CSE – 344 Homework 3 Report

Süleyman Burak Yaşar
1901042662

Content

1. Usage of Program

1. Makefile

There is a makefile.

make: compiles the program
make run: run program
make run2: run program with memory leak check
make clean: remove executable files

2. Initial Values

At the beginning of the program, there are initial values:
MAX_PICKUP: Number of pickup parking spaces
MAX_AUTOMOBILE: Number of automobile parking spaces
NUM_CAR_OWNERS: Number of car owners coming to the parking lot
RANDOM_ARRIVAL: Should car owners arrive at the parking lot at random times or at the same time. I will explain this value later.

```
#define TRUE 1
#define FALSE 0
#define MAX_PICKUP 4
#define MAX_AUTOMOBILE 8
#define NUM_CAR_OWNERS 20
#define RANDOM_ARRIVAL 1
```

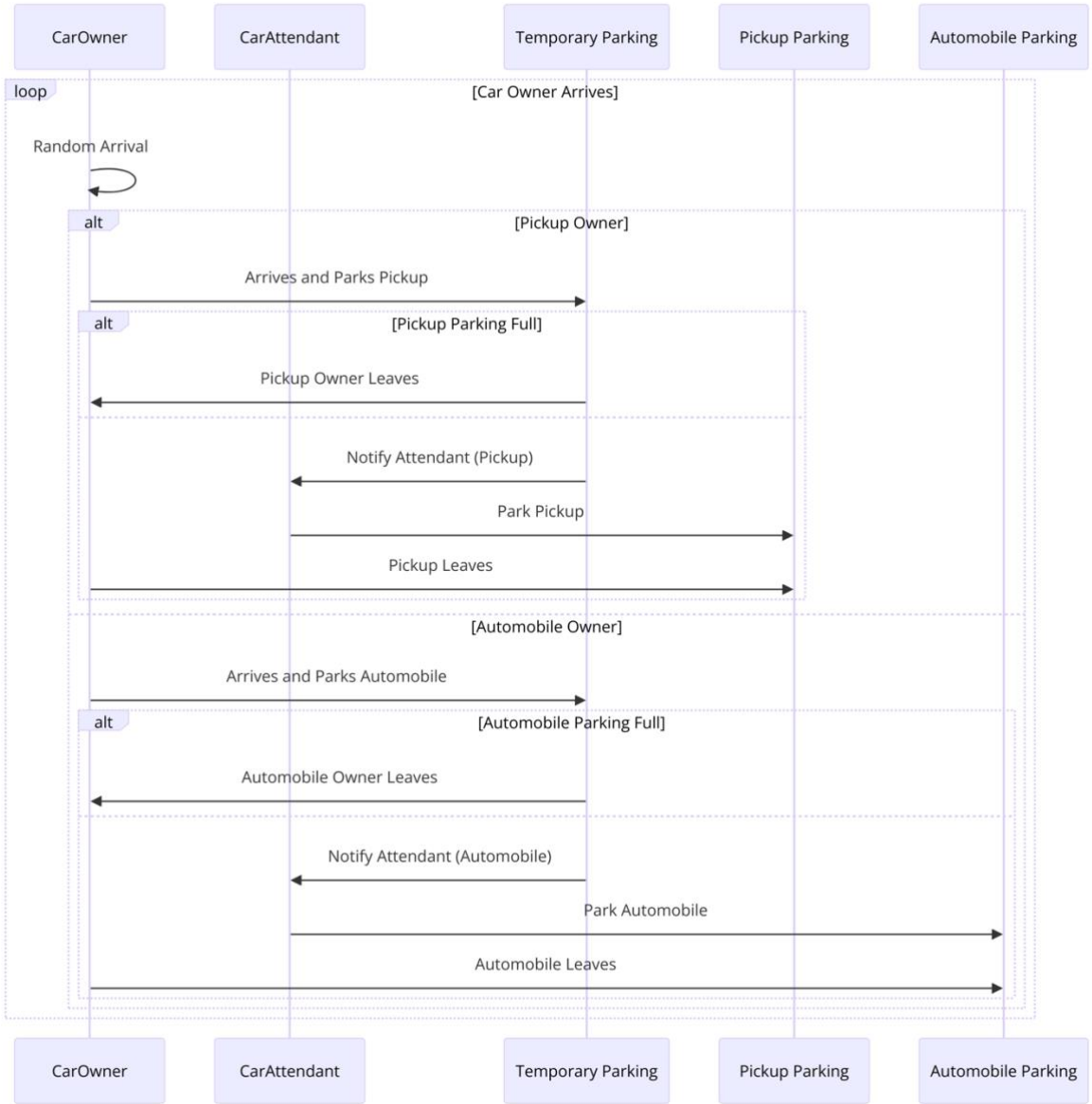
2. Introduction and Design

I designed the project to be a parking lot simulation.

Here is how the scenario works:

- Car owners arrive at the parking lot at random times.
- Only one vehicle can enter the temporary parking area at a time.

