COMP 304

Operating Systems

Term Project Report

Instructor: Şeniz Demir

Arda Serdar Pektezol - 042001037
Ahmet Batuhan Ulugergerli - 041901025

1. Introduction

Summer time is approaching and there are many vehicles and ferries that travel between Kocaeli and Yalova ports. Focus of this project was to implement this vehicle-ferry transportation solution between two ports using threads and process synchronization tools in the C programming language. We have implemented both vehicles and ferries as individual POSIX threads which both of them are also stored in a global resource array. Once all of the threads are created, the main thread waits for all of them to complete which then and only then it can successfully exit the program. Using mutex locks, resources between the vehicle and ferry threads are accessed and modified without interfering with each other or any inconsistent data. This project almost completely applies the requirements and constraints that it had to do.

2. Implementation

```c
#define _POSIX_C_SOURCE 199309L

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>
#include "structs.h"

#define NUM_VEHICLES 32

// Shared resources
Port ports[2];
Vehicle* vehicles[NUM_VEHICLES];
Ferry* ferries[2];
pthread_mutex_t print_lock;
// Threads
int vehicle_threads_completed[NUM_VEHICLES];
int vehicle_start_ports[NUM_VEHICLES];
int vehicle_end_ports[NUM_VEHICLES];
pthread_t vehicle_threads[NUM_VEHICLES];
pthread_t ferry_threads[2];
// Function declarations
int get_total_vehicles_in_port(int port_id);
int vehicle_left_in_booths(int port_id);
int ferry_in_port(int port_id);
int available_vehicle_left(Ferry* f);
int board_ferry(Ferry* f, Vehicle* v);
void* ferry_thread(void* arg);
void* vehicle_thread(void* arg);
void msleep(int ms);
```

Figure 1: Defines, includes, global resources, threads and function declarations.

In Figure 1, we can see the macros that are defined, libraries that are included, global resources which are used by the threads, thread and thread status arrays, and function declarations.

First macro is needed in order to have the ability to make a sleep with nanosecond precision along with time.h library, instead of the second precision given to us by sleep() from the unistd.h library. Second macro defines the number of vehicles to be created during initialization. All included libraries are necessary for the program to work. Last library included, “structs.h” is our own header file containing our struct typedefs and function declarations.

After these, in shared resources, ports[] array of 2 can be seen that holds the data structure of each port, including its booths, waiting lines and mutex locks. There are also vehicles[] and ferries[] arrays of their respective struct that hold each initialized vehicle and ferry. Pointer values are stored which gives us the ability to modify the values when they are retrieved from their arrays.

Functions declarations are written for ease of use but also in order to see every function.

```c
#ifndef STRUCTS_H
#define STRUCTS_H

#include <pthread.h>

// Vehicle implementation in a struct.
typedef struct {
    int id;
    int type;       // 1 = motorcycle, 2 = car, 3 = bus, 4 = truck
    int special;    // 0 = normal, 1 = special
    int port_id;
    int booth_id;
} Vehicle;

typedef struct Node Node;

// Linked list implementation in a struct.
struct Node {
    Vehicle* data;
    Node* next;
};

// Queue struct containing the head of the list.
typedef struct {
    Node* head;
} Queue;

// Booth implementation in a struct.
typedef struct {
    int id;
    pthread_mutex_t booth_lock;
} Booth;
```

```c
// Ferry implementation in a struct.
typedef struct {
    int id;
    int port_id;
    int docked; // bool, 0 for sailing, 1 for waiting
    int waiting_amount;
    int ready_to_load;
    int ready_for_round_trip;
    Queue loading_line;
    pthread_mutex_t ferry_lock;
    pthread_mutex_t waiting_lock;
} Ferry;

// Port implementation in a struct.
typedef struct {
    int id;
    Booth booths[4];
    Queue waiting_lines[3];
    pthread_mutex_t waiting_line_lock;
    int current_line;
} Port;

// Function declarations for queue system.
void enqueue(Queue* list, Vehicle* v);
void dequeue(Queue* list);
int length(Queue* list);
void print_queue(Queue* list);
void new_queue(Queue* list);
// Function declaration for getting vehicle type.
char* get_vehicle_type(Vehicle* v);

#endif
```

Figure 2: structs.h File Including Typedef Structs and Function Declarations.

In Figure 2, typedef structs have been created for Vehicle, Node, Queue, Booth, Ferry and Port types. There are function declarations for queue operations and one function declaration for getting vehicle type as a string.

```c
#include <stdio.h>
#include <stdlib.h>
#include "structs.h"

// For adding a new vehicle to the end of the queue.
void enqueue(Queue* list, Vehicle* v) {
    Node* new = (Node*)malloc(sizeof(Node));
    new->data = v;
    new->next = NULL;
    if (list->head == NULL) {
        // No head, make vehicle the start.
        list->head = new;
    } else {
        // Add vehicle to the end.
        Node* current = list->head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = new;
    }
}

// For removing a vehicle from the start of the queue.
void dequeue(Queue* list) {
    // Check if queue is empty.
    if (list->head == NULL) {
        printf("ERROR: Cannot dequeue from an empty list.\n");
        return;
    }
    // Check if queue has only one vehicle.
    if (list->head->next == NULL) {
        list->head = NULL;
        return;
    }
    // Assign second vehicle as first vehicle in a queue.
    list->head = list->head->next;
}

// For getting the total sum of types (units) of vehicles from a queue.
int length(Queue* list) {
    int sum = 0;
    Node* current = list->head;
    while (current != NULL) {
        sum += current->data->type;
        current = current->next;
    }
    return sum;
}

// For initializing a new queue.
void new_queue(Queue* list) {
    list = (Queue*)malloc(sizeof(Queue));
    list->head = NULL;
}
```

