King Abdulaziz University
Faculty of Engineering
Electrical and Computer Engineering Department

Operating Systems (EE463)

Term Project

Design Document Final Report

Car Park Simulator

Team BG02

Member	Name	ID
1	**Hayan Al-machnouk**	**1945954**
2	**Tamam Alahdal**	**1851778**
3	**Zyad Zamil**	**1854399**

Instructor:

Dr. Abdulghani M. Al-Qasimi

Table of Contents

Main Algorithm Design

Introduction

In this project, a system of queueing and serving is implemented to manage the flow of cars entering and leaving the car-park. A random number generator is used to simulate the arrival of cars, which are then added to a first-in, first-out (FIFO) queue for processing. This ensures that cars are served in the order in which they arrived. In-valet worker threads are responsible for removing cars from the queue and parking them in available slots. These slots are represented using a highest priority in, first out (HPIFO) queue, which assigns priority to cars based on the amount of time they have remaining to leave. The in-valet threads check the availability of slots in this priority queue and, if there is a vacancy, they will park a new car in it. Out-valet threads are responsible for removing cars that have completed their stay in the car-park. They check for cars whose leaving time has come, and then remove them from the parking slots and update related states and data. To ensure that no two threads manipulate the same queue at the same time, each queue has its own mutex lock. Additionally, since the priority queue is shared between both in-valet and out-valet threads, two counting semaphores are used to ensure that the car-park does not become overfull or empty. The critical change we have made is that me created a separate thread that generates the cars.

Features

- Parking space utilization: A statistic indicating the current car park space utilization.
- Total number of cars: A statistic indicating the total number of cars created during the simulation.
- Total number of cars allowed to park: A statistic indicating the total number of cars that have been allowed to park.
- Total number of cars not allowed to park: A statistic indicating the total number of cars that have been not allowed to park.
- Car-waiting time in the arrival queue: A statistic indicating the total sum of car-waiting times in the queue.
- Car-parking duration: A statistic indicating the total sum of car-parking durations.
- Current number of occupied slots in the parking space: A statistic indicating the current number of occupied slots in the parking space.

Design Decisions & Implementations Details

- Constants: A set of constants will be used to set the default values for the parking space capacity, number of in-valets, number of out-valets, size of the waiting cars queue, and expected number of incoming cars.
- Variables: A set of variables will be used to store the configurable values for the parking space capacity, number of in-valets, number of out-valets, size of the waiting cars queue, and expected number of incoming cars.
- Threads: The program will use a set of threads to simulate the in-valets, out-valets, monitor, and car generation processes.
- Mutex: A mutex will be used to protect the shared resources in the program (e.g. the queue and parking space).

- Semaphores: Semaphores will be used to avoid the busy-waiting for the parking space and queue.
- Car park array: An array of cars will be used to represent the car park space.
- Statistical variables: During the simulation, a set of statistical variables will be used to gather and display some data about the performance of the car-park operations.
- Time variables: A set of time variables will be used to print the times in the simulator's final report.
- Helper functions: The program will include several helper functions to print the statistics, cancel threads, and handle signals.

Main Thread

The main thread is the thread that handles the command line arguments and sets up the necessary data structures and threads. It is responsible for setting the default values as well as validating and setting the command line arguments. It also creates the necessary threads for the simulation, such as the monitor thread, the car generator thread and the in/out valet threads. Finally, it joins all the threads to wait for them to complete and then exits the program.

Arguments

The program takes in the following arguments:

- psize: The size of the car park. The default value is 10 and the range is between 5 and 15.
- inval: The number of in-valets. The default value is 3 and the range is between 1 and 5.
- outval: The number of out-valets. The default value is 2 and the range is between 1 and 5.
- qsize: The size of the queue. The default value is 10 and the range is between 5 and 15.
- expnum: The expected number of cars. The default value is 10.0 and the range is between 5.0 and 15.0.

Functions

The program consists of the following functions:

- main: Handles the command line arguments and sets up the necessary data structures and threads.
- monitor_func: Monitors the progress of the simulation.
- in_valet_func: Handles the process of cars entering the car park.
- out_valet_func: Handles the process of cars exiting the car park.
- car_gen_func: Generates cars and places them in the queue for either entering or exiting the car park.
- sigterm_handler, sigterm_quit, sigterm_kill: Handles the SIGTERM, SIGQUIT and SIGKILL signals to abort the program.
- Qinit, pQinit, Ainit, Aiterator: Set up the necessary data structures such as the queue, parking queue and car park array.

In-valet Thread

Entry Section

The entry section of the in-valet function is responsible for ensuring that all necessary conditions are met before entering into its critical section. The entry section begins by waiting on the Q_have_Cars semaphore. This semaphore is used to ensure that there is a car in the arrival queue that needs to be serviced. If there are no cars in the queue, the valet will remain in this section until a car arrives.

Once the semaphore is satisfied, the valet will then set its state to WAIT, signifying that it is waiting to enter the parking space. The valet will then wait on the P_emptySlots semaphore, which ensures that there is an available parking slot for the car.