- The vehicle in the temporary area calls the relevant valet according to the type of vehicle and hands over the car. If the parking lot is full, the vehicle leaves the temporary area and exits the parking lot completely.
- The valet parks the car in the relevant parking area.
- The car owners retrieve their cars from the parking lot after a certain random period.
- The simulation ends when all car owners have finished their tasks.



3. Implementation

   a. Semaphores

      There is four semaphore

      newAutomobile: Controls the transition to the temporary automobile parking area.
      newPickup: Controls the transition to the temporary pickup parking area.
      inChargeForPickup: Wakes up the valet responsible for pickups.
      inChargeForAutomobile: Wakes up the valet responsible for automobiles.

      Semaphores

```
sem_t newAutomobile;
sem_t newPickup;

sem_t inChargeForPickup;
sem_t inChargeForAutomobile;
```

      Initialize semaphores

```
sem_init(&newAutomobile, 0, 1);
sem_init(&newPickup, 0, 1);

sem_init(&inChargeForPickup, 0, 0);
sem_init(&inChargeForAutomobile, 0, 0);
```

      Destroy semaphores

```
sem_destroy(&inChargeForPickup);
sem_destroy(&inChargeForAutomobile);
sem_destroy(&newAutomobile);
sem_destroy(&newPickup);
```

b. Threads

There is two main threads:

carOwner:

- The *carOwner* thread is a simulation of a car owner's actions in a parking lot. It takes a single argument, which is cast to an integer and stored in *carOwnerID*. The thread first determines the type of vehicle the owner has, either a pickup or an automobile, by generating a random number and taking the modulus by 2.

```
int type = rand() % 2;

int carOwnerID = (int)arg;
```

- If the *RANDOM_ARRIVAL* flag is set, the thread simulates a random arrival time by sleeping for a random duration between 1 and 5 seconds.

```
if(RANDOM_ARRIVAL){
    sleep_random(1, 5);
}
```

- If the vehicle type is PICKUP, the thread increments the ***pickupID*** and prints a message indicating the arrival of the pickup owner. It then waits for a semaphore *newPickup* to become available, simulating the parking of the pickup in a temporary parking spot.

```
int current_id =  ++pickupID;

printf("\n-> Pickup %d owner arrives.\n", current_id);

sem_wait(&newPickup);
```

- If there are no free parking spots for pickups (*free_parking_spots_pickup* is 0), the thread prints a message indicating that the pickup owner is leaving the parking lot, posts to the *newPickup* semaphore to indicate that the temporary parking spot is now available, and returns.

```
if(free_parking_spots_pickup == 0){

    printf("\n- The parking lot is full. "
    "Pickup %d owner leaves the temporary parking slot.\n",
    current_id);

    sem_post(&newPickup);
    return NULL;
}
```

- If there are free parking spots, the thread posts to the *inChargeForPickup* semaphore to wakes up to valet. The valet takes the car and parks it. Then sleeps for a random duration between 5 and 10 seconds to simulate the time the owner spends at the parking lot, and then prints a message indicating

that the pickup owner is leaving the parking lot and increments the number of free parking spots for pickups.

```
sem_post(&inChargeForPickup);

sleep_random(5, 10);

printf("\n$ Pickup %d owner leaves the park slot."
"Free parking spots on Pickup: %d\n",
current_id, ++free_parking_spots_pickup);

return NULL;
```

- The process is same for the automobile.

carAttendant:

- The carAttendant thread simulates the actions of a car attendant in a parking lot. It takes a single argument, which is cast to an integer and stored in type. This type represents the type of vehicle the attendant is responsible for, either a pickup or an automobile.

```
int type = (int) arg;
```

- The thread runs an infinite loop, simulating the continuous work of the attendant.

- If the vehicle type is PICKUP, the thread waits for the semaphore inChargeForPickup to become available. This semaphore represents the availability of a pickup for the attendant to park in the permanent parking.

```
sem_wait(&inChargeForPickup);
```

- Once the semaphore is available, The valet takes the car and parks it. The thread decreases the number of free parking spots for pickups (free_parking_spots_pickup) and prints a message indicating that the attendant has taken the pickup from the temporary parking and parked it in the permanent parking.

```
free_parking_spots_pickup--;

printf("\n# Vale takes the pickup %d from temporary parking.\n"
        "# Pickup %d is parked in permanent parking.\n"
        "# Available free permanent pickup parking spots now: %d\n",
        pickupID, pickupID, free_parking_spots_pickup);
```

- It then posts to the newPickup semaphore to indicate that the temporary parking spot is now available.

```
sem_post(&newPickup);
```

- The process is same for the automobile.

c. Main

- At the beginning, four semaphores are initialized using sem_init. The newAutomobile and newPickup semaphores are

initialized with a value of 1, indicating that there is initially one free spot for each type of vehicle. The inChargeForPickup and inChargeForAutomobile semaphores are initialized with a value of 0, indicating that no attendant is initially in charge of parking a vehicle of either type.

```
// Initialize semaphores
sem_init(&newAutomobile, 0, 1);
sem_init(&newPickup, 0, 1);

sem_init(&inChargeForPickup, 0, 0);
sem_init(&inChargeForAutomobile, 0, 0);
```

