Alper Tavşanoğlu-210104004142-HW-3-Report

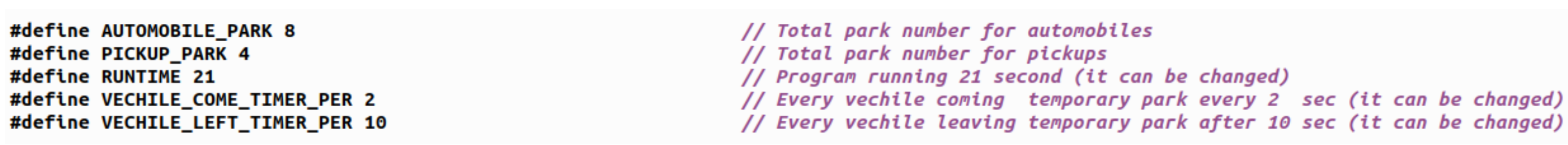
The purpose of this program is to simulate a parking lot system using Threads and Semaphores for synchronization. The parking system simulates to two types of vehicles which are automobiles and pickups, each managed by designated parking valets in a parking lot environment. (0 = For Automobiles, 1=Pickups)

The simulation controls vehicle entry and exit through semaphore synchronization, ensuring that each vehicle type is handled by a dedicated parking valet. The parking lot has separate sections for automobiles and pickups, with 8 place for automobiles and 4 for pickups. The simulation is designed to run for a predetermined period, during which vehicles attempt to park if spaces are available. If no place are available, the vehicle will leave.

Main Informations :

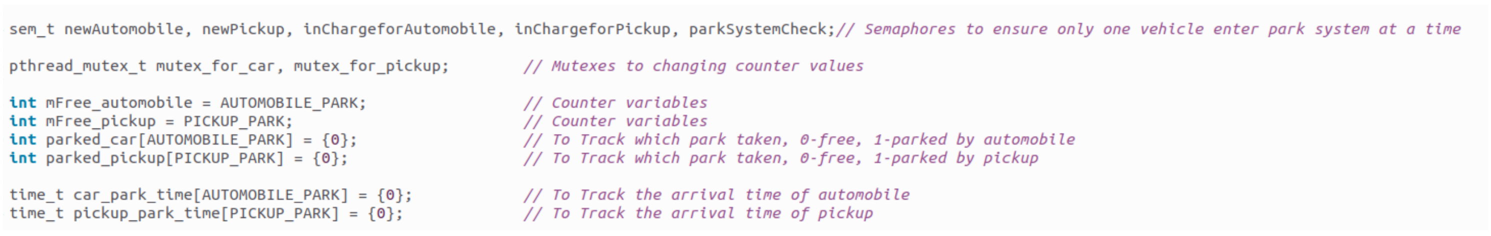
- We have two types of vehicles: automobiles and pickups.
- The parking lot has 8 place designated for automobiles and 4 place for pickups.
- By using semaphores we need to make sure that only one vehicle can enter the parking system at once.
- We need count empty parking places and the parking status of each places. Each vehicle is assigned a parking spot if available. If not, the vehicle leaves without parking.
- Since it is a temporary parking area, vehicles must leave the parking area after a while. (I developed my own system)
- Separate threads will simulate the behavior of vehicle owners and parking attendants, synchronized via semaphores.

First of all, I defined park with capacity 8 for automobiles and 4 for pickups. I created a system which is accepts random vehicles every 2 second (it can be changed i defined it on **line 11**). After that, every oldest car must left park 10 second later (it can be changed i defined it on **line 12**). So, for example one automobile/pickup came at 2nd second it must leave from park after 10 second later(which is 12th second). And if another automobile/pickup came 4th second it must leave from park after 10 second later(which is 14th second). I also defined program runtime for 21 second which means program will run for 21 second (it can be changed i defined it on **line 10**).