Finally, the valet will acquire locks on both the arrival queue and the parking space, ensuring that no other valets are able to access the same resources.

Critical Section

Once all the necessary resources have been acquired, the valet can enter its critical section. This section is responsible for servicing the car in the arrival queue. The valet will pop the car from the queue and set its state to FETCH. The valet will then simulate the time it takes to fetch the car, before setting its state to MOVE.

The valet will then assign itself to the car, increment the number of cars currently acquired by in-valets, and add the car to the parking queue. The valet will also record the time that the car arrived in the parking space and assign it a random amount of time for its stay.

Exit Section

Once the car has been successfully parked, the valet can leave its critical section and enter its exit section. This section is responsible for releasing any resources that were acquired during the entry section. The valet will first post to the P_haveCars semaphore, signifying that there is now a car that can be serviced by an out-valet.

The valet will then unlock the parking space, allowing other valets to enter its critical section. The valet will also unlock the arrival queue, allowing new cars to enter the system. Finally, the valet will set its state to READY and update the system statistics before returning back to the pool.

Out-valet Thread

Entry Section

The entry section is responsible for the out-valet thread's initial operations. First, the out-valet thread waits for the semaphore P_haveCars to be signaled, indicating that there are cars in the parking. Once this semaphore is signaled, the out-valet thread acquires the park_lock mutex in order to enter the critical section.

Critical Section

The critical section is responsible for the out-valet thread's main operations. First, the out-valet thread retrieves the car at the top of the priority queue and checks if its leaving time is valid. If it is valid, the out-valet thread sets its state to MOVE, retrieves the car from the priority queue, serves the car from the arrival queue, decreases the number of occupied slots, signals the semaphore P_emptySlots to indicate that a slot is now empty, updates the statistics, and sleeps for one second to simulate the driving of a car until it leaves.

Exit Section

The exit section is responsible for the out-valet thread's final operations. First, the out-valet thread releases the park_lock mutex, updates the statistics, and shows the statistics. Then, the out-valet thread sets its state to READY and sleeps for a random amount of time before repeating the operations in the entry section.

Car Generation Thread

Endless Loop

This part of the code enters an endless loop where it generates incoming cars. It begins by assigning a probability value to a variable, which is passed in as an argument. Then, it enters a while loop that runs indefinitely, generating a pseudo random number of cars each cycle.

Enter CS for Queue

For each car that is generated, the code enters a critical section for the queue. It locks the queue, then allocates memory for the car and checks if the queue is full. If the queue is not full, the car is enqueued and a semaphore is posted to indicate that the queue has cars.

Wait Before Adding a Car

After enqueuing the car, the code waits before adding the next car. This is done by generating a random value between 0 and 0.2, then sleeping for that length of time. This is done to ensure that cars are not added too quickly.

Exit CS for Queue

Once the car has been enqueued, the code exits the critical section for the queue by unlocking it. It then updates the statistics and shows them on the screen, then sleeps for a random amount of time before generating the next car.

The Monitor Thread

Enter CS for park

This section of code is responsible for locking the arrival queue, which is a synchronization tool used to ensure that only one thread can access the resources in the parking lot at a time.

This done by utilizing the pthread_mutex_lock() function to lock the arrival queue. This ensures that only one thread can access the resources in the parking lot at a time, thus preventing any potential race conditions.

Updating State

This section of code is responsible for printing the current state of the parking lot and updating it accordingly. This is done because the code creates an array of cars in the parking lot, and then loops through them to print out their id numbers, as well as the time they have been in the parking lot. Additionally, the code also keeps track of the amount of time that has passed since the last iteration, which is used to update the state of the parking lot.

