GEBZE TECHNICAL UNIVERSITY
CSE344 HW3 REPORT

Student Name: Emre Oytun
Student No: 200104004099

1) Design Decisions:

1.a) Critical Design Decisions about the System:

My system is working as described below:

- There is an entrance to the parking lot. Only one vehicle can be present at the entrance.

- When one vehicle comes, it checks if there is a free spot for this type of vehicle. If there is not any free spot, it leaves. Otherwise, the related valet parks the car while decrementing free parking lot number for this vehicle type.

Note: I thought there is only one parking area with 4 lots for cars and 4 lots for pickups because I concluded that from both the PDF and the responses from the assistant to the questions about this homework. The assistant said that he thought there is only one parking area, so there is nothing like temporary parking area and permanent parking area.

- Every 10 seconds, the valets remove one vehicle for each vehicle type if possible.

- 4 threads are created except main thread as specified in the homework PDF such that: “Two threads must be created to fulfil the task for every vehicle type: carOwner() and carAttendant().” Therefore, I created 1 carOwner thread and 2 carAttendant threads. I also created another thread “carRemover()” to remove vehicles from parking lots in every 10 seconds.

2) Semaphore Explanations:

```
sem_t newPickup;
sem_t inChargeforPickup;
sem_t newAutomobile;
sem_t inChargeforAutomobile;

// Counters for temporary parking spaces
volatile int mFree_automobile = 8;
volatile int mFree_pickup = 4;
volatile int program_stop = 0;

// Mutexes as binary semaphores to prevent race conditions
sem_t mutex_auto; // Mutex for mFree_automobile global variable
sem_t mutex_pickup; // Mutex for mFree_pickup global variable
sem_t valet_auto;   // Mutex for automobile valet
sem_t valet_pickup; // Mutex for pickup valet
sem_t printf_lock; // Mutex for printf usage
```

newPickup:
- This semaphore is used to indicate a new pickup has arrived to the entrance so that the pickup valet can park this car.

- It initialized with value 0.
- When a new pickup comes and there is a free pickup spot in the parking area, it increments this semaphore to signal the carAttendant() thread such that the pickup valet parks the car.

inChargeforPickup:
- This semaphore is used by carOwner() thread to wait the pickup valet while it's parking the pickup.
- It initialized with value 0.
- When a new pickup will be parked, the carOwner() thread should wait the valet before putting new car to the entrance, so it waits for this semaphore. Pickup valet increments this semaphore when it has parked the pickup.

newAutomobile:
- This semaphore is used to indicate a new automobile has arrived to the entrance so that the automobile valet can park this car.
- It initialized with value 0.
- When a new automobile comes and there is a free automobile spot in the parking area, it increments this semaphore to signal the carAttendant() thread such that the automobile valet parks the car.

inChargeforAutomobile:
- This semaphore is used by carOwner() thread to wait the automobile valet while it's parking the pickup.
- It initialized with value 0.
- When a new automobile will be parked, the carOwner() thread should wait the valet before putting new car to the entrance, so it waits for this semaphore. Automobile valet increments this semaphore when it has parked the automobile.

mutex_auto:
- This used when "mFree_automobile" variable is accessed or modified.
- It's used as binary semaphore to limit access to the critical region.

mutex_pickup:
- This used when "mFree_pickup" variable is accessed or modified.
- It's used as binary semaphore to limit access to the critical region.

valet_auto:
- This is used to prevent carAttendant() and carRemover() threads using the automobile valet at the same time.

valet_pickup:
- This is used to prevent carAttendant() and carRemover() threads using the pickup valet at the same time.

printf_lock:
- This is used to prevent threads printing at the same time because it can cause a race condition.
- Before every print, threads need to take this lock and release afterwards.

3) Threads Explanations:

3.b) carOwner():

```
const int total_coming_vehicle_num = 50;
void* carOwner(void* arg) {
    for (int i = 0; i < total_coming_vehicle_num; i++) {
        int vehicle_type = rand() % 2; // 0 for car, 1 for pickup

        if (vehicle_type == 0) { // Car
            sem_wait(&mutex_auto);
            if (mFree_automobile > 0) {
                sem_wait(&printf_lock);
                printf("Car owner arrives. Free car parking lots before parking: %d\n", mFree_automobile);
                sem_post(&printf_lock);

                sem_post(&newAutomobile);
                sem_post(&mutex_auto);

                // Wait for the valet to park the vehicle
                sem_wait(&inChargeforAutomobile);
            } else {
                sem_wait(&printf_lock);
                printf("No free car spots. Car owner leaves.\n");
                sem_post(&printf_lock);

                sem_post(&mutex_auto);
            }
        }
        else { // Pickup
            sem_wait(&mutex_pickup); // Critical region for mFree_pickup variable
            if (mFree_pickup > 0) {
                sem_wait(&printf_lock);
                printf("Pickup owner arrives. Free pickup parking lots before parking: %d\n", mFree_pickup);
                sem_post(&printf_lock);

                sem_post(&newPickup); // Increment newPickup semaphore to signal the valet
                sem_post(&mutex_pickup); // Exit critical region for mFree_pickup variable

                sem_wait(&inChargeforPickup); // Wait until valet has parked the vehicle
            } else {
                sem_wait(&printf_lock);
                printf("No free pickup parking lots. Pickup owner leaves.\n");
                sem_post(&printf_lock);

                sem_post(&mutex_pickup); // Exit critical region for mFree_pickup variable
            }
        }

        sleep(1); // 1 seconds interval for the next car arrival
    }

    program_stop = 1;
    return NULL;
}
```

- This thread provides random type of cars to the parking system.

- The cars are coming to the entrance one-by-one, 1 second after another.

- To make the program finite, I'm using "total_coming_vehicle_num" to determine total number of cars coming to the system. After these cars are arrived, the program finishes.

Note: You can change "total_coming_vehicle_num" as you wish to determine the total number of cars coming to the system through program's lifecycle.

- The coming car's type is determined by the randomly generated random. If it's 0, it's car. Otherwise, it's pickup.

- I'll explain the pickup arrival logic here which is in the else block with lots of comments. The automobile arrival logic is exactly the same with different semaphores.

- When a pickup arrives:

i) It gets the “mutex_pickup” to enter critical region since it’ll read “mFree_pickup” variable.
ii) It checks if there is an available spot for pickups. If not, it leaves the parking lot by releasing the “mutex_pickup”.
iii) If there is an available spot, it increment “newPickup” semaphore so that pickup valet will take the car and park it. It releases “mutex_pickup” since it will not access the “mFree_pickup” variable anymore.
iv) It waits for “inChargeforPickup” semaphore so it’s waiting the pickup valet to park the car here before going to the next car arrival.

3.b) carAttendant():

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

    while (!program_stop) {
        // Check for new automobiles
        if (vehicle_type == 0) {
            if (sem_wait(&newAutomobile) == 0) {
                if (!program_stop) {
                    sem_wait(&valet_auto); // Lock automobile valet
                    sem_wait(&mutex_auto);
                    mFree_automobile--;

                    sem_wait(&printf_lock);
                    printf("Car valet parks a car. Free car parking lots after parking: %d\n", mFree_automobile);
                    sem_post(&printf_lock);

                    sem_post(&mutex_auto);
                    sem_post(&valet_auto); // Unlock automobile valet
                    sem_post(&inChargeforAutomobile);
                }
            }
        }

        // Check for new pickups
        else {
            if (sem_wait(&newPickup) == 0) {
                if (!program_stop) {
                    sem_wait(&valet_pickup); // Lock valet_pickup so that we ensure there is only one valet for pickups (necessary because carRemover for pickup should not work at the same time)
                    sem_wait(&mutex_pickup); // Critical region for mFree_pickup variable
                    mFree_pickup--;

                    sem_wait(&printf_lock);
                    printf("Pickup valet parks a pickup. Free pickup parking lots after parking: %d\n", mFree_pickup);
                    sem_post(&printf_lock);

                    sem_post(&mutex_pickup); // Exit critical region for mFree_pickup variable
                    sem_post(&valet_pickup); // Unlock valet_pickup so that pickup valet is free now
                    sem_post(&inChargeforPickup); // Signal to the carOwner thread indicating the vehicle has been parked
                }
            }
        }
    }
    return NULL;
}
```

- This thread is responsible for one of the valets. It stops when the program_stop is 1.
- I’ll explain it only for pickup valet because they work exactly the same with another valet; only the semaphores they use are different.

- It checks if new pickup has arrived by using “sem_wait” on semaphore “newPickup”.

- When it wakes up in a new pickup arrival:

i) It decrements the “valet_pickup” to lock the pickup valet so that carRemover() thread’s pickup valet can not be used at the same time.
ii) It decrements the “mutex_pickup” semaphore to lock the “mFree_pickup” variable since it will modify it.
iii) It decrements the “mFree_pickup” semaphore to indicate one more spot is used right now.
iv) It increments the “valet_pickup” and “mutex_pickup” semaphores to release them at the end.
v) It also increments “inChargeforPickup” semaphore to indicate the valet has come and parked the car so that carAttendant() thread can continue.

3.c) carRemover():

