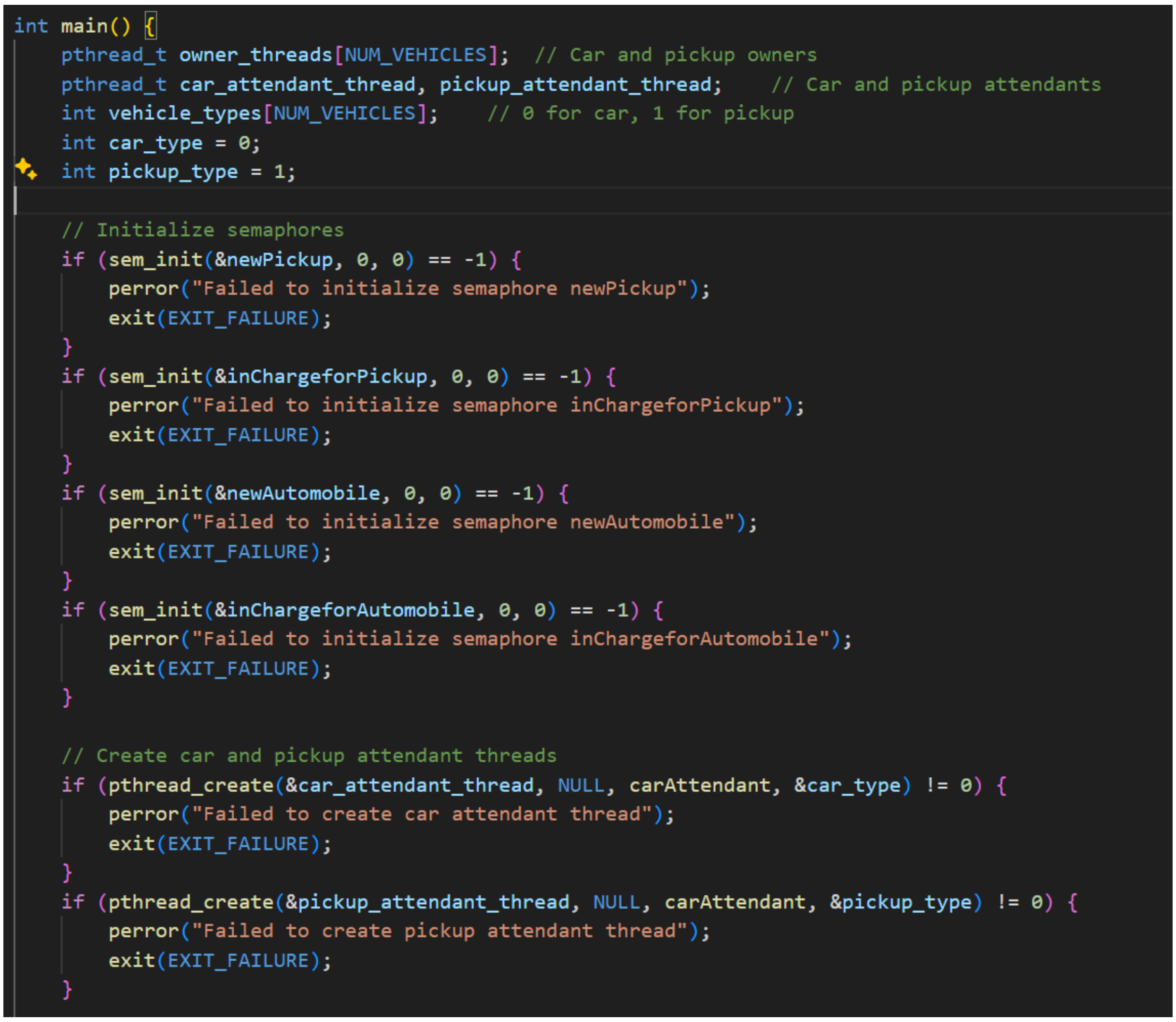
The `main` function initializes semaphores, creates threads, and handles synchronization:

There will be total of NUM_VEHİCLES that is predefined and can be changed for various experiments. It defines how many vehicle owner there will be .