Figure 3: Part 1 of structs.c File Including Queue Functions.

```c
// For getting the string of the vehicle type.
char* get_vehicle_type(Vehicle* v) {
    char* name;
    switch (v->type)
    {
        case 1:
            name = "Motorcycle";
            break;
        case 2:
            name = "Car";
            break;
        case 3:
            name = "Bus";
            break;
        case 4:
            name = "Truck";
            break;
        default:
            name = "UNKNOWN";
            break;
    }
    return name;
}

// For printing queue data, debugging purposes only.
void print_queue(Queue* list) {
    Node* current = list->head;
    printf("------HEAD------\n");
    while (current != NULL) {
        printf("---------------\n");
        printf("Vehicle ID: %d\n", current->data->id);
        printf("Type: %d\n", current->data->type);
        printf("Queue Length: %d\n", length(list));
        printf("---------------\n");
        current = current->next;
    }
}
```

Figure 4: Part 2 of structs.c Including String Functions.

In Figure 3 and 4, queue and print functions can be observed. Since we need queues for waiting and loading lines and they are implemented as a linked list structure, there are enqueue and dequeue functions to add and remove vehicles from the queue. Length traverses the list and sums up the vehicle types, which are also their units. New queue allocates a memory space for the given queue. Get vehicle type returns the vehicle type as its string equivalent. Print queue outputs the given queue with all of its vehicles and their data.

```c
int main() {
    // Set seed for rng.
    srand(time(NULL) % 306);
    printf("INFO: Initialization begun.\n");
    pthread_mutex_init(&print_lock, NULL);
    // Create 2 Ports.
    for (int i = 0; i < 2; i++) {
        ports[i].id = i;
        printf("INFO: Created new port with id %d.\n", ports[i].id);
        // Create 4 Booths.
        for (int j = 0; j < 4; j++) {
            ports[i].booths[j].id = j;
            int result = pthread_mutex_init(&ports[i].booths[j].booth_lock, NULL);
            if (result != 0) {
                printf("ERROR: Could not initialize mutex lock for booth.");
                exit(1);
            }
            printf("INFO: Created new booth with id %d.\n", ports[i].booths[j].id);
        }
        // Create 3 Waiting Lines.
        for (int j = 0; j < 3; j++) {
            new_queue(&ports[i].waiting_lines[j]);
            printf("INFO: Created new waiting line with id %d in port %d.\n", j, ports[i].id);
        }
        int result = pthread_mutex_init(&ports[i].waiting_line_lock, NULL);
        if (result != 0) {
            printf("ERROR: Could not initialize mutex lock for waiting line.");
            exit(1);
        }
    }
```

Figure 5: Part 1 of the Main Function that initializes ports, booths and waiting lines.

In Figure 5, we have a for loop for creating 2 ports. Inside this loop, we have two nested loops for creating 4 booths and creating 3 waiting lines. Each booth also has a mutex lock initialized. For waiting lines, one mutex lock will control all of the lines. There is also exception checking when a mutex init fails.

```c
// Create 2 Ferries.
for (int i = 0; i < 2; i++) {
    ferries[i] = malloc(sizeof(Ferry));
    ferries[i]->id = i;
    ferries[i]->port_id = i;
    ferries[i]->docked = 1;
    int result = pthread_mutex_init(&ferries[i]->ferry_lock, NULL);
    if (result != 0) {
        printf("ERROR: Could not initialize mutex lock for ferry.");
        exit(1);
    }
    result = pthread_mutex_init(&ferries[i]->waiting_lock, NULL);
    if (result != 0) {
        printf("ERROR: Could not initialize mutex lock for ferry.");
        exit(1);
    }
    new_queue(&ferries[i]->loading_line);
    printf("INFO: Created new ferry with id %d in port %d.\n", ferries[i]->id, ports[i].id);
}
// Create NUM_VEHICLES Vehicles. (Default: 32)
for (int i = 0; i < NUM_VEHICLES; i++) {
    vehicles[i] = malloc(sizeof(Vehicle));
    vehicles[i]->id = i;
    vehicles[i]->special = rand() % 2;
    vehicles[i]->port_id = rand() % 2;
    vehicles[i]->booth_id = -1;
    vehicle_start_ports[i] = vehicles[i]->port_id;
    // Assign type (unit) based on vehicle id.
    if (i < 8) {
        vehicles[i]->type = 1; // Motorcycle
    } else if (i < 16) {
        vehicles[i]->type = 2; // Car
    } else if (i < 24) {
        vehicles[i]->type = 3; // Bus
    } else {
        vehicles[i]->type = 4; // Truck
    }
}
```

Figure 6: Part 2 of the Main Function that is Responsible for Ferry and Vehicle Creation

