CSE344 SYSTEMS PROGRAMMING

HOMEWORK 3

REPORT

Burak Demirkaya

210104004274

16/05/2024

1. Design Decisions

In this homework, I used makefile to run the program with valgrind to check for the memory leaks during the runtime of the program. By typing “make all” at the terminal the program will compile and run with valgrind option.

```
#define AUTOMOBILE_TEMP_SPOTS 8
#define PICKUP_TEMP_SPOTS 4

// Semaphores
sem_t newPickup, newAutomobile;
sem_t inChargeforPickup, inChargeforAutomobile;
pthread_mutex_t newCar;

// Counters for free spots
int mFree_automobile = AUTOMOBILE_TEMP_SPOTS;
int mFree_pickup = PICKUP_TEMP_SPOTS;
```

I have defined the semaphores, mutex and the counter variables.

From what I understand from the homework PDF is that **there are 4 and 8 temporary parking spaces** for the corresponding cars and **the final permanent parking space will be infinite**. So I have implemented my program depending on this decision.

```
pthread_t carOwners[20]; // Car owners
pthread_t valetForPickup, valetForAutomobile; // 2 valets
```

I have defined 20 threads for the car owners and 2 different car attendant threads for 2 different car types.

```
// Initialize semaphores and mutex
if(sem_init(&newPickup, 0, PICKUP_TEMP_SPOTS) != 0 ||
    sem_init(&newAutomobile, 0, AUTOMOBILE_TEMP_SPOTS) != 0 ||
    sem_init(&inChargeforPickup, 0, 0) != 0 ||
    sem_init(&inChargeforAutomobile, 0, 0) != 0 ||
    pthread_mutex_init(&newCar, NULL) != 0){
        perror("Initilazation failed");
        return 1;
}
```

I have initialized semaphores to manage synchronization between threads. Additionally, I have defined a mutex named newCar to ensure that only one car can enter the parking system at a time.

```c
// Create two different attendant threads for the two different types of cars
int pickup = 0, automobile = 1;
if(pthread_create(&valetForPickup, NULL, carAttendant, &pickup) != 0 ||
    pthread_create(&valetForAutomobile, NULL, carAttendant, &automobile) != 0){
    perror("pthread_create for vallets failed");
    return 1;
}

// Create threads for car owners
for(int i = 0; i < 20; i++){
    if(pthread_create(&carOwners[i], NULL, carOwner, NULL) != 0){
        perror("pthread_create for car owners failed");
        return 1;
    }
}

// Wait for all threads to finish
for(int i = 0; i < 20; i++){
    if(pthread_join(carOwners[i], NULL) != 0){
        perror("pthread_join failed");
        return 1;
    }
}
```

I have created two separate threads for car attendants, each receiving a specific car type as an argument. This ensures that each attendant handles a specific type of car. Additionally, I have created 20 threads to simulate car owners. I have used pthread_join to ensure that the main thread waits for all car owner threads to exit.

```c
// Handle remaining cars in the temporary parking lot
int remainingPickupCars = PICKUP_TEMP_SPOTS - mFree_pickup;
printf("Remaining pickups in temporary parking lot: %d\n", remainingPickupCars);
int remainingAutomobileCars = AUTOMOBILE_TEMP_SPOTS - mFree_automobile;
printf("Remaining automobiles in temporary parking lot: %d\n", remainingAutomobileCars);

while(remainingPickupCars > 0 || remainingAutomobileCars > 0){
    if(remainingPickupCars >= 0){
        if(sem_post(&inChargeforPickup) != 0){
            perror("sem_post");
            return 1;
        }
        remainingPickupCars--;
    }
    if(remainingAutomobileCars >= 0){
        if(sem_post(&inChargeforAutomobile) != 0){
            perror("sem_post");
            return 1;
        }
        remainingAutomobileCars--;
    }
}
```

I handled the remaining cars that are at the temporary parking lot after the car owner threads have finished by posting the corresponding semaphores.

```c
// Destroy semaphores and mutex
if(sem_destroy(&newPickup) != 0 ||
    sem_destroy(&newAutomobile) != 0 ||
    sem_destroy(&inChargeforPickup) != 0 ||
    sem_destroy(&inChargeforAutomobile) != 0 ||
    pthread_mutex_destroy(&newCar) != 0){
        perror("Destroy failed");
        return 1;
}
```

Finally inside main, I have destroyed each semaphores and mutex to clear the resources.