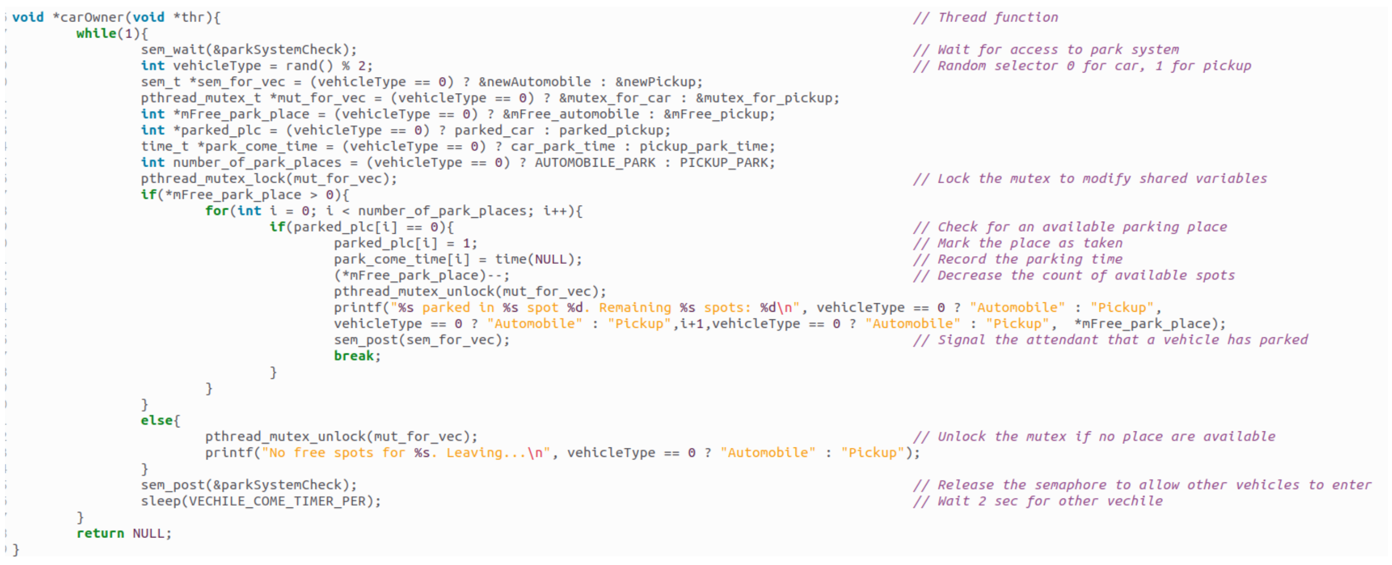
```
#define AUTOMOBILE_PARK 8                                  // Total park number for automobiles
#define PICKUP_PARK 4                                      // Total park number for pickups
#define RUNTIME 21                                         // Program running 21 second (it can be changed)
#define VECHILE_COME_TIMER_PER 2                           // Every vechile coming  temporary park every 2  sec (it can be changed)
#define VECHILE_LEFT_TIMER_PER 10                          // Every vechile leaving temporary park after 10 sec (it can be changed)
```

I initialized semaphores to ensure that shared resources are accessed in a synchronized manner. This includes semaphores for vehicle parking (newAutomobile, newPickup, inChargeforAutomobile, inChargeforPickup, parkSystemCheck) and i also initilazed mutexes for modifying the number of free parking spots (mutex_for_car, mutex_for_pickup). And also for checking times of vehicles (car_park_time, pickup_park_time). entryGate, Controls the entry of vehicles to ensure that only one vehicle attempts to park at a time.newAutomobile, newPickup, Signal when a new vehicle has been successfully parked.inChargeforAutomobile, inChargeforPickup, Indicating when an attendant starts processing a vehicle for departure. mutexAuto, mutexPickup: Protect the shared state, such as the count of free spots and the status of each parking spot.



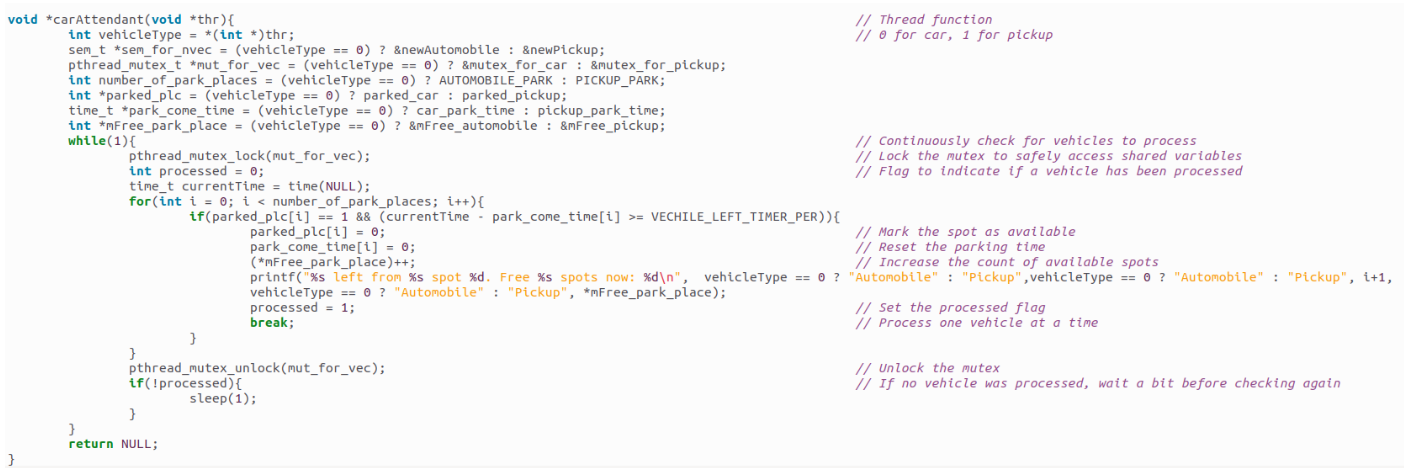
```
sem_t newAutomobile, newPickup, inChargeforAutomobile, inChargeforPickup, parkSystemCheck;// Semaphores to ensure only one vehicle enter park system at a time

pthread_mutex_t mutex_for_car, mutex_for_pickup;        // Mutexes to changing counter values

int mFree_automobile = AUTOMOBILE_PARK;                 // Counter variables
int mFree_pickup = PICKUP_PARK;                         // Counter variables
int parked_car[AUTOMOBILE_PARK] = {0};                  // To Track which park taken, 0-free, 1-parked by automobile
int parked_pickup[PICKUP_PARK] = {0};                   // To Track which park taken, 0-free, 1-parked by pickup

time_t car_park_time[AUTOMOBILE_PARK] = {0};            // To Track the arrival time of automobile
time_t pickup_park_time[PICKUP_PARK] = {0};             // To Track the arrival time of pickup
```