In Figure 6, ferries used for transportation are created. Here, their ids and ports are determined. Also mutex locks for ferry_lock and waiting_lock are initialized. Last but not least, the ferry's loading line is created with the new_queue function. In addition, the creation of vehicles is provided. Id, type, special and booth data is initialized here. The types are determined by the id of the vehicle. As a requirement, 8 vehicles of each type are assigned. By default, 32 vehicles are created.

```
    printf("INFO: Initialization done.\n");
    printf("INFO: Creating threads.\n");
    // Create 2 Ferry threads with ferry_thread func and assign Ferry data from ferries[] array.
    for (int i = 0; i < 2; i++) {
        pthread_create(&ferry_threads[i], NULL, ferry_thread, ferries[i]);
    }
    // Create NUM_VEHICLES Vehicle threads with vehicle_thread func and assign Vehicle data from
vehicles[] array.
    for (int i = 0; i < NUM_VEHICLES; i++) {
        pthread_create(&vehicle_threads[i], NULL, vehicle_thread, vehicles[i]);
    }
    // Join vehicle threads and wait for all of them to finish.
    for (int i = 0; i < NUM_VEHICLES; i++) {
        pthread_join(vehicle_threads[i], NULL);
    }
    printf("INFO: Vehicle threads are done. Waiting for ferries..\n");
    // Join ferry threads last and wait for both of them to finish.
    for (int i = 0; i < 2; i++) {
        pthread_join(ferry_threads[i], NULL);
    }
    printf("INFO: Ferry threads are done. Testing completeness..\n");
    // Test if every vehicle has made a round trip and came back to their starting position.
    // Prints out if the program was successful or not.
    int complete = 1;
    for (int i = 0; i < NUM_VEHICLES; i++) {
        if (vehicle_start_ports[i] != vehicle_end_ports[i]) {
            printf("INFO: Vehicle (%d) started on port %d but ended on port %d!\n", i,
vehicle_start_ports[i], vehicle_end_ports[i]);
            complete = 0;
        }
    }
    if (complete) {
        printf("INFO: %d/%d checks are complete. Every vehicle has made a round trip. Success!\n",
NUM_VEHICLES, NUM_VEHICLES);
    } else {
        printf("INFO: Not every vehicle has made a round trip. Fail!\n");
    }
    return 0;
}
```

Figure 7: Part 3 of the Main Function That is Responsible for Ferry and Vehicle Threads

In Figure 7, the threads of vehicles and ferries are created. After that, all of the vehicle and ferry threads are joined so that the main thread waits for all of the threads to exit, and then it will continue. When all of the threads are finished, the correctness of the vehicles' round trip status are tested. They are tested by checking the starting port id and the ending port id, which should be the same if every vehicle has to make a round trip. Accordingly, the results obtained are printed.

```
void* ferry_thread(void* arg) {
    // Grab ferry pointer from parameter.
    Ferry* f = (Ferry*)arg;
    // For tracking how many trips has a ferry made.
    int repetition = 0;
    while (1) {
        int done;
        while (1) {
            // If all the vehicles have terminated, ferry has no reason to make any more trips.
            for (int c = 0; c < NUM_VEHICLES; c++) {
                if (vehicle_threads_completed[c] == 0) {
                    done = 0;
                    break;
                }
                done = 1;
            }
            if (done) {
                break;
            }
            sleep(1);
            pthread_mutex_lock(&f->waiting_lock);
            f->waiting_amount++;
            // For waiting either 30 seconds OR the whole waiting lines in the port to almost fill up, so
the ferry can start loading vehicles.
            if (repetition == 0) {
                if (f->waiting_amount >= 30 || (length(&ports[f->port_id].waiting_lines[0]) > 16 &&
length(&ports[f->port_id].waiting_lines[1]) > 16 && length(&ports[f->port_id].waiting_lines[2]) > 16)) {
                    f->ready_to_load = 1;
                }
            } else {
                f->ready_to_load = 1;
            }
            pthread_mutex_unlock(&f->waiting_lock);
            // Lock the ferry and waiting lines.
            pthread_mutex_lock(&f->ferry_lock);
            pthread_mutex_lock(&ports[f->port_id].waiting_line_lock);
```

Figure 8: The Ferry Thread Function

In Figure 8, the ferry pointer is grabbed from the parameter that was passed during the thread creation process. Since we have the pointer, we can read and modify data as we wish on this and it will update the ferry stored in the global resource.

Repetition count is tracked on ferry departures. This is necessary for the first iteration where ferries have to wait until 30 seconds or waiting lines to be full in order to accept vehicles into the loading line.

At the start of every repetition, the status of every vehicle thread is checked. If all of the threads are completed, the done flag is marked as true and the loop is broken since there is no need for any ferry transportation.

In every read/write action to the ferry data, necessary mutex locks are locked and unlocked.

```
            // Check if ferry is full OR there are no more vehicles that ferry can pick up AND ferry is
not empty.
            // OR check if current port has no vehicles AND target port has vehicles AND target port has
no ferries. This is in order to rescue trapped vehicles in a port.
            if (((length(&f->loading_line) == 30 || !available_vehicle_left(f)) && length(&f-
>loading_line) != 0) ||
                ((get_total_vehicles_in_port(f->port_id) == 0 && get_total_vehicles_in_port((f->port_id
== 0) ? 1 : 0) != 0) && (!ferry_in_port((f->port_id == 0) ? 1 : 0)))) {
                // We can move to the other port.
                pthread_mutex_unlock(&ports[f->port_id].waiting_line_lock);
                pthread_mutex_lock(&print_lock);
                printf("UPDATE: Ferry%d is moving to Port %d.\n", f->id, (f->port_id == 0) ? 1 : 0);
                pthread_mutex_unlock(&print_lock);
                pthread_mutex_lock(&f->waiting_lock);
                // Disable vehicle loading.
                f->ready_to_load = 0;
                pthread_mutex_unlock(&f->waiting_lock);
                // Reset wait counter, undock the ferry.
                f->docked = 0;
                f->waiting_amount = 0;
                // Take your time according to target port, and then change your port status.
                if (f->port_id == 0) {
                    sleep(6);
                    f->port_id = 1;
                } else if (f->port_id == 1) {
                    sleep(4);
                    f->port_id = 0;
                }
                // Change port id of every vehicle inside the ferry loading line.
                Node* current = f->loading_line.head;
                while (current != NULL) {
                    current->data->port_id = f->port_id;
                    current = current->next;
                }
```