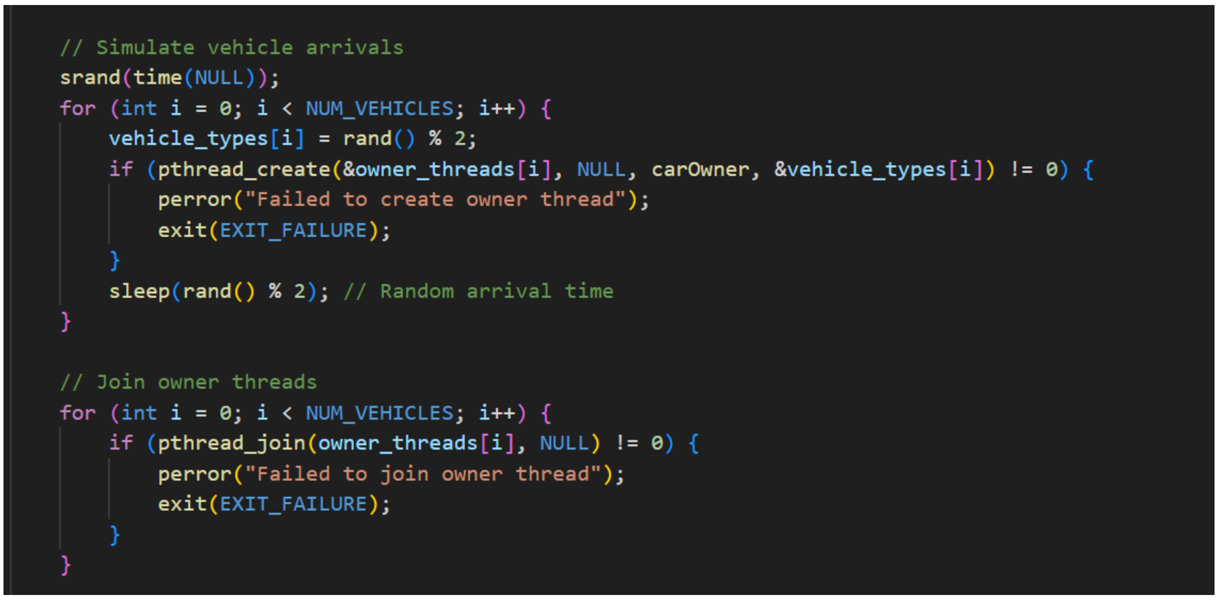
Owner, car attendant and pickup attendant threads are defined.

Later with sem_init four semaphores like in homework file is initialized.

Eren Torlak 210104004090



```c
int main() {
    pthread_t owner_threads[NUM_VEHICLES];  // Car and pickup owners
    pthread_t car_attendant_thread, pickup_attendant_thread;    // Car and pickup attendants
    int vehicle_types[NUM_VEHICLES];    // 0 for car, 1 for pickup
    int car_type = 0;
    int pickup_type = 1;

    // Initialize semaphores
    if (sem_init(&newPickup, 0, 0) == -1) {
        perror("Failed to initialize semaphore newPickup");
        exit(EXIT_FAILURE);
    }
    if (sem_init(&inChargeforPickup, 0, 0) == -1) {
        perror("Failed to initialize semaphore inChargeforPickup");
        exit(EXIT_FAILURE);
    }
    if (sem_init(&newAutomobile, 0, 0) == -1) {
        perror("Failed to initialize semaphore newAutomobile");
        exit(EXIT_FAILURE);
    }
    if (sem_init(&inChargeforAutomobile, 0, 0) == -1) {
        perror("Failed to initialize semaphore inChargeforAutomobile");
        exit(EXIT_FAILURE);
    }

    // Create car and pickup attendant threads
    if (pthread_create(&car_attendant_thread, NULL, carAttendant, &car_type) != 0) {
        perror("Failed to create car attendant thread");
        exit(EXIT_FAILURE);
    }
    if (pthread_create(&pickup_attendant_thread, NULL, carAttendant, &pickup_type) != 0) {
        perror("Failed to create pickup attendant thread");
        exit(EXIT_FAILURE);
    }
```



```c
    // Simulate vehicle arrivals
    srand(time(NULL));
    for (int i = 0; i < NUM_VEHICLES; i++) {
        vehicle_types[i] = rand() % 2;
        if (pthread_create(&owner_threads[i], NULL, carOwner, &vehicle_types[i]) != 0) {
            perror("Failed to create owner thread");
            exit(EXIT_FAILURE);
        }
        sleep(rand() % 2); // Random arrival time
    }

    // Join owner threads
    for (int i = 0; i < NUM_VEHICLES; i++) {
        if (pthread_join(owner_threads[i], NULL) != 0) {
            perror("Failed to join owner thread");
            exit(EXIT_FAILURE);
        }
    }
```

Eren Torlak 210104004090

How It Satisfies Homework Requirements

Two Parking Attendants

- The code creates two attendant threads, one for cars and one for pickups:
 ```c
 if (pthread_create(&car_attendant_thread, NULL, carAttendant, &car_type) != 0) {
    perror("Failed to create car attendant thread");
    exit(EXIT_FAILURE);
 }
 if (pthread_create(&pickup_attendant_thread, NULL, carAttendant, &pickup_type) != 0) {
    perror("Failed to create pickup attendant thread");
    exit(EXIT_FAILURE);
 }
 ```

Parking Lot Capacity

- The parking lot capacity is defined by `MAX_PICKUP_SPOTS = 4` and `MAX_AUTOMOBILE_SPOTS = 8` constants, and the shared counters `mFree_automobile` and `mFree_pickup` track the available spots.