With thread function carOwner, vehicle owners can attempt to park their vehicles. It Simulates the behavior of vehicle owners trying to park in the parking lot. It randomly selects a vehicle type and tries to park it if a spot is available. Each invocation represents a vehicle arriving at the lot. It checks for available parking space. If found, it parks; otherwise, it exits the lot. Uses the parkSystemCheck semaphore to ensure exclusive access to checking and updating parking spaces.



```
void *carOwner(void *thr){                                                              // Thread function
        while(1){
                sem_wait(&parkSystemCheck);                                             // Wait for access to park system
                int vehicleType = rand() % 2;                                           // Random selector 0 for car, 1 for pickup
                sem_t *sem_for_vec = (vehicleType == 0) ? &newAutomobile : &newPickup;
                pthread_mutex_t *mut_for_vec = (vehicleType == 0) ? &mutex_for_car : &mutex_for_pickup;
                int *mFree_park_place = (vehicleType == 0) ? &mFree_automobile : &mFree_pickup;
                int *parked_plc = (vehicleType == 0) ? parked_car : parked_pickup;
                time_t *park_come_time = (vehicleType == 0) ? car_park_time : pickup_park_time;
                int number_of_park_places = (vehicleType == 0) ? AUTOMOBILE_PARK : PICKUP_PARK;
                pthread_mutex_lock(mut_for_vec);                                        // Lock the mutex to modify shared variables
                if(*mFree_park_place > 0){
                        for(int i = 0; i < number_of_park_places; i++){
                                if(parked_plc[i] == 0){                                 // Check for an available parking place
                                        parked_plc[i] = 1;                              // Mark the place as taken
                                        park_come_time[i] = time(NULL);                 // Record the parking time
                                        (*mFree_park_place)--;                          // Decrease the count of available spots
                                        pthread_mutex_unlock(mut_for_vec);
                                        printf("%s parked in %s spot %d. Remaining %s spots: %d\n", vehicleType == 0 ? "Automobile" : "Pickup",
                                        vehicleType == 0 ? "Automobile" : "Pickup",i+1,vehicleType == 0 ? "Automobile" : "Pickup",  *mFree_park_place);
                                        sem_post(sem_for_vec);                          // Signal the attendant that a vehicle has parked
                                        break;
                                }
                        }
                }
                else{
                        pthread_mutex_unlock(mut_for_vec);                              // Unlock the mutex if no place are available
                        printf("No free spots for %s. Leaving...\n", vehicleType == 0 ? "Automobile" : "Pickup");
                }
                sem_post(&parkSystemCheck);                                             // Release the semaphore to allow other vehicles to enter
                sleep(VECHILE_COME_TIMER_PER);                                          // Wait 2 sec for other vechile
        }
        return NULL;
}
```

With thread function carAttendant, attendants can manage the parking and departure of vehicles. It acts as a parking attendant, managing the parked vehicles. Monitors parked vehicles and marks spaces as available after a vehicle leaves. Uses vehicle-specific mutexes to update parking status and the semaphore to signal when a vehicle has been processed.

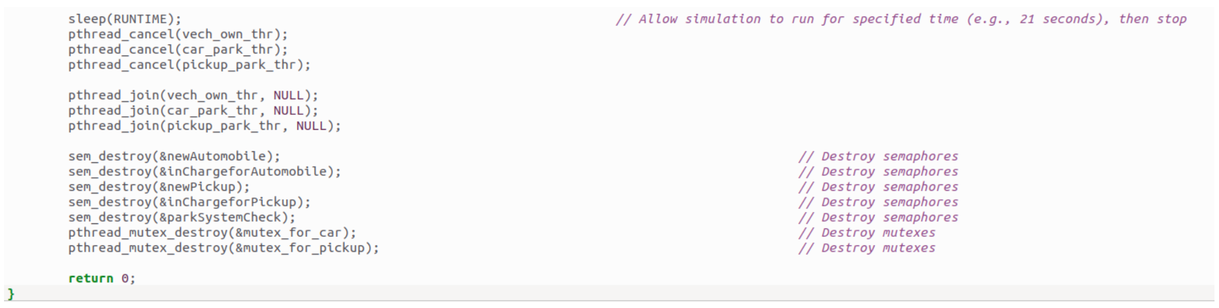


```c
void *carAttendant(void *thr){                                                                                          // Thread function
        int vehicleType = *(int *)thr;                                                                                  // 0 for car, 1 for pickup
        sem_t *sem_for_nvec = (vehicleType == 0) ? &newAutomobile : &newPickup;
        pthread_mutex_t *mut_for_vec = (vehicleType == 0) ? &mutex_for_car : &mutex_for_pickup;
        int number_of_park_places = (vehicleType == 0) ? AUTOMOBILE_PARK : PICKUP_PARK;
        int *parked_plc = (vehicleType == 0) ? parked_car : parked_pickup;
        time_t *park_come_time = (vehicleType == 0) ? car_park_time : pickup_park_time;
        int *mFree_park_place = (vehicleType == 0) ? &mFree_automobile : &mFree_pickup;
        while(1){                                                                                                       // Continuously check for vehicles to process
                pthread_mutex_lock(mut_for_vec);                                                                        // Lock the mutex to safely access shared variables
                int processed = 0;                                                                                      // Flag to indicate if a vehicle has been processed
                time_t currentTime = time(NULL);
                for(int i = 0; i < number_of_park_places; i++){
                        if(parked_plc[i] == 1 && (currentTime - park_come_time[i] >= VECHILE_LEFT_TIMER_PER)){
                                parked_plc[i] = 0;                                                                      // Mark the spot as available
                                park_come_time[i] = 0;                                                                  // Reset the parking time
                                (*mFree_park_place)++;                                                                  // Increase the count of available spots
                                printf("%s left from %s spot %d. Free %s spots now: %d\n",  vehicleType == 0 ? "Automobile" : "Pickup",vehicleType == 0 ? "Automobile" : "Pickup", i+1,
                                vehicleType == 0 ? "Automobile" : "Pickup", *mFree_park_place);
                                processed = 1;                                                                          // Set the processed flag
                                break;                                                                                  // Process one vehicle at a time
                        }
                }
                pthread_mutex_unlock(mut_for_vec);                                                                      // Unlock the mutex
                if(!processed){                                                                                         // If no vehicle was processed, wait a bit before checking again
                        sleep(1);
                }
        }
        return NULL;
}
```