```
// Function to periodically remove cars and pickups from the parking lot
void* carRemover(void* arg) {
    while (!program_stop) {
        sleep(10); // Sleep for 10 seconds

        // Remove a car if available
        sem_wait(&valet_auto); // Lock automobile valet
        sem_wait(&mutex_auto);
        if (mFree_automobile < 8) {
            mFree_automobile++;

            sem_wait(&printf_lock);
            printf("Car valet removes a car. Free car parking lots: %d\n", mFree_automobile);
            sem_post(&printf_lock);
        }
        sem_post(&mutex_auto);
        sem_post(&valet_auto); // Unlock automobile valet

        // Remove a pickup if available
        sem_wait(&valet_pickup); // Lock pickup valet
        sem_wait(&mutex_pickup);
        if (mFree_pickup < 4) {
            mFree_pickup++;

            sem_wait(&printf_lock);
            printf("Pickup valet removes a pickup. Free pickup parking lots: %d\n", mFree_pickup);
            sem_post(&printf_lock);
        }
        sem_post(&mutex_pickup);
        sem_post(&valet_pickup); // Unlock pickup valet
    }
    return NULL;
}
```

- This thread is responsible for two valets. It stops when the program_stop is 1.

- In every 10 seconds, pickup and automobile valets check if they can remove one car.

- If we examine one of them, let us look at the pickup removal logic again:

i) It decrements the “valet_pickup” to lock the pickup valet so that carAttendant() thread’s pickup valet can not be used at the same time.
ii) It decrements the “mutex_pickup” semaphore to lock the “mFree_pickup” variable since it will read and modify it.
iii) It checks if there is at least one pickup to remove. If there is, it removes the car so it increments “mFree_pickup” counter variable.
iv) It releases the locked semaphores at the end.

3.d) Main thread:

```c
int main() {
    srand(time(NULL));

    // Initialize semaphores
    sem_init(&newPickup, 0, 0);
    sem_init(&inChargeforPickup, 0, 0);
    sem_init(&newAutomobile, 0, 0);
    sem_init(&inChargeforAutomobile, 0, 0);
    sem_init(&mutex_auto, 0, 1);
    sem_init(&mutex_pickup, 0, 1);
    sem_init(&valet_auto, 0, 1);
    sem_init(&valet_pickup, 0, 1);
    sem_init(&printf_lock, 0, 1);

    pthread_t carOwnerThread, carAttendantPickupThread, carAttendantAutomobileThread, carRemoverThread;

    // Create threads
    int automobile_arg = 0;
    int pickup_arg = 1;

    pthread_create(&carOwnerThread, NULL, carOwner, NULL);
    pthread_create(&carAttendantAutomobileThread, NULL, carAttendant, (void*) &automobile_arg);
    pthread_create(&carAttendantPickupThread, NULL, carAttendant, (void*) &pickup_arg);
    pthread_create(&carRemoverThread, NULL, carRemover, NULL);

    // Wait for the car owner thread to finish
    pthread_join(carOwnerThread, NULL);

    // Here, I'm posting these two semaphores so that attendant threads can wake up and see the program has stopped
    sem_post(&newPickup);
    sem_post(&newAutomobile);

    pthread_join(carAttendantAutomobileThread, NULL);
    pthread_join(carAttendantPickupThread, NULL);
    pthread_join(carRemoverThread, NULL);

    // Destroy semaphores
    sem_destroy(&newPickup);
    sem_destroy(&inChargeforPickup);
    sem_destroy(&newAutomobile);
    sem_destroy(&inChargeforAutomobile);
    sem_destroy(&mutex_auto);
    sem_destroy(&mutex_pickup);
    sem_destroy(&valet_auto);
    sem_destroy(&valet_pickup);
    sem_destroy(&printf_lock);