- Next, two arrays of pthread_t (a data type representing a thread) are declared. carOwnerThread will hold the threads for the car owners, and carAttendantThread will hold the threads for the car attendants.

```
pthread_t carOwnerThread[NUM_CAR_OWNERS];
pthread_t carAttendantThread[2];
```

- The program then creates a number of threads equal to NUM_CAR_OWNERS for the car owners, and two threads for the car attendants. The pthread_create function is used to create these threads. If thread creation fails, an error message is printed and the program exits.

```
// Create car owner threads
for (int i = 0; i < NUM_CAR_OWNERS; i++) {
    int check = pthread_create(&carOwnerThread[i], NULL, carOwner, (void *) i );
    if (check != 0) {
        printf("Error in creating thread\n");
        exit(1);
    }
}

// Create car attendant threads
for (int i = 0; i < 2; i++) {
    int check = pthread_create(&carAttendantThread[i], NULL, carAttendant, (void *) i);
    if (check != 0) {
        printf("Error in creating thread\n");
        exit(1);
    }
}
```

- After all threads have been created, the program waits for all car owner threads to finish using pthread_join. If joining a thread fails, an error message is printed and the program exits.

```
// Join car owner threads
for (int i = 0; i < NUM_CAR_OWNERS; i++) {
    int check = pthread_join(carOwnerThread[i], NULL);
    if (check != 0) {
        printf("Error in detaching thread in carOwner Error: %d\n", check);
        exit(1);
    }
}
```

- Once all car owner threads have finished, the program cancels the car attendant threads using pthread_cancel. This is necessary because the car attendant threads are designed to run indefinitely.

```
// Cancel car attendant threads
pthread_cancel(carAttendantThread[0]);
pthread_cancel(carAttendantThread[1]);
```

- After the car attendant threads have been cancelled, the program waits for them to finish using pthread_join. Again, if joining a thread fails, an error message is printed and the program exits.

```
// Join car attendant threads
for (int i = 0; i < 2; i++) {

    int check = pthread_join(carAttendantThread[i], NULL);
    if (check != 0) {
        printf("Error in detaching thread in carAttendant Error: %d\n", check);
        exit(1);
    }
}
```

- Finally, the program destroys all semaphores using sem_destroy, freeing the resources they were using.

```
sem_destroy(&inChargeForPickup);
sem_destroy(&inChargeForAutomobile);
sem_destroy(&newAutomobile);
sem_destroy(&newPickup);
```

4. Test

Test with a smaller area than the normal parking lot for demonstration purposes.

```
#define MAX_PICKUP 1
#define MAX_AUTOMOBILE 2
#define NUM_CAR_OWNERS 5
```

Result

```
root@957a2f030edd:/workspace/HW3/src# make run
./test

-> Pickup 1 owner arrives.

+ Pickup 1 is parked in temporary parking.

# Vale takes the pickup 1 from temporary parking.
# Pickup 1 is parked in permanent parking.
# Available free permanent pickup parking spots now: 0

-> Pickup 2 owner arrives.

+ Pickup 2 is parked in temporary parking.

- The parking lot is full. Pickup 2 owner leaves the temporary parking slot.

$ Pickup 1 owner leaves the park slot.Free parking spots on Pickup: 1

-> Automobile 1 owner arrives.

+ Automobile 1 is parked in temporary parking.

# Vale takes the automobile 1 from temporary parking.
# Automobile 1 is parked in permanent parking.
# Available free permanent automobile parking spots now: 1

-> Automobile 2 owner arrives.

+ Automobile 2 is parked in temporary parking.

# Vale takes the automobile 2 from temporary parking.
# Automobile 2 is parked in permanent parking.
# Available free permanent automobile parking spots now: 0

-> Pickup 3 owner arrives.

+ Pickup 3 is parked in temporary parking.

# Vale takes the pickup 3 from temporary parking.
# Pickup 3 is parked in permanent parking.
# Available free permanent pickup parking spots now: 0

$ Automobile 2 owner leaves the park slot. Free parking spots on Automobile: 1

$ Automobile 1 owner leaves the park slot. Free parking spots on Automobile: 2

$ Pickup 3 owner leaves the park slot.Free parking spots on Pickup: 1
root@957a2f030edd:/workspace/HW3/src#
```

Test with the normal parking lot area.

```
#define MAX_PICKUP 4
#define MAX_AUTOMOBILE 8
#define NUM_CAR_OWNERS 30
```

Result

The output is too long to include here.
The test results can be viewed in the ***test/normal_test_result.txt*** file.

Test with the enormous number of car owners

```
#define MAX_PICKUP 100
#define MAX_AUTOMOBILE 200
#define NUM_CAR_OWNERS 500
```

Result

The output is too long to include here.
The test results can be viewed in the ***test/enormous_test_result.txt*** file.

Test with memory check

```
Program is exiting...
==4627==
==4627== HEAP SUMMARY:
==4627==     in use at exit: 0 bytes in 0 blocks
==4627==   total heap usage: 40 allocs, 40 frees, 13,872 bytes allocated
==4627==
==4627== All heap blocks were freed -- no leaks are possible
==4627==
==4627== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@957a2f030edd:/workspace/HW3/src#
```