Handling concurrency with multiple threads was a significant challenge, especially ensuring that vehicle processing did not result in race conditions. Semaphores and mutexes were crucial in this regard.

Ensuring that each vehicle stayed in the lot for exactly 10 seconds required precise control over thread execution, which was achieved through careful semaphore management.

After the simulation runs for a specified duration (21 seconds which is defined **line 10**), the program cancels all threads and joins them, ensuring they have finished executing. It then cleans up by destroying all initialized semaphores and mutexes.



```c
        sleep(RUNTIME);                                                 // Allow simulation to run for specified time (e.g., 21 seconds), then stop
        pthread_cancel(vech_own_thr);
        pthread_cancel(car_park_thr);
        pthread_cancel(pickup_park_thr);

        pthread_join(vech_own_thr, NULL);
        pthread_join(car_park_thr, NULL);
        pthread_join(pickup_park_thr, NULL);

        sem_destroy(&newAutomobile);                                                        // Destroy semaphores
        sem_destroy(&inChargeforAutomobile);                                                // Destroy semaphores
        sem_destroy(&newPickup);                                                            // Destroy semaphores
        sem_destroy(&inChargeforPickup);                                                    // Destroy semaphores
        sem_destroy(&parkSystemCheck);                                                      // Destroy semaphores
        pthread_mutex_destroy(&mutex_for_car);                                              // Destroy mutexes
        pthread_mutex_destroy(&mutex_for_pickup);                                           // Destroy mutexes

        return 0;
}
```