    return 0;
}
```

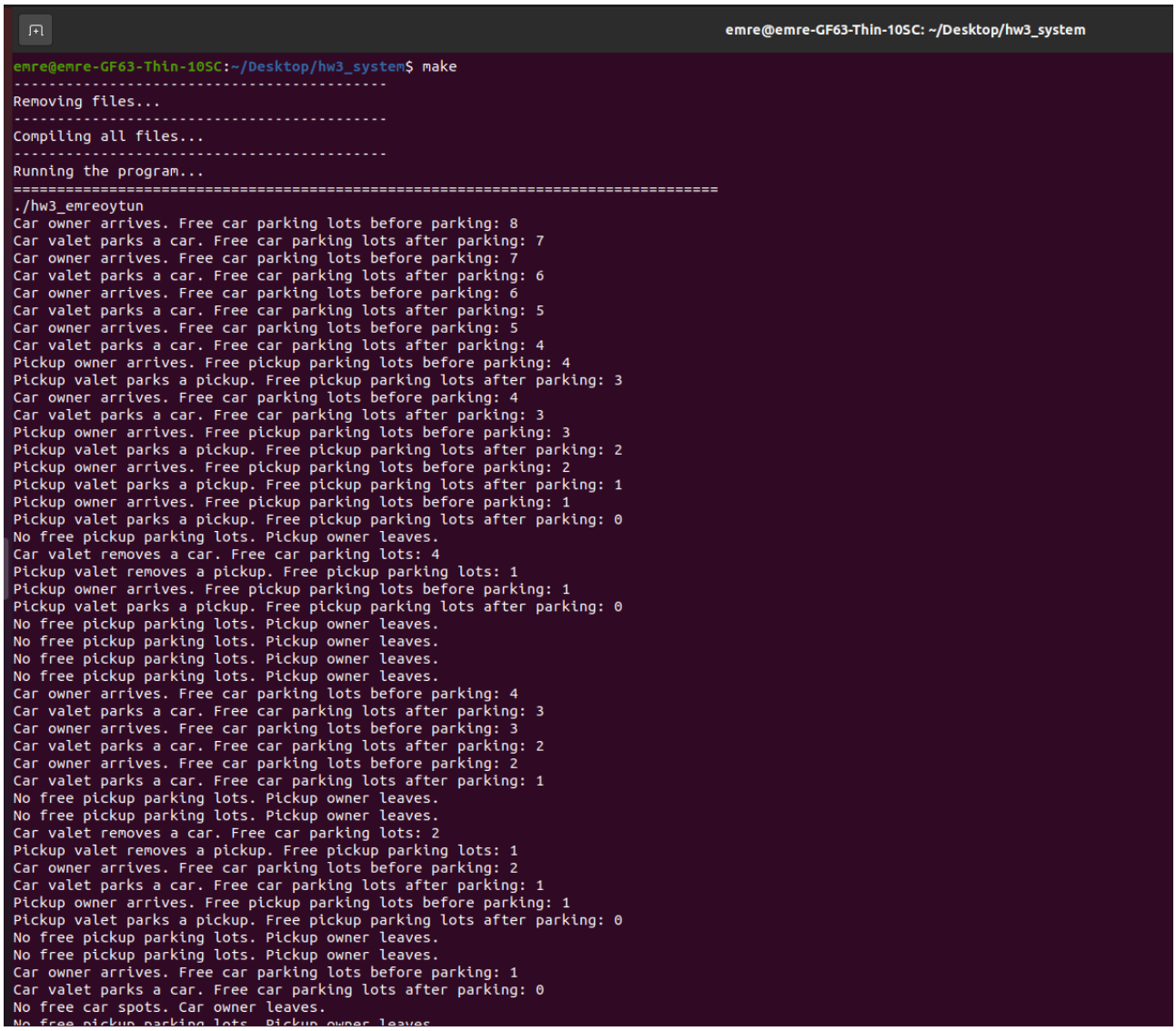
- The main thread initializes the semaphores and creates threads.

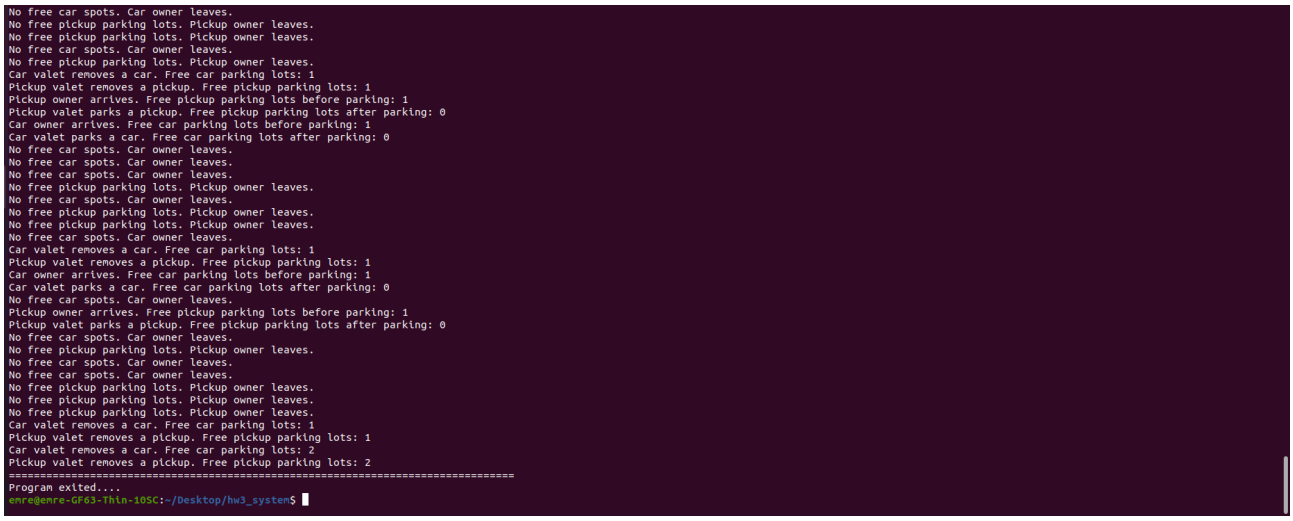
- It gives 0 to carAttendant thread when it is automobile, and 1 when it is pickup.

- Afterwards, it waits for the threads by joining them and destroys the semaphores to prevent the memory leaks.

4) Screenshots:

4.a) The Whole Program Output:





4.b) Program Output Explanations:

```
Car owner arrives. Free car parking lots before parking: 8
Car valet parks a car. Free car parking lots after parking: 7
Car owner arrives. Free car parking lots before parking: 7
Car valet parks a car. Free car parking lots after parking: 6
Car owner arrives. Free car parking lots before parking: 6
Car valet parks a car. Free car parking lots after parking: 5
Car owner arrives. Free car parking lots before parking: 5
Car valet parks a car. Free car parking lots after parking: 4
Pickup owner arrives. Free pickup parking lots before parking: 4
Pickup valet parks a pickup. Free pickup parking lots after parking: 3
Car owner arrives. Free car parking lots before parking: 4
Car valet parks a car. Free car parking lots after parking: 3
Pickup owner arrives. Free pickup parking lots before parking: 3
Pickup valet parks a pickup. Free pickup parking lots after parking: 2
Pickup owner arrives. Free pickup parking lots before parking: 2
Pickup valet parks a pickup. Free pickup parking lots after parking: 1
Pickup owner arrives. Free pickup parking lots before parking: 1
Pickup valet parks a pickup. Free pickup parking lots after parking: 0
No free pickup parking lots. Pickup owner leaves.
```

- As you see from the first part of the program, initially there are 8 parking lots for cars(Car = Automobile) and 4 parking lots for pickups.

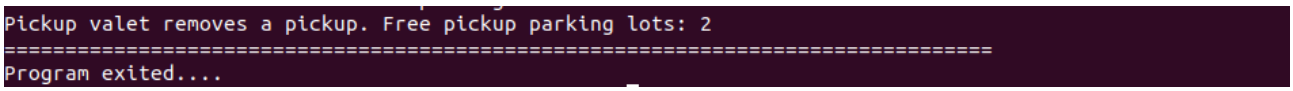
- When the valet parks a vehicle, the number of free parking lots for this vehicle type decrements as expected.

- The vehicles come to the park entrance one-by-one as expected.

```
Pickup valet parks a pickup. Free pickup parking lots after parking: 0
No free pickup parking lots. Pickup owner leaves.
Car valet removes a car. Free car parking lots: 4
Pickup valet removes a pickup. Free pickup parking lots: 1
Pickup owner arrives. Free pickup parking lots before parking: 1
Pickup valet parks a pickup. Free pickup parking lots after parking: 0
No free pickup parking lots. Pickup owner leaves.
No free pickup parking lots. Pickup owner leaves.
No free pickup parking lots. Pickup owner leaves.
No free pickup parking lots. Pickup owner leaves.
```

- As you see from the 2nd line and last 4 lines, pickups arrives to the entrance but there is no free parking lot so it immediately leaves.

- Also, there was no free lots for pickups at the beginning of this part but pickup valet removes one pickup and another pickup was parked afterwards to this newly freed parking lot. So, car removal is working as expected.



- Program is closed after the specified number of cars have arrived to the system.

Note: At the end, the program can wait 10 seconds at max because the carRemover() thread is sleeping 10 seconds, then it wakes up and check if the program has stopped.