Figure 9: Part 2 of the Ferry Thread Function Port and Ferry Checker

In Figure 9, it is checked whether the ferry is full or there are no more vehicles that ferry can accept to its loading line and the ferry is not empty. However, in edge cases, if the current port has no vehicles left but the other port has vehicles left and also there isn't any ferry on the other port, the ferry goes to the other port in order to rescue the left vehicles.

Once these are true, the ferry starts moving to the other port. It also changes some of its values during its departure. Depending on the target port, the ferry thread sleeps for a specified amount of time. Lastly, the ferry thread also changes the port id data of every vehicle inside the loading line to the target port id.

```
                // Ferry has arrived to the new port. Dock and signal that you are ready for a round
trip.
                pthread_mutex_lock(&print_lock);
                printf("UPDATE: Ferry%d arrived at Port %d.\n", f->id, f->port_id);
                pthread_mutex_unlock(&print_lock);
                f->docked = 1;
                f->ready_for_round_trip = 1;
                pthread_mutex_unlock(&f->ferry_lock);
                break;
            }
            pthread_mutex_unlock(&ports[f->port_id].waiting_line_lock);
            pthread_mutex_unlock(&f->ferry_lock);
        }
        // If all the vehicles have terminated, ferry has no reason to make any more trips.
        if (done) {
            break;
        }
        // Wait until all the vehicles have been unloaded from the ferry loading line.
        while (1) {
            msleep(100);
            pthread_mutex_lock(&f->ferry_lock);
            if (length(&f->loading_line) == 0) {
                pthread_mutex_lock(&print_lock);
                printf("UPDATE: Ferry%d is fully unloaded on Port %d.\n", f->id, f->port_id);
                pthread_mutex_unlock(&print_lock);
                pthread_mutex_unlock(&f->ferry_lock);
                break;
            }
            pthread_mutex_unlock(&f->ferry_lock);
        }
        repetition++;
    }
    pthread_exit(NULL);
}
```

Figure 10: Part 3 of the Ferry Thread Function Port and Ferry Checker

In Figure 10, the ferry thread arrives at the target port id and docks there. Done flag is checked here as well in order to escape the master while loop.

Continuing on, the ferry thread checks whether it is totally empty, then prints out that it is fully unloaded. Unloading process is done inside the vehicle threads.

```c
void* vehicle_thread(void* arg) {
    // Grab vehicle pointer from parameter.
    Vehicle* v = (Vehicle*)arg;
    // Select random booth based on if you are a special passenger or not.
    v->booth_id = rand() % (v->special ? 4 : 3);
    // Try to talk to booth.
    pthread_mutex_lock(&ports[v->port_id].booths[v->booth_id].booth_lock);
    pthread_mutex_lock(&print_lock);
    printf("UPDATE: %s (%d) approaches to Booth%d on Port %d.\n", get_vehicle_type(v), v->id, ports[v-
>port_id].booths[v->booth_id].id, ports[v->port_id].id);
    pthread_mutex_unlock(&print_lock);
    // Try to get in a waiting line.
    int line = 0;
    while (1) {
        msleep(100);
        pthread_mutex_lock(&ports[v->port_id].waiting_line_lock);
        // If the waiting line length would not exceed 20 if you were to join in.
        if (length(&ports[v->port_id].waiting_lines[line]) + v->type <= 20) {
            // Add vehicle to the specified waiting line.
            enqueue(&ports[v->port_id].waiting_lines[line], v);
            pthread_mutex_lock(&print_lock);
            printf("UPDATE: %s (%d) enters Line%d on Port %d.\n", get_vehicle_type(v), v->id, line,
ports[v->port_id].id);
            pthread_mutex_unlock(&print_lock);
            pthread_mutex_unlock(&ports[v->port_id].waiting_line_lock);
            pthread_mutex_unlock(&ports[v->port_id].booths[v->booth_id].booth_lock);
            break;
        } else {
            // This line is full, check other lines in a circular manner.
            // pthread_mutex_lock(&print_lock);
            // printf("UPDATE: %s (%d) is waiting in the booth since Line%d is full on Port %d.\n",
get_vehicle_type(v), v->id, line, ports[v->port_id].id);
            // pthread_mutex_unlock(&print_lock);
            pthread_mutex_unlock(&ports[v->port_id].waiting_line_lock);
            line = (line + 1) % 3;
        }
    }
```

Figure 11: Vehicle Thread Function

In Figure 11, the vehicle pointer is grabbed from the parameter that was passed during the thread creation process. Since we have the pointer, we can read and modify data as we wish on this and it will update the vehicle stored in the global resource.