Compile and Examples

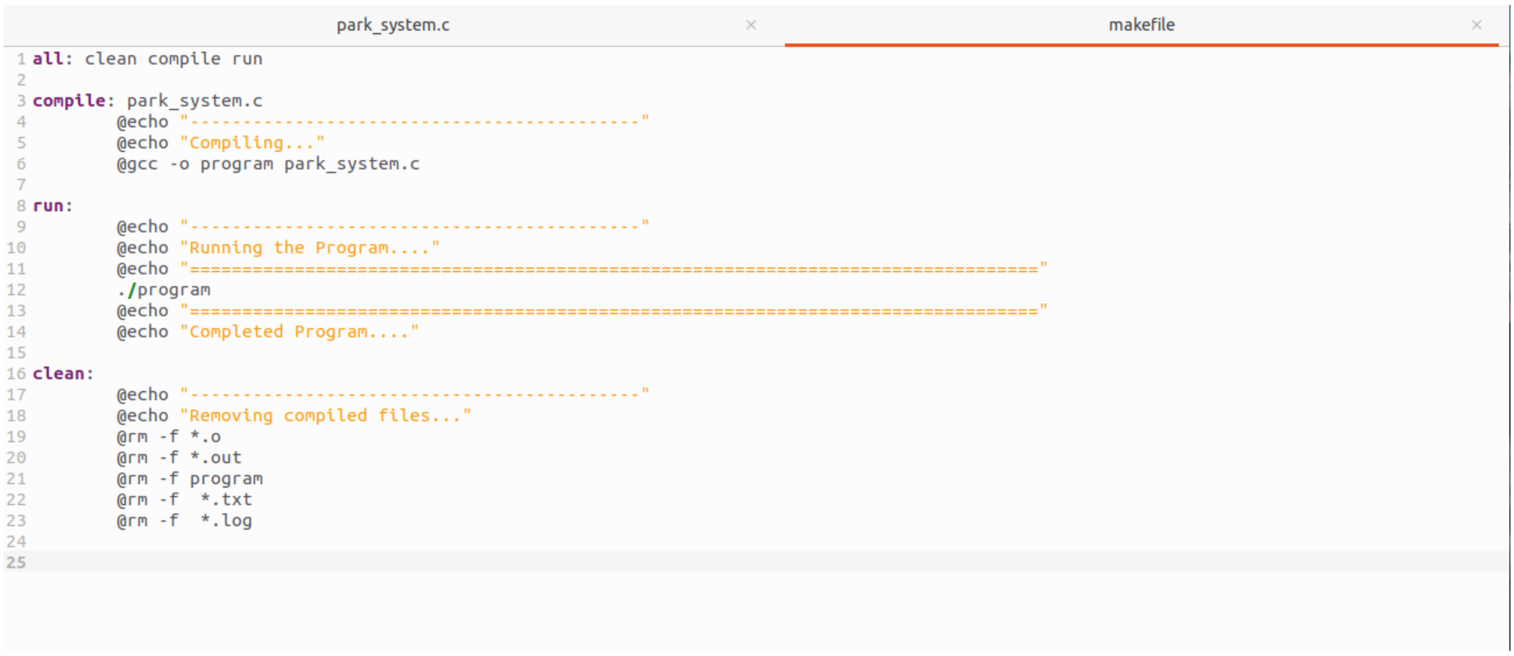
First Option

For running the code we can use makefile directly. With **make** command program runs automaticly for 21 second (it can be changed in code **line 10**). System will accept random vehicles every 2 second (it can be changed in code **line 11**). Parked vehicles will leave park after 10 seconds (it can be changed in code **line 12**).

Second Option

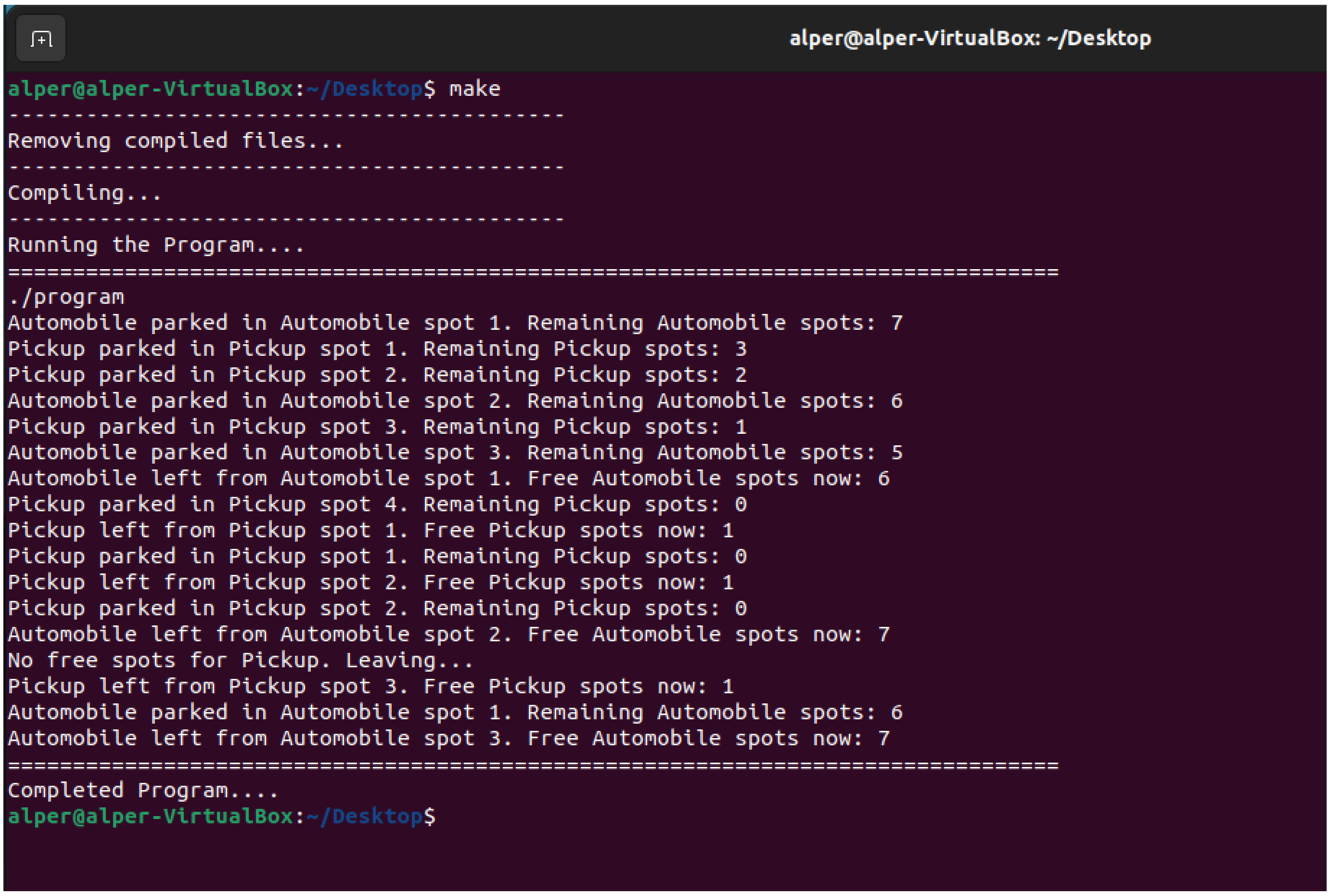
Or without using makefile we can compile with these commads first **gcc park_system.c** and **./a.out** .

