The Roller Coaster Problem

Bianca Garcia Martins 606723
Gabriel Alves 726515
Giulia Mazzutti Teixeira 725817

Problema

Suponha que exista n threads de passageiros e n threads de carros. Os passageiros repetidamente esperam para andar no carrinho, que suporta C passageiros, onde C < n. O carro pode fazer seu percurso apenas quando está lotado. Estes são alguns detalhes adicionais:

- A thread passageiros deve invocar as funções de: embarcar e desembarcar.

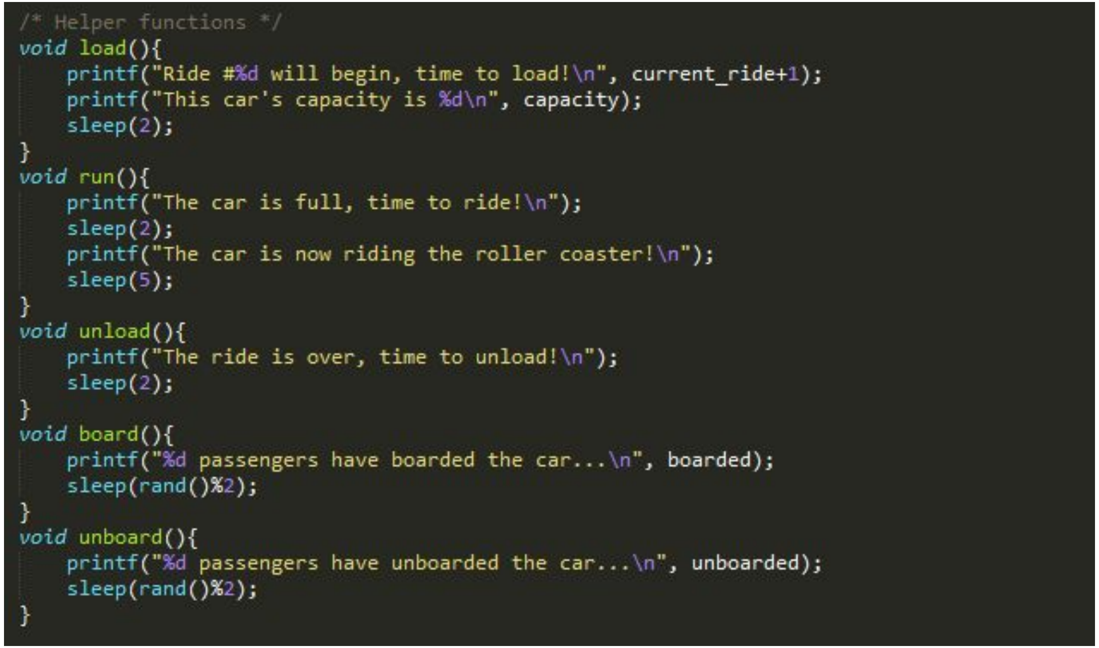
- A thread carro deve invocar as função de: ser preenchido, esvaziado e andar.

- A thread de passageiros não podem embarcar até que o carro esteja completamente preenchido.

- A thread carro não pode dar a partida até que os C passageiros estejam a bordo.