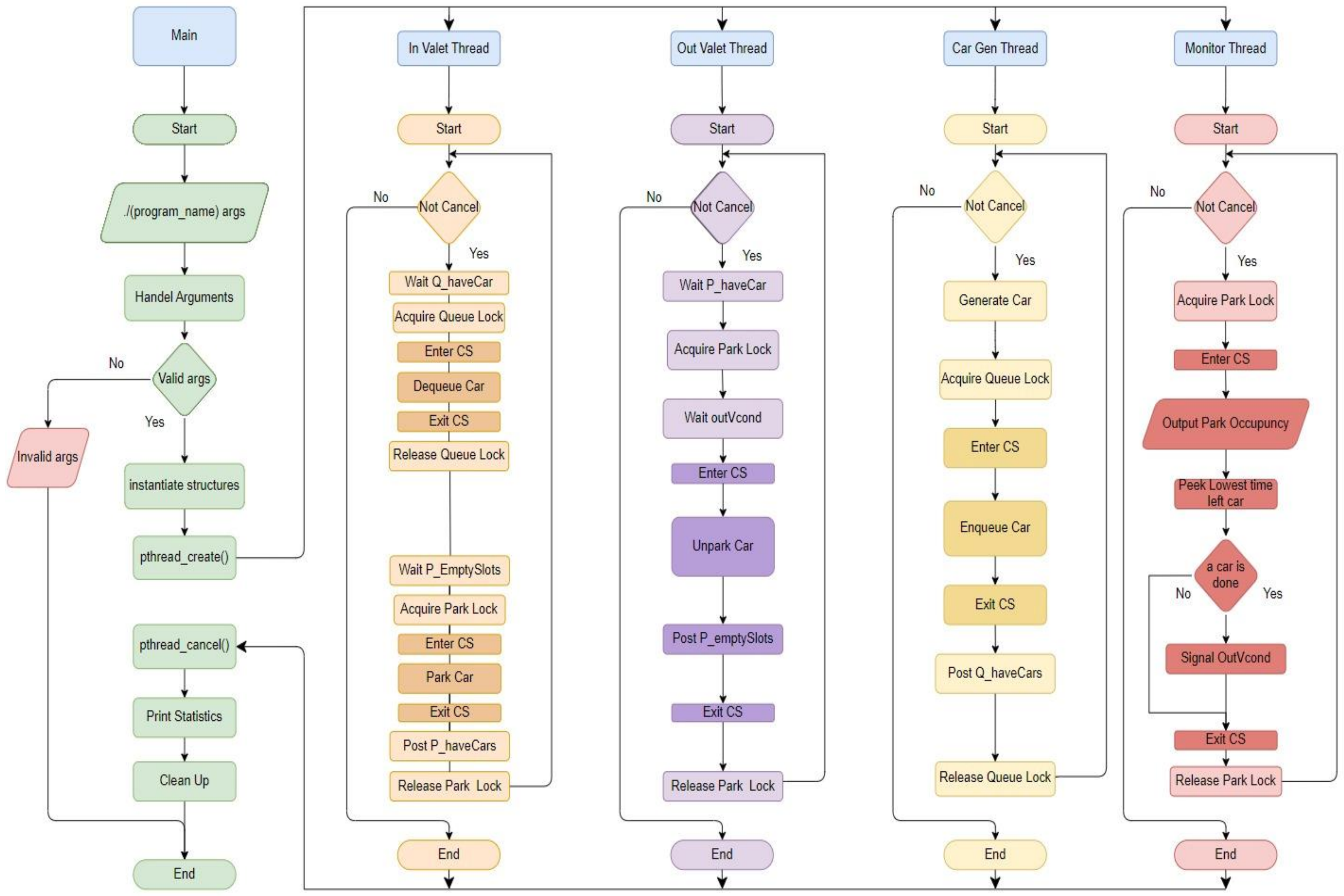
Exit CS for park

This section of code is responsible for unlocking the arrival queue. This is done by utilizing the pthread_mutex_unlock() function to unlock the arrival queue. This allows other threads to access the resources in the parking lot, and ensures that race conditions do not occur.

Calculating the Utilization

This section of code is responsible for calculating the utilization of the parking lot as a percentage. The code takes the number of cars in the parking lot and divides it by the capacity of the parking lot, and then multiplies it by 100 to get the percentage. The utilization is then stored in a variable and added to a total utilization variable, which is used to calculate the average utilization of the parking lot. Additionally, the code also updates the statistics of the parking lot, such as the number of cars and the number of refuels.

Flowchart


Main
Start
./(program_name) args
Handel Arguments
Valid args
No
Yes
Invalid args
instantiate structures
pthread_create()
pthread_cancel()
Print Statistics
Clean Up
End
In Valet Thread
Start
Not Cancel
No
Yes
Wait Q_haveCar
Acquire Queue Lock
Enter CS
Dequeue Car
Exit CS
Release Queue Lock
Wait P_EmptySlots
Acquire Park Lock
Enter CS
Park Car
Exit CS
Post P_haveCars
Release Park Lock
End
Out Valet Thread
Start
Not Cancel
No
Yes
Wait P_haveCar
Acquire Park Lock
Wait outVcond
Enter CS
Unpark Car
Post P_emptySlots
Exit CS
Release Park Lock
End
Car Gen Thread
Start
Not Cancel
No
Yes
Generate Car
Acquire Queue Lock
Enter CS
Enqueue Car
Exit CS
Post Q_haveCars
Release Queue Lock
End
Monitor Thread
Start
Not Cancel
No
Yes
Acquire Park Lock
Enter CS
Output Park Occupuncy
Peek Lowest time left car
a car is done
No
Yes
Signal OutVcond
Exit CS
Release Park Lock
End

Conclusion

In conclusion, this project successfully implemented a system of queueing and serving to manage the flow of cars entering and leaving the car-park. Through the use of various threads, such as the in-valet, out-valet, monitor and car generation threads, along with synchronization techniques such as mutexes and semaphores, the system was able to ensure that the cars were served in the order in which they arrived and that the car park space was efficiently utilized. Additionally, various statistical variables were utilized to gather and display data about the performance of the car-park operations, such as the total number of cars, the total number of cars allowed to park, the total number of cars not allowed to park, the car-waiting time in the arrival queue, the car-parking duration, and the current number of occupied slots in the parking space. Through the use of these variables, the system was able to provide an accurate and insightful analysis of the car park system. Overall, the project was able to achieve its goal of simulating a car park system with high accuracy and efficiency.