Makefile also have **makeclean** command which removes everything except **c code** and **makefile**.



```
all: clean compile run

compile: park_system.c
        @echo "-------------------------------------------"
        @echo "Compiling..."
        @gcc -o program park_system.c

run:
        @echo "-------------------------------------------"
        @echo "Running the Program...."
        @echo "================================================================================="
        ./program
        @echo "================================================================================="
        @echo "Completed Program...."

clean:
        @echo "-------------------------------------------"
        @echo "Removing compiled files..."
        @rm -f *.o
        @rm -f *.out
        @rm -f program
        @rm -f  *.txt
        @rm -f  *.log

```

Examples on next page



alper@alper-VirtualBox: ~/Desktop

```
alper@alper-VirtualBox:~/Desktop$ make
-------------------------------------------
Removing compiled files...
-------------------------------------------
Compiling...
-------------------------------------------
Running the Program....
=================================================================================
./program
Automobile parked in Automobile spot 1. Remaining Automobile spots: 7
Pickup parked in Pickup spot 1. Remaining Pickup spots: 3
Pickup parked in Pickup spot 2. Remaining Pickup spots: 2
Automobile parked in Automobile spot 2. Remaining Automobile spots: 6
Pickup parked in Pickup spot 3. Remaining Pickup spots: 1
Automobile parked in Automobile spot 3. Remaining Automobile spots: 5
Automobile left from Automobile spot 1. Free Automobile spots now: 6
Pickup parked in Pickup spot 4. Remaining Pickup spots: 0
Pickup left from Pickup spot 1. Free Pickup spots now: 1
Pickup parked in Pickup spot 1. Remaining Pickup spots: 0
Pickup left from Pickup spot 2. Free Pickup spots now: 1
Pickup parked in Pickup spot 2. Remaining Pickup spots: 0
Automobile left from Automobile spot 2. Free Automobile spots now: 7
No free spots for Pickup. Leaving...
Pickup left from Pickup spot 3. Free Pickup spots now: 1
Automobile parked in Automobile spot 1. Remaining Automobile spots: 6
Automobile left from Automobile spot 3. Free Automobile spots now: 7
=================================================================================
Completed Program....
alper@alper-VirtualBox:~/Desktop$
```

Explanation of this example:

1) 0th second (when program started) Auto came spot 1. (will leave 10th second)(7 left)

2) 2nd second Pickup came spot 1. (will leave 12th second) (remaining pickup spot 3)

3) 4th second Pickup came spot 2. (will leave 14th second) (remaining pickup spot 2)

4) 6th second Auto came spot 2. (will leave 16th second) (remaining auto spot 6)

5) 8th second Pickup came spot 3. (will leave 18th second) (remaining pickup spot 1)

6) 10th second Auto came spot 3. (will leave 20th second) (remaining auto spot 5)

   10th second Auto left from spot 1. (0th second gone) (remaining auto spot 6)

7) 12th second Pickup came spot 4. (will leave 22th second)(reamining pickup spot 0)

   12th second Pickup left from spot 1.(2th second gone)(remaining pickup spot 1)

8) 14th second Pickup came spot 1. (will leave 24th second)(remaining pickup spot 0)

   14th second Pickup left from spot 2. (4th second gone)(remaining pickup spot 1)

9) 16th second Pickup came spot 2.(will leave 26th second)(remaining pickup spot 0)

   16th second auto left from spot 2.(6th second gone)(remaining auto spot 7)