First, a booth is randomly selected. If the vehicle is special, it can choose from 4 booths, while a normal vehicle can choose from 3 booths. When going to the booth, the booth's mutex is locked so that no one else can enter the booth. After this, the vehicle needs to enter a waiting line of the port that is big enough to accommodate itself. Each waiting line is 20 units long. First, it locks the waiting line mutex and when it finds an empty space on the waiting line, it enters the queue of said line and unlocks the waiting line mutex as well as booth mutex. If it can’t find an empty line, it checks the next line in a circular manner.

```
    int ferry_id;
    // Try to board the ferry.
    while (1) {
        msleep(100);
        int select = 0;
        while (!select) {
            // Check all ferries: if the ferry port id is the same as vehicle port id
            // AND the ferry is docked, that's the ferry vehicle is going to board.
            for (int f = 0; f < 2; f++) {
                pthread_mutex_lock(&ferries[f]->ferry_lock);
                if (ferries[f]->port_id == v->port_id && ferries[f]->docked) {
                    ferry_id = f;
                    pthread_mutex_unlock(&ferries[f]->ferry_lock);
                    select = 1;
                    break;
                }
                pthread_mutex_unlock(&ferries[f]->ferry_lock);
            }
        }
        Node* current = ports[v->port_id].waiting_lines[ports[v->port_id].current_line].head;
        pthread_mutex_lock(&ferries[ferry_id]->ferry_lock);
        pthread_mutex_lock(&ports[v->port_id].waiting_line_lock);
        pthread_mutex_lock(&ferries[ferry_id]->waiting_lock);
        // Check the current line's head vehicle and try to board it to the ferry
        if (current != NULL && ferries[ferry_id]->docked && current->data->id == v->id &&
ferries[ferry_id]->ready_to_load) {
            int boarded = board_ferry(ferries[ferry_id], v);
            pthread_mutex_unlock(&ferries[ferry_id]->waiting_lock);
            // Vehicle could not board due to space, go to next line in a circular manner.
            if (!boarded) {
                ports[v->port_id].current_line = (ports[v->port_id].current_line + 1) % 3;
                pthread_mutex_unlock(&ports[v->port_id].waiting_line_lock);
                pthread_mutex_unlock(&ferries[ferry_id]->ferry_lock);
                continue;
            }
            // Vehicle successfully boarded.
            pthread_mutex_unlock(&ports[v->port_id].waiting_line_lock);
            pthread_mutex_unlock(&ferries[ferry_id]->ferry_lock);
            break;
        }
```

Figure 12: Part 2 of the Vehicle Thread Function

In Figure 12, first every ferry is searched and matched in order to select which ferry the current vehicle will board. This ferry has to be in the same port as the vehicle and also has to be docked.

When selected, ferry_lock and waiting_line_lock mutexes are locked and the current line queue starts boarding the ferry one by one, starting from the head. Once the vehicle thread boards into the ferry, ferry_lock and waiting_line_lock are unlocked.

```
        // If the waiting line is empty, go to next line in a circular manner.
        if (current == NULL) {
            ports[v->port_id].current_line = (ports[v->port_id].current_line + 1) % 3;
        }
        pthread_mutex_unlock(&ferries[ferry_id]->waiting_lock);
        pthread_mutex_unlock(&ports[v->port_id].waiting_line_lock);
        pthread_mutex_unlock(&ferries[ferry_id]->ferry_lock);
    }
    // Wait until docks back.
    while (1) {
        pthread_mutex_lock(&ferries[ferry_id]->ferry_lock);
        if (ferries[ferry_id]->docked) {
            pthread_mutex_unlock(&ferries[ferry_id]->ferry_lock);
            break;
        }
        pthread_mutex_unlock(&ferries[ferry_id]->ferry_lock);
    }
    // Try to unload vehicle in new port.
    while (1) {
        msleep(100);
        pthread_mutex_lock(&ferries[ferry_id]->ferry_lock);
        pthread_mutex_lock(&ferries[ferry_id]->waiting_lock);
        // Check the loading line's head vehicle and try to unload it from the ferry.
        if (ferries[ferry_id]->loading_line.head != NULL && ferries[ferry_id]->ready_for_round_trip &&
ferries[ferry_id]->loading_line.head->data->id == v->id && !ferries[ferry_id]->ready_to_load) {
            // Unload from the start of the queue.
            pthread_mutex_unlock(&ferries[ferry_id]->waiting_lock);
            pthread_mutex_lock(&print_lock);
            printf("UPDATE: %s (%d) unloaded on Port %d.\n", get_vehicle_type(v), v->id, ports[v-
>port_id].id);
            pthread_mutex_unlock(&print_lock);
            // Reset vehicle's booth id.
            v->booth_id = -1;
            dequeue(&ferries[ferry_id]->loading_line);
            pthread_mutex_unlock(&ferries[ferry_id]->ferry_lock);
            break;
        }
        pthread_mutex_unlock(&ferries[ferry_id]->waiting_lock);
        pthread_mutex_unlock(&ferries[ferry_id]->ferry_lock);
    }
```

Figure 13: Part 3 of the Vehicle Thread Function

In Figure 13, boarding of the ferry function is continued. If the current line queue is empty, it goes to the next line in a circular manner.

Then, assuming the ferry is undocked and traveling to the other port, the vehicle thread constantly checks if it docks again with locking and unlocking the ferry_lock mutex.