Random Vehicle Generation

- Vehicles are generated randomly using `rand() % 2` to simulate arrivals of cars and pickups:
 ```c
 srand(time(NULL));
 for (int i = 0; i < NUM_VEHICLES; i++) {
    vehicle_types[i] = rand() % 2;
 }
 ```

Temporary Parking Lot

- Vehicle owners check the availability of parking spots before proceeding. If no spots are available, they leave:
 ```c
 if (mFree_automobile > 0) {
    mFree_automobile--;
    sem_post(&newAutomobile);
 } else {
    printf("Car owner left. No free car spots.\n");
 }
 ```

Occupancy Tracking

Eren Torlak 210104004090

- The counters `mFree_automobile` and

`mFree_pickup` track the occupancy of the parking spots. They are protected by a mutex to ensure thread safety.

Thread Creation

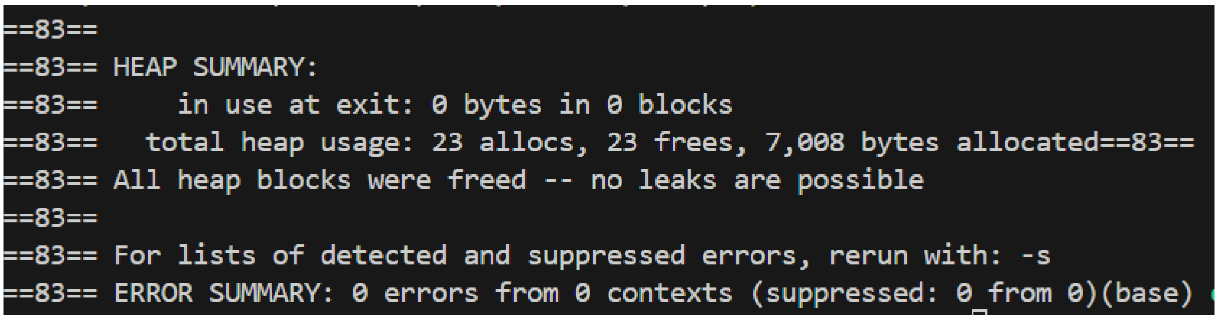
- Threads for vehicle owners and attendants are created and synchronized using semaphores and mutexes.

Error Handling and Graceful Stop

- The program includes error handling for thread creation and semaphore initialization. It also ensures a graceful stop of attendant threads by signaling semaphores.