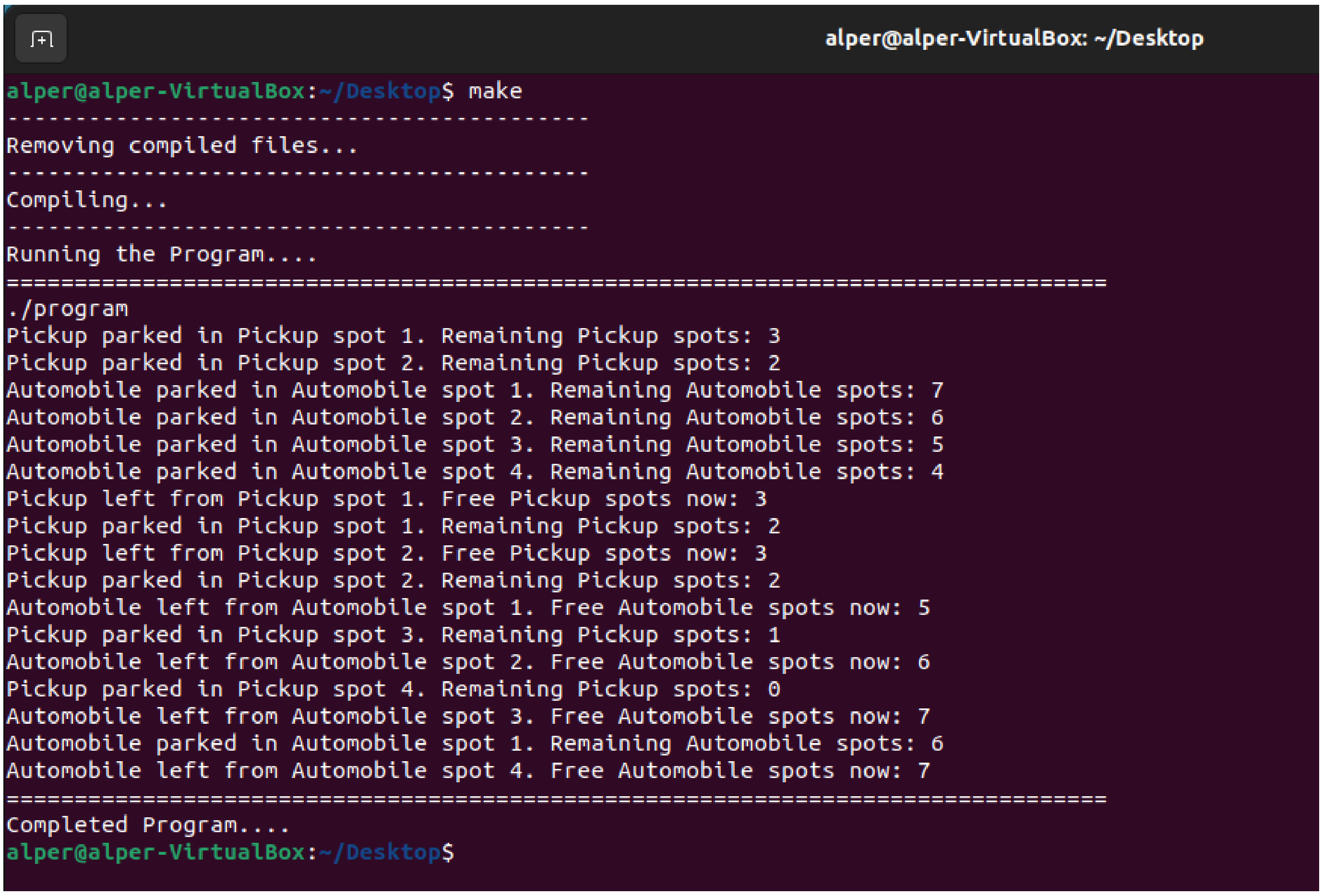
10) 18th second Pickup came but there is no spot. Leaving…

    18th second Pickup left from spot 3.(8th second gone)(remaining pickup spot 1)

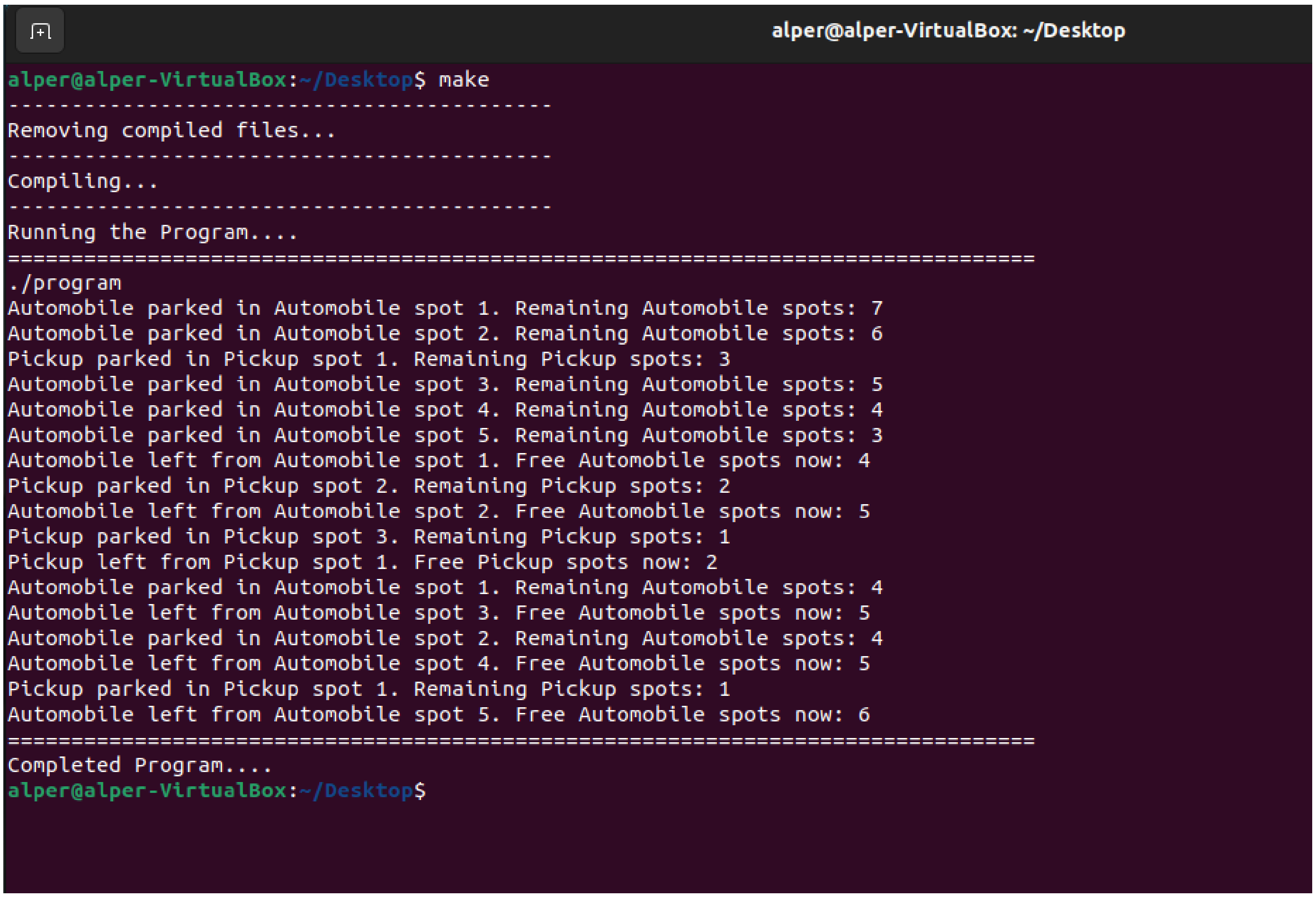
11) 20th second auto came spot 1. (will leave 30th second)(remaining auto spot 6)