When landed, vehicles start unloading from the start of the ferry loading line one by one, starting from the head. Once landed, the vehicle’s booth id is reset. Here, these operations are also accompanied by mutexes.

```
    // Start again.
    sleep((rand() % 5) + 1);
    // Select random booth based on if you are a special passenger or not.
    v->booth_id = rand() % (v->special ? 4 : 3);
    // Try to talk to booth.
    pthread_mutex_lock(&ports[v->port_id].booths[v->booth_id].booth_lock);
    pthread_mutex_lock(&print_lock);
    printf("UPDATE: %s (%d) approaches to Booth%d on Port %d.\n", get_vehicle_type(v), v->id, ports[v-
>port_id].booths[v->booth_id].id, ports[v->port_id].id);
    pthread_mutex_unlock(&print_lock);
    // Try to get in a waiting line.
    line = 0;
    while (1) {
        msleep(100);
        pthread_mutex_lock(&ports[v->port_id].waiting_line_lock);
        // If the waiting line length would not exceed 20 if you were to join in.
        if (length(&ports[v->port_id].waiting_lines[line]) + v->type <= 20) {
            // Add vehicle to the specified waiting line.
            enqueue(&ports[v->port_id].waiting_lines[line], v);
            pthread_mutex_lock(&print_lock);
            printf("UPDATE: %s (%d) enters Line%d on Port %d.\n", get_vehicle_type(v), v->id, line,
ports[v->port_id].id);
            pthread_mutex_unlock(&print_lock);
            pthread_mutex_unlock(&ports[v->port_id].waiting_line_lock);
            pthread_mutex_unlock(&ports[v->port_id].booths[v->booth_id].booth_lock);
            break;
        } else {
            // This line is full, check other lines in a circular manner.
            // pthread_mutex_lock(&print_lock);
            // printf("UPDATE: %s (%d) is waiting in the booth since Line%d is full on Port %d.\n",
get_vehicle_type(v), v->id, line, ports[v->port_id].id);
            // pthread_mutex_unlock(&print_lock);
            pthread_mutex_unlock(&ports[v->port_id].waiting_line_lock);
            line = (line + 1) % 3;
        }
    }
```

Figure 14: Part 4 of the Vehicle Thread Function

In Figure 14, this part marks the start of the round trip part where the vehicle thread has to do all the things that it did one more time. Vehicle sleeps for a random amount of time between 1 and 5 seconds. After being awoken, it selects a new booth to go to. Rest of the code is the exact copy of Figure 11.

```
    // Try to board the ferry.
    while (1) {
        msleep(100);
        int select = 0;
        while (!select) {
            // Check all ferries: if the ferry port id is the same as vehicle port id
            // AND the ferry is docked, that's the ferry vehicle is going to board.
            for (int f = 0; f < 2; f++) {
                pthread_mutex_lock(&ferries[f]->ferry_lock);
                if (ferries[f]->port_id == v->port_id && ferries[f]->docked) {
                    ferry_id = f;
                    pthread_mutex_unlock(&ferries[f]->ferry_lock);
                    select = 1;
                    break;
                }
                pthread_mutex_unlock(&ferries[f]->ferry_lock);
            }
        }
        Node* current = ports[v->port_id].waiting_lines[ports[v->port_id].current_line].head;
        pthread_mutex_lock(&ferries[ferry_id]->ferry_lock);
        pthread_mutex_lock(&ports[v->port_id].waiting_line_lock);
        pthread_mutex_lock(&ferries[ferry_id]->waiting_lock);
        // Check the current line's head vehicle and try to board it to the ferry
        if (current != NULL && ferries[ferry_id]->docked && current->data->id == v->id &&
ferries[ferry_id]->ready_to_load) {
            int boarded = board_ferry(ferries[ferry_id], v);
            pthread_mutex_unlock(&ferries[ferry_id]->waiting_lock);
            // Vehicle could not board due to space, go to next line in a circular manner.
            if (!boarded) {
                ports[v->port_id].current_line = (ports[v->port_id].current_line + 1) % 3;
                pthread_mutex_unlock(&ports[v->port_id].waiting_line_lock);
                pthread_mutex_unlock(&ferries[ferry_id]->ferry_lock);
                continue;
            }
            // Vehicle successfully boarded.
            pthread_mutex_unlock(&ports[v->port_id].waiting_line_lock);
            pthread_mutex_unlock(&ferries[ferry_id]->ferry_lock);
            break;
        }
        // If the waiting line is empty, go to next line in a circular manner.
        if (current == NULL) {
            ports[v->port_id].current_line = (ports[v->port_id].current_line + 1) % 3;
        }
        pthread_mutex_unlock(&ferries[ferry_id]->waiting_lock);
        pthread_mutex_unlock(&ports[v->port_id].waiting_line_lock);
        pthread_mutex_unlock(&ferries[ferry_id]->ferry_lock);
    }
```