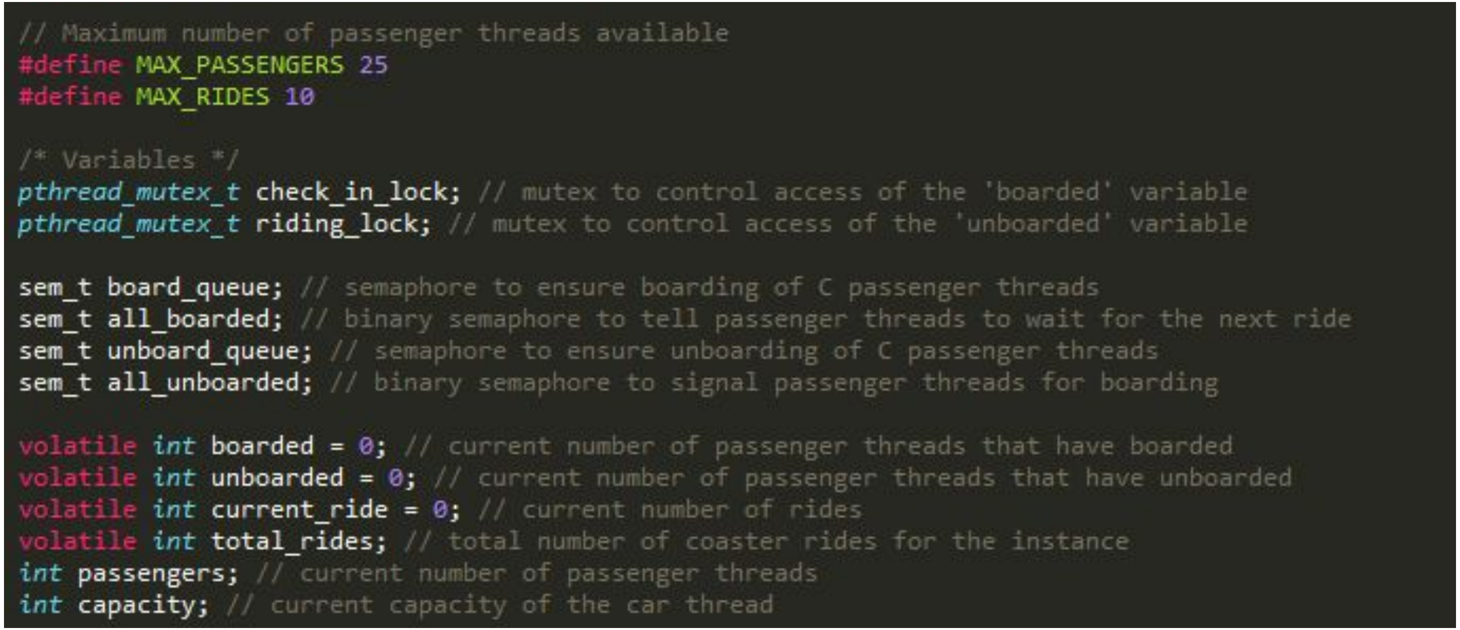
- A thread passageiros não podem desembarcar até que a thread carro invoque a função esvaziar.

Funções utilitárias:



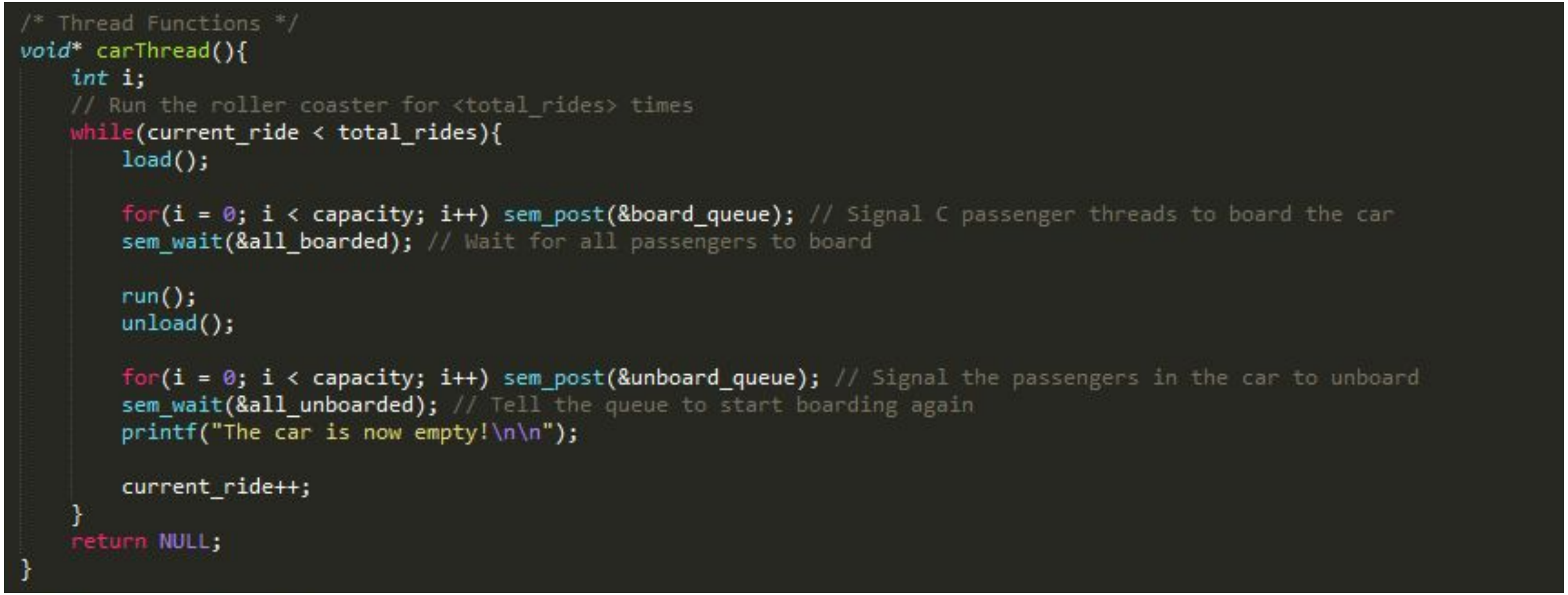
```c
/* Helper functions */
void load(){
    printf("Ride #%d will begin, time to load!\n", current_ride+1);
    printf("This car's capacity is %d\n", capacity);
    sleep(2);
}
void run(){
    printf("The car is full, time to ride!\n");
    sleep(2);
    printf("The car is now riding the roller coaster!\n");
    sleep(5);
}
void unload(){
    printf("The ride is over, time to unload!\n");
    sleep(2);
}
void board(){
    printf("%d passengers have boarded the car...\n", boarded);
    sleep(rand()%2);
}
void unboard(){
    printf("%d passengers have unboarded the car...\n", unboarded);
    sleep(rand()%2);
}
```

Declaração de variáveis:



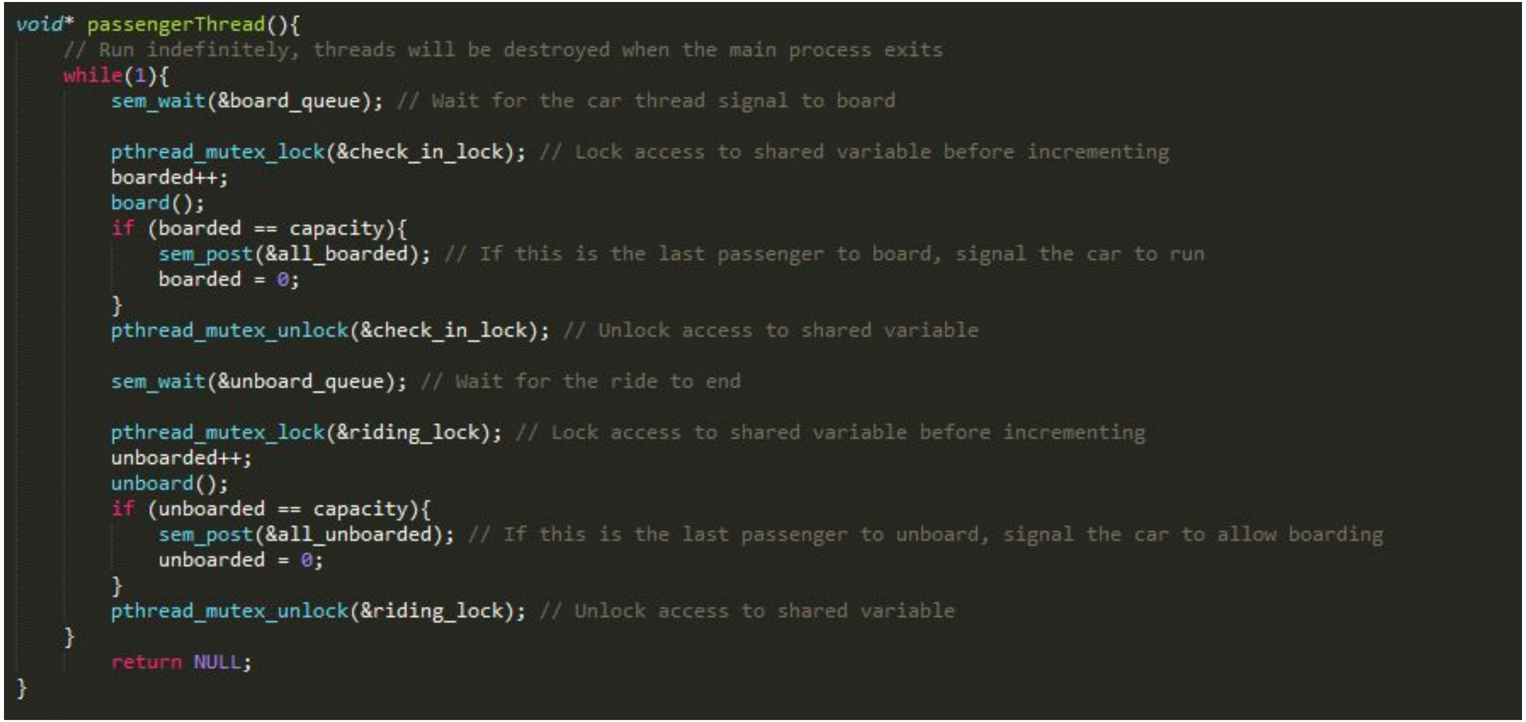
```c
// Maximum number of passenger threads available
#define MAX_PASSENGERS 25
#define MAX_RIDES 10

/* Variables */
pthread_mutex_t check_in_lock; // mutex to control access of the 'boarded' variable
pthread_mutex_t riding_lock; // mutex to control access of the 'unboarded' variable

sem_t board_queue; // semaphore to ensure boarding of C passenger threads
sem_t all_boarded; // binary semaphore to tell passenger threads to wait for the next ride
sem_t unboard_queue; // semaphore to ensure unboarding of C passenger threads
sem_t all_unboarded; // binary semaphore to signal passenger threads for boarding

volatile int boarded = 0; // current number of passenger threads that have boarded
volatile int unboarded = 0; // current number of passenger threads that have unboarded
volatile int current_ride = 0; // current number of rides
volatile int total_rides; // total number of coaster rides for the instance
int passengers; // current number of passenger threads
int capacity; // current capacity of the car thread
```

Thread Car:



```
/* Thread Functions */
void* carThread(){
    int i;
    // Run the roller coaster for <total_rides> times
    while(current_ride < total_rides){
        load();

        for(i = 0; i < capacity; i++) sem_post(&board_queue); // Signal C passenger threads to board the car
        sem_wait(&all_boarded); // Wait for all passengers to board

        run();
        unload();

        for(i = 0; i < capacity; i++) sem_post(&unboard_queue); // Signal the passengers in the car to unboard
        sem_wait(&all_unboarded); // Tell the queue to start boarding again
        printf("The car is now empty!\n\n");

        current_ride++;
    }
    return NULL;
}
```

Thread Passageiros:



```
void* passengerThread(){
    // Run indefinitely, threads will be destroyed when the main process exits
    while(1){
        sem_wait(&board_queue); // Wait for the car thread signal to board

        pthread_mutex_lock(&check_in_lock); // Lock access to shared variable before incrementing
        boarded++;
        board();
        if (boarded == capacity){
            sem_post(&all_boarded); // If this is the last passenger to board, signal the car to run
            boarded = 0;
        }
        pthread_mutex_unlock(&check_in_lock); // Unlock access to shared variable

        sem_wait(&unboard_queue); // Wait for the ride to end

        pthread_mutex_lock(&riding_lock); // Lock access to shared variable before incrementing
        unboarded++;
        unboard();
        if (unboarded == capacity){
            sem_post(&all_unboarded); // If this is the last passenger to unboard, signal the car to allow boarding
            unboarded = 0;
        }
        pthread_mutex_unlock(&riding_lock); // Unlock access to shared variable
    }
        return NULL;
}
```