    20th second auto left from spot 3. (10th second gone)(remaining auto spot 7)

And Another examples



```
alper@alper-VirtualBox:~/Desktop$ make
-------------------------------------------
Removing compiled files...
-------------------------------------------
Compiling...
-------------------------------------------
Running the Program....
=================================================================================
./program
Pickup parked in Pickup spot 1. Remaining Pickup spots: 3
Pickup parked in Pickup spot 2. Remaining Pickup spots: 2
Automobile parked in Automobile spot 1. Remaining Automobile spots: 7
Automobile parked in Automobile spot 2. Remaining Automobile spots: 6
Automobile parked in Automobile spot 3. Remaining Automobile spots: 5
Automobile parked in Automobile spot 4. Remaining Automobile spots: 4
Pickup left from Pickup spot 1. Free Pickup spots now: 3
Pickup parked in Pickup spot 1. Remaining Pickup spots: 2
Pickup left from Pickup spot 2. Free Pickup spots now: 3
Pickup parked in Pickup spot 2. Remaining Pickup spots: 2
Automobile left from Automobile spot 1. Free Automobile spots now: 5
Pickup parked in Pickup spot 3. Remaining Pickup spots: 1
Automobile left from Automobile spot 2. Free Automobile spots now: 6
Pickup parked in Pickup spot 4. Remaining Pickup spots: 0
Automobile left from Automobile spot 3. Free Automobile spots now: 7
Automobile parked in Automobile spot 1. Remaining Automobile spots: 6
Automobile left from Automobile spot 4. Free Automobile spots now: 7
=================================================================================
Completed Program....
alper@alper-VirtualBox:~/Desktop$
```



```
alper@alper-VirtualBox:~/Desktop$ make
-------------------------------------------
Removing compiled files...
-------------------------------------------
Compiling...
-------------------------------------------
Running the Program....
=================================================================================
./program
Automobile parked in Automobile spot 1. Remaining Automobile spots: 7
Automobile parked in Automobile spot 2. Remaining Automobile spots: 6
Pickup parked in Pickup spot 1. Remaining Pickup spots: 3
Automobile parked in Automobile spot 3. Remaining Automobile spots: 5
Automobile parked in Automobile spot 4. Remaining Automobile spots: 4
Automobile parked in Automobile spot 5. Remaining Automobile spots: 3
Automobile left from Automobile spot 1. Free Automobile spots now: 4
Pickup parked in Pickup spot 2. Remaining Pickup spots: 2
Automobile left from Automobile spot 2. Free Automobile spots now: 5
Pickup parked in Pickup spot 3. Remaining Pickup spots: 1
Pickup left from Pickup spot 1. Free Pickup spots now: 2
Automobile parked in Automobile spot 1. Remaining Automobile spots: 4
Automobile left from Automobile spot 3. Free Automobile spots now: 5
Automobile parked in Automobile spot 2. Remaining Automobile spots: 4
Automobile left from Automobile spot 4. Free Automobile spots now: 5
Pickup parked in Pickup spot 1. Remaining Pickup spots: 1
Automobile left from Automobile spot 5. Free Automobile spots now: 6
=================================================================================
Completed Program....
alper@alper-VirtualBox:~/Desktop$
```

Final notes and Conclusion

I ensured the program compiled without errors using GCC.
The parking lot simulation successfully demonstrates the use of multithreading and semaphore synchronization in a practical scenario.
The simulation handled multiple vehicle types, synchronized the activities of car owners and attendants, and managed the limited parking space efficiently.
The project compiled without errors, and extensive testing confirmed that the logic for vehicle parking and departure was functioning as intended, with all resources being cleaned up correctly after execution. The system handled all specified scenarios gracefully, providing a robust demonstration of threading and synchronization in system programming.
All resources are properly managed, and no memory leaks or resource leaks were observed. I used the valgrind tool to confirm that no memory leaks observed.

Program performed these steps:

- There are separate spots in the parking lot for cars and pickups.
- A random number is generated at the entrance (odd: car, even: pickup).
- Only one vehicle can enter the system at a time.
- carOwner() represents the vehicle owner, and carAttendant() represents the valet.
- Threads are synchronized using semaphores.
- Owners must confirm their vehicle's parking availability.
- The occupancy of the temporary parking area and vehicle types are handled correctly.
- Semaphores are initialized and used correctly.