Figure 15: Part 5 of the Vehicle Thread Function

Figure 15 is the continuation of the second trip which is the exact copy of Figure 12.

```
    // Wait until docks back.
    while (1) {
        pthread_mutex_lock(&ferries[ferry_id]->ferry_lock);
        if (ferries[ferry_id]->docked) {
            pthread_mutex_unlock(&ferries[ferry_id]->ferry_lock);
            break;
        }
        pthread_mutex_unlock(&ferries[ferry_id]->ferry_lock);
    }
    // Try to unload vehicle in new port.
    while (1) {
        msleep(100);
        pthread_mutex_lock(&ferries[ferry_id]->ferry_lock);
        pthread_mutex_lock(&ferries[ferry_id]->waiting_lock);
        // Check the loading line's head vehicle and try to unload it from the ferry
        if (ferries[ferry_id]->loading_line.head != NULL && ferries[ferry_id]->ready_for_round_trip &&
ferries[ferry_id]->loading_line.head->data->id == v->id && !ferries[ferry_id]->ready_to_load) {
            // Unload from the start of the queue.
            pthread_mutex_unlock(&ferries[ferry_id]->waiting_lock);
            pthread_mutex_lock(&print_lock);
            printf("UPDATE: %s (%d) unloaded on Port %d.\n", get_vehicle_type(v), v->id, ports[v-
>port_id].id);
            pthread_mutex_unlock(&print_lock);
            // Reset vehicle's booth id.
            v->booth_id = -1;
            dequeue(&ferries[ferry_id]->loading_line);
            pthread_mutex_unlock(&ferries[ferry_id]->ferry_lock);
            break;
        }
        pthread_mutex_unlock(&ferries[ferry_id]->waiting_lock);
        pthread_mutex_unlock(&ferries[ferry_id]->ferry_lock);
    }
    vehicle_threads_completed[v->id] = 1;
    vehicle_end_ports[v->id] = v->port_id;
    pthread_exit(NULL);
}
```

Figure 16: Part 6 of the Vehicle Thread Function

Figure 16 is the continuation of the second trip which is the exact copy of Figure 13.

Once it unloads for the last time in the same port that the vehicle started, it marks the id of the thread as completed in the vehicle_threads_completed array, and also adds the end port id to the vehicle_end_ports id for testing purposes at the end of the main function.

```c
int board_ferry(Ferry* f, Vehicle* v) {
    if (v->type <= 30 - length(&f->loading_line)) {
        // We can board since there is enough space.
        pthread_mutex_lock(&print_lock);
        printf("UPDATE: %s (%d) is loaded to Ferry%d on Port %d.\n", get_vehicle_type(v), v->id, f->id,
ports[v->port_id].id);
        pthread_mutex_unlock(&print_lock);
        // Add to the ferry loading line and remove from waiting line.
        enqueue(&f->loading_line, v);
        dequeue(&ports[v->port_id].waiting_lines[ports[v->port_id].current_line]);
        return 1;
    }
    return 0;
}

int available_vehicle_left(Ferry* f) {
    // Check every waiting line to see if there are any vehicle in them or not.
    for (int i = 0; i < 3; i++) {
        msleep(100);
        if (ports[f->port_id].waiting_lines[i].head != NULL && ports[f->port_id].waiting_lines[i].head-
>data->type <= 30 - length(&f->loading_line) && !vehicle_left_in_booths(f->port_id)) {
            return 1;
        }
    }
    return 0;
}

int get_total_vehicles_in_port(int port_id) {
    // Count the total amount of vehicles inside a specified port.
    int sum = 0;
    for (int i = 0; i < NUM_VEHICLES; i++) {
        if (vehicles[i]->port_id == port_id && vehicle_threads_completed[i] == 0) {
            sum++;
        }
    }
    return sum;
}
```

Figure 17: board_ferry, available_vehicle_left, get_total_vehicles_in_port Functions

In Figure 17, there is a function in the board_ferry function that allows the vehicles to be loaded onto the ferry. Ferry and vehicle pointers are retrieved from the parameter. It is checked whether the ferry has enough space to accommodate the given vehicle. If there is enough space, the vehicle in the waiting queue is loaded onto the ferry in turn and leave the waiting queue.

In the available_vehicle_left function, each waiting line created at the port where the ferry is located is checked one by one to see if there are any vehicles that can board the ferry. This is checked by whether the ferry has enough space to accommodate the vehicle. It also checks if there aren't any vehicles in the booth section of the port.

Get_total_vehicles_in_port function counts the number of vehicles in a given port.

```c
int vehicle_left_in_booths(int port_id) {
    // Check if there is any vehicle in the booths section.
    for (int i = 0; i < NUM_VEHICLES; i++) {
        if (vehicles[i]->port_id == port_id && vehicles[i]->booth_id == -1) {
            return 1;
        }
    }
    return 0;
}

int ferry_in_port(int port_id) {
    // Check if there is any ferry in a specified port.
    for (int i = 0; i < 2; i++) {
        if (ferries[i]->port_id == port_id && ferries[i]->docked) {
            return 1;
        }
    }
    return 0;
}

void msleep(int ms) {
    struct timespec time;
    time.tv_sec = 0;
    time.tv_nsec = ms * 1000000;
    nanosleep(&time, NULL);
}
```

Figure 18: vehicle_left_in_booths, ferry_in_port and msleep Functions

In Figure 18, the vehicle_left_in_booths function checks if there are any vehicles that are on that port but not in the waiting line or not.

Ferry_in_port function checks if there are any docked ferries in a given port or not.

Msleep function is necessary for making a thread sleep in cases of less than a second. It gets milliseconds as input parameter and multiplies it by a factor of 6 to make it work with nanosleep() function provided by time.h and the first macro defined.

3. Results

Sample input/outputs have been given in Appendix 1, Appendix 2, Appendix 3, and Appendix 4. Appendix 1 contains 4 vehicles, Appendix 2 contains 8 vehicles, Appendix 3 contains 16 vehicles, and Appendix 4 contains 32 vehicles. Our code was tested multiple times with different configurations, and also on both Linux and Mac operating systems.