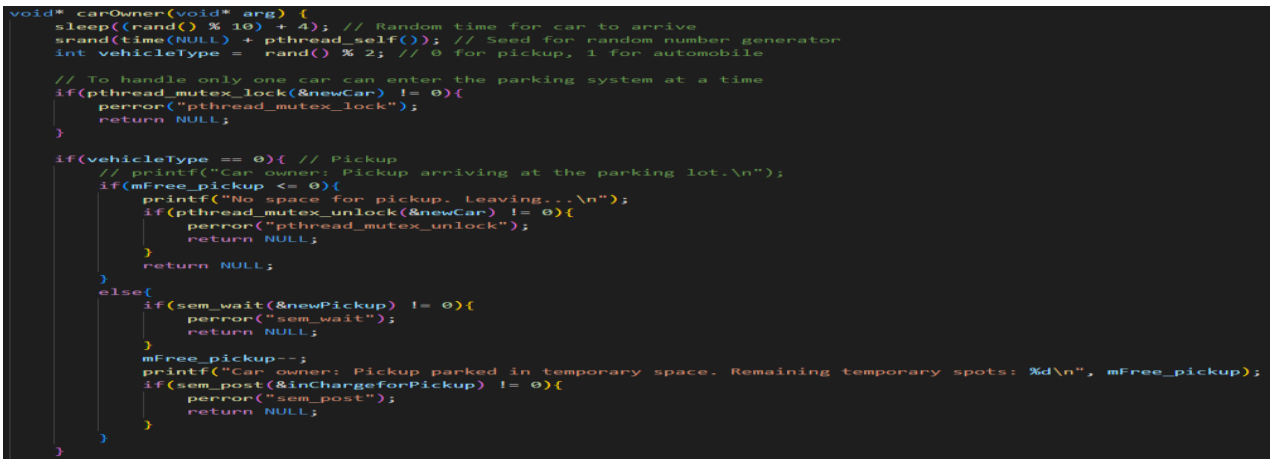
Car Owner Thread:

The car owner thread begins by generating a random value, either 0 or 1, which corresponds to the type of vehicle (0 for pickup, 1 for automobile).

If the vehicle is a pickup, the thread first checks if there is available space in the pickup parking area. If there is no space, the pickup leaves the parking lot. If there is space, the pickup parks in a temporary spot, decrements the count of available spots, and signals the pickup attendant semaphore to indicate that a pickup is waiting to be parked.

If the vehicle is an automobile (value 1), the thread performs a similar process, but for the automobile parking area and the automobile attendant semaphore.

Throughout this process, the thread uses a mutex lock to ensure that only one car owner can enter the parking system at a time. The mutex is locked before the vehicle type is checked and is unlocked after the vehicle has either left the parking lot or been parked in a temporary spot.



```c
void* carOwner(void* arg) {
    sleep((rand() % 10) + 4); // Random time for car to arrive
    srand(time(NULL) + pthread_self()); // Seed for random number generator
    int vehicleType =  rand() % 2; // 0 for pickup, 1 for automobile

    // To handle only one car can enter the parking system at a time
    if(pthread_mutex_lock(&newCar) != 0){
        perror("pthread_mutex_lock");
        return NULL;
    }

    if(vehicleType == 0){ // Pickup
        // printf("Car owner: Pickup arriving at the parking lot.\n");
        if(mFree_pickup <= 0){
            printf("No space for pickup. Leaving...\n");
            if(pthread_mutex_unlock(&newCar) != 0){
                perror("pthread_mutex_unlock");
                return NULL;
            }
            return NULL;
        }
        else{
            if(sem_wait(&newPickup) != 0){
                perror("sem_wait");
                return NULL;
            }
            mFree_pickup--;
            printf("Car owner: Pickup parked in temporary space. Remaining temporary spots: %d\n", mFree_pickup);
            if(sem_post(&inChargeforPickup) != 0){
                perror("sem_post");
                return NULL;
            }
        }
    }
```

```
    else if(vehicleType == 1){ // Automobile
        printf("Car owner: Automobile arriving at the parking lot.\n");
        if(mFree_automobile <= 0){
            printf("No space for automobile. Leaving...\n");
            if(pthread_mutex_unlock(&newCar) != 0){
                perror("pthread_mutex_unlock");
                return NULL;
            }
            return NULL;
        }
        else{
            if(sem_wait(&newAutomobile) != 0){
                perror("sem_wait");
                return NULL;
            }
            mFree_automobile--;
            printf("Car owner: Automobile parked in temporary space. Remaining temporary spots: %d\n", mFree_automobile);
            if(sem_post(&inChargeforAutomobile) != 0){
                perror("sem_post");
                return NULL;
            }
        }
    }
    if(pthread_mutex_unlock(&newCar) != 0){
        perror("pthread_mutex_unlock");
        return NULL;
    }
    return NULL;
}
```

Car Attendant Thread:

The car attendant thread begins by determining the type of car it is responsible for, based on the argument passed to it when it was created. This argument is either 0 or 1 corresponding to a pickup or an automobile, respectively. The thread then enters an infinite loop, where it waits for a car of its type to arrive. Once a car arrives, the thread increments the count of free final spots for its car type indicating that a car has been parked in a final spot. Finally, the thread signals the appropriate semaphore using the sem_post function, indicating that a temporary spot has been freed up. Once the car owner threads has finished I sent sem_post signal to handle the remaining cars at the temporary spot and break if there does not exist a car.

```
void* carAttendant(void* arg) {
    int carType = *((int*)arg);
    while(1){
        sleep((rand() % 5) + 1); // Random time for car to be parked
        if(carType == 0){ // Pickup
            if(sem_wait(&inChargeforPickup) != 0){
                perror("sem_wait");
                return NULL;
            }
            if(mFree_pickup >= PICKUP_TEMP_SPOTS){ // After all temporary spots are free, break the loop
                break;
            }
            mFree_pickup++;
            printf("Car attendant: Pickup has been parked in the final spot. Remaining temporary spots: %d\n", mFree_pickup);
            if(sem_post(&newPickup) != 0){
                perror("sem_post");
                return NULL;
            }
        }
        else if(carType == 1){ // Automobile
            if(sem_wait(&inChargeforAutomobile) != 0){
                perror("sem_wait");
                return NULL;
            }
            if(mFree_automobile >= AUTOMOBILE_TEMP_SPOTS){ // After all temporary spots are free, break the loop
                break;
            }
            mFree_automobile++;
            printf("Car attendant: Automobile has been parked in the final spot. Remaining temporary spots: %d\n", mFree_automobile);
            if(sem_post(&newAutomobile) != 0){
                perror("sem_post");
                return NULL;
            }
        }
    }
    return NULL;
}
```

2. Test Result

```
==1154== Command: ./park
==1154==
==1154== error calling PR_SET_PTRACER, vgdb might block
#########################################################
#               Welcome to the parking lot!             #
#########################################################
Car owner: Automobile parked in temporary space. Remaining temporary spots: 7
Car attendant: Automobile has been parked in the final spot. Remaining temporary spots: 8
Car owner: Automobile parked in temporary space. Remaining temporary spots: 7
Car owner: Pickup parked in temporary space. Remaining temporary spots: 3
Car attendant: Pickup has been parked in the final spot. Remaining temporary spots: 4
Car owner: Automobile parked in temporary space. Remaining temporary spots: 6
Car owner: Pickup parked in temporary space. Remaining temporary spots: 3
Car owner: Pickup parked in temporary space. Remaining temporary spots: 2
Car owner: Automobile parked in temporary space. Remaining temporary spots: 5
Car owner: Pickup parked in temporary space. Remaining temporary spots: 1
Car owner: Automobile parked in temporary space. Remaining temporary spots: 4
Car attendant: Pickup has been parked in the final spot. Remaining temporary spots: 2
Car attendant: Automobile has been parked in the final spot. Remaining temporary spots: 5
Car owner: Pickup parked in temporary space. Remaining temporary spots: 1
Car owner: Automobile parked in temporary space. Remaining temporary spots: 4
Car owner: Automobile parked in temporary space. Remaining temporary spots: 3
Car owner: Pickup parked in temporary space. Remaining temporary spots: 0
No space for pickup. Leaving...
Car owner: Automobile parked in temporary space. Remaining temporary spots: 2
Car owner: Automobile parked in temporary space. Remaining temporary spots: 1
Car attendant: Pickup has been parked in the final spot. Remaining temporary spots: 1
Car owner: Pickup parked in temporary space. Remaining temporary spots: 0
Car attendant: Automobile has been parked in the final spot. Remaining temporary spots: 2
No space for pickup. Leaving...
Car owner: Automobile parked in temporary space. Remaining temporary spots: 1
Car owner: Automobile parked in temporary space. Remaining temporary spots: 0
Remaining pickups in temporary parking lot: 4
Remaining automobiles in temporary parking lot: 8
Car attendant: Pickup has been parked in the final spot. Remaining temporary spots: 1
Car attendant: Pickup has been parked in the final spot. Remaining temporary spots: 2
Car attendant: Automobile has been parked in the final spot. Remaining temporary spots: 1
Car attendant: Automobile has been parked in the final spot. Remaining temporary spots: 2
Car attendant: Pickup has been parked in the final spot. Remaining temporary spots: 3
Car attendant: Automobile has been parked in the final spot. Remaining temporary spots: 3
Car attendant: Pickup has been parked in the final spot. Remaining temporary spots: 4
Car attendant: Automobile has been parked in the final spot. Remaining temporary spots: 4
Car attendant: Automobile has been parked in the final spot. Remaining temporary spots: 5
Car attendant: Automobile has been parked in the final spot. Remaining temporary spots: 6
Car attendant: Automobile has been parked in the final spot. Remaining temporary spots: 7
Car attendant: Automobile has been parked in the final spot. Remaining temporary spots: 8
==1154==
==1154== HEAP SUMMARY:
==1154==     in use at exit: 0 bytes in 0 blocks
==1154==   total heap usage: 23 allocs, 23 frees, 6,496 bytes allocated
==1154==
==1154== All heap blocks were freed -- no leaks are possible
==1154==
==1154== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

After all the car owner threads have completed their execution, the program calculates the number of remaining cars in the temporary parking lot for both pickup and automobile categories. It then prints these counts to the console. Next, the program enters a loop that continues until all cars in the temporary parking lot have been handled. If there are remaining pickup cars, it signals the pickup valet thread by posting to the inChargeforPickup semaphore and decrements the count of remaining pickup cars. Similarly, if there are remaining automobile cars, it signals the automobile valet thread by posting to the inChargeforAutomobile semaphore and decrements the count of remaining automobile cars.