##OUTPUTS

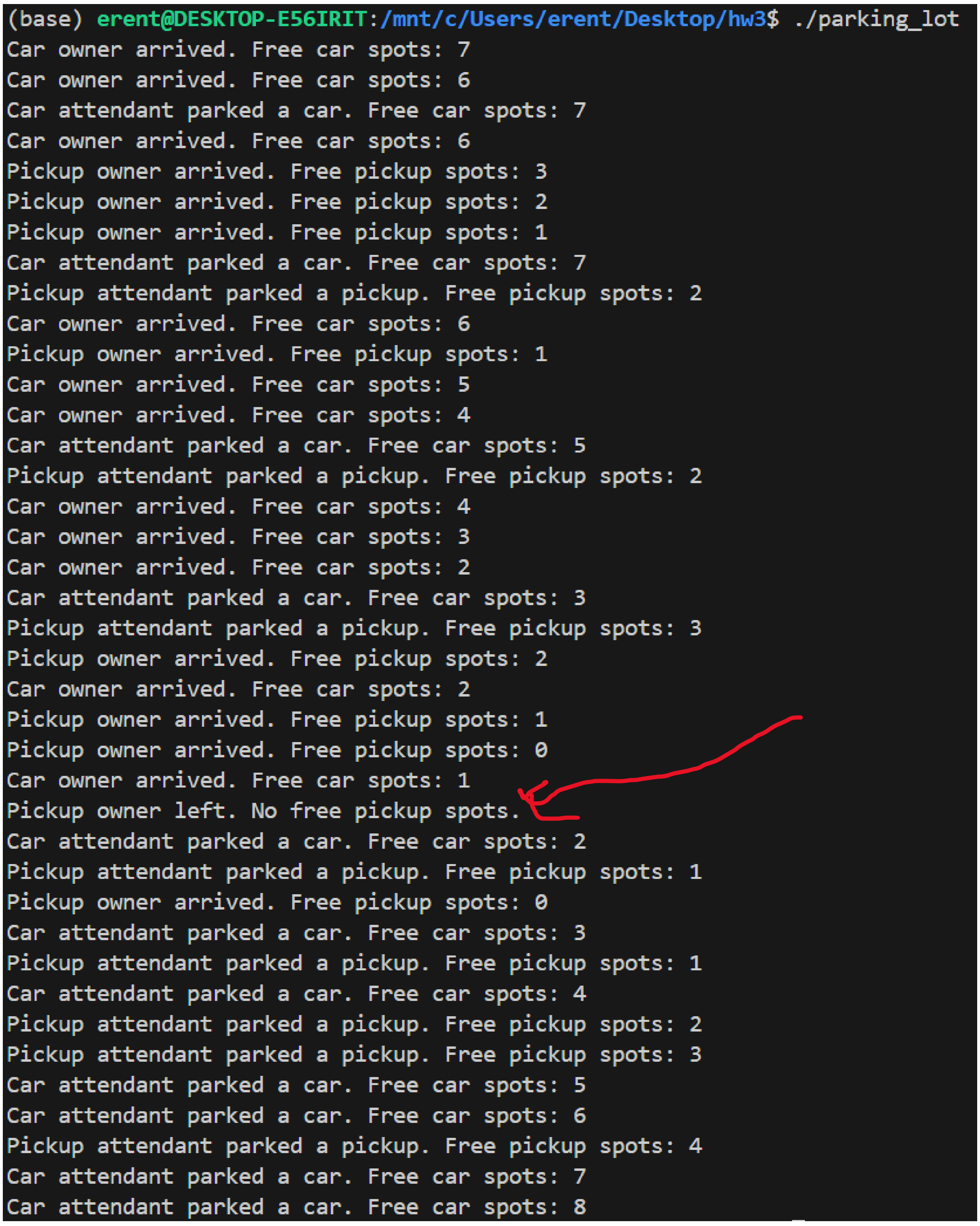
No memory leaks:



```
==83==
==83== HEAP SUMMARY:
==83==     in use at exit: 0 bytes in 0 blocks
==83==   total heap usage: 23 allocs, 23 frees, 7,008 bytes allocated==83==
==83== All heap blocks were freed -- no leaks are possible
==83==
==83== For lists of detected and suppressed errors, rerun with: -s
==83== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)(base)
```

If there are no parking place left it leaves immidetaly.

Eren Torlak 210104004090



```
(base) erent@DESKTOP-E56IRIT:/mnt/c/Users/erent/Desktop/hw3$ ./parking_lot
Car owner arrived. Free car spots: 7
Car owner arrived. Free car spots: 6
Car attendant parked a car. Free car spots: 7
Car owner arrived. Free car spots: 6
Pickup owner arrived. Free pickup spots: 3
Pickup owner arrived. Free pickup spots: 2
Pickup owner arrived. Free pickup spots: 1
Car attendant parked a car. Free car spots: 7
Pickup attendant parked a pickup. Free pickup spots: 2
Car owner arrived. Free car spots: 6
Pickup owner arrived. Free pickup spots: 1
Car owner arrived. Free car spots: 5
Car owner arrived. Free car spots: 4
Car attendant parked a car. Free car spots: 5
Pickup attendant parked a pickup. Free pickup spots: 2
Car owner arrived. Free car spots: 4
Car owner arrived. Free car spots: 3
Car owner arrived. Free car spots: 2
Car attendant parked a car. Free car spots: 3
Pickup attendant parked a pickup. Free pickup spots: 3
Pickup owner arrived. Free pickup spots: 2
Car owner arrived. Free car spots: 2
Pickup owner arrived. Free pickup spots: 1
Pickup owner arrived. Free pickup spots: 0
Car owner arrived. Free car spots: 1
Pickup owner left. No free pickup spots.
Car attendant parked a car. Free car spots: 2
Pickup attendant parked a pickup. Free pickup spots: 1
Pickup owner arrived. Free pickup spots: 0
Car attendant parked a car. Free car spots: 3
Pickup attendant parked a pickup. Free pickup spots: 1
Car attendant parked a car. Free car spots: 4
Pickup attendant parked a pickup. Free pickup spots: 2
Pickup attendant parked a pickup. Free pickup spots: 3
Car attendant parked a car. Free car spots: 5
Car attendant parked a car. Free car spots: 6
Pickup attendant parked a pickup. Free pickup spots: 4
Car attendant parked a car. Free car spots: 7
Car attendant parked a car. Free car spots: 8
```