4. Conclusion

Since this project had a time constraint, there are certain improvements that it can have in the future. First of all, the following requirement: “If the vehicle in the current line cannot fit to the remaining space in the ferry, following vehicles in the current line..” could not be implemented. With a new pointer that traverses the current line queue and loads the ferry according to that, this requirement may be accomplished. Last but not least, and obviously, our code can be improved in ways of both computation and space management. There might be redundant sections in the code that we may have missed. However, overall, our skills in terms of the C programming language and the grasp of threads and process synchronization have dramatically improved during the time of this project.

Appendix 1

```
$ ./program
INFO: Initialization begun.
INFO: Created new port with id 0.
INFO: Created new booth with id 0.
INFO: Created new booth with id 1.
INFO: Created new booth with id 2.
INFO: Created new booth with id 3.
INFO: Created new waiting line with id 0 in port 0.
INFO: Created new waiting line with id 1 in port 0.
INFO: Created new waiting line with id 2 in port 0.
INFO: Created new port with id 1.
INFO: Created new booth with id 0.
INFO: Created new booth with id 1.
INFO: Created new booth with id 2.
INFO: Created new booth with id 3.
INFO: Created new waiting line with id 0 in port 1.
INFO: Created new waiting line with id 1 in port 1.
INFO: Created new waiting line with id 2 in port 1.
INFO: Created new ferry with id 0 in port 0.
INFO: Created new ferry with id 1 in port 1.
INFO: Initialization done.
INFO: Creating threads.
UPDATE: Motorcycle (0) approaches to Booth2 on Port 0.
UPDATE: Motorcycle (2) approaches to Booth2 on Port 1.
UPDATE: Motorcycle (3) approaches to Booth1 on Port 0.
UPDATE: Motorcycle (0) enters Line0 on Port 0.
UPDATE: Motorcycle (2) enters Line0 on Port 1.
UPDATE: Motorcycle (3) enters Line0 on Port 0.
UPDATE: Motorcycle (1) approaches to Booth2 on Port 0.
UPDATE: Motorcycle (1) enters Line0 on Port 0.
UPDATE: Motorcycle (2) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (0) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (3) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (1) is loaded to Ferry0 on Port 0.
UPDATE: Ferry1 is moving to Port 0.
UPDATE: Ferry0 is moving to Port 1.
UPDATE: Ferry1 arrived at Port 0.
UPDATE: Motorcycle (2) unloaded on Port 0.
UPDATE: Ferry1 is fully unloaded on Port 0.
UPDATE: Ferry0 arrived at Port 1.
UPDATE: Motorcycle (0) unloaded on Port 1.
UPDATE: Motorcycle (3) unloaded on Port 1.
UPDATE: Motorcycle (1) unloaded on Port 1.
UPDATE: Ferry0 is fully unloaded on Port 1.
UPDATE: Motorcycle (2) approaches to Booth1 on Port 0.
UPDATE: Motorcycle (2) enters Line0 on Port 0.
UPDATE: Motorcycle (3) approaches to Booth3 on Port 1.
UPDATE: Motorcycle (1) approaches to Booth2 on Port 1.
UPDATE: Motorcycle (2) is loaded to Ferry1 on Port 0.
UPDATE: Motorcycle (3) enters Line0 on Port 1.
UPDATE: Motorcycle (1) enters Line0 on Port 1.
```

UPDATE: Motorcycle (3) is loaded to Ferry0 on Port 1.
UPDATE: Motorcycle (1) is loaded to Ferry0 on Port 1.
UPDATE: Ferry0 is moving to Port 0.
UPDATE: Ferry1 is moving to Port 1.
UPDATE: Motorcycle (0) approaches to Booth2 on Port 1.
UPDATE: Motorcycle (0) enters Line0 on Port 1.
UPDATE: Ferry0 arrived at Port 0.
UPDATE: Motorcycle (3) unloaded on Port 0.
UPDATE: Motorcycle (1) unloaded on Port 0.
UPDATE: Ferry0 is fully unloaded on Port 0.
UPDATE: Ferry1 arrived at Port 1.
UPDATE: Motorcycle (2) unloaded on Port 1.
UPDATE: Ferry1 is fully unloaded on Port 1.
UPDATE: Motorcycle (0) is loaded to Ferry1 on Port 1.
UPDATE: Ferry1 is moving to Port 0.
UPDATE: Ferry0 is moving to Port 1.
UPDATE: Ferry1 arrived at Port 0.
UPDATE: Motorcycle (0) unloaded on Port 0.
INFO: Vehicle threads are done. Waiting for ferries..
UPDATE: Ferry1 is fully unloaded on Port 0.
UPDATE: Ferry0 arrived at Port 1.
UPDATE: Ferry0 is fully unloaded on Port 1.
INFO: Ferry threads are done. Testing completeness..
INFO: 4/4 checks are complete. Every vehicle has made a round trip. Success!

Appendix 2

```
$ ./program
INFO: Initialization begun.
INFO: Created new port with id 0.
INFO: Created new booth with id 0.
INFO: Created new booth with id 1.
INFO: Created new booth with id 2.
INFO: Created new booth with id 3.
INFO: Created new waiting line with id 0 in port 0.
INFO: Created new waiting line with id 1 in port 0.
INFO: Created new waiting line with id 2 in port 0.
INFO: Created new port with id 1.
INFO: Created new booth with id 0.
INFO: Created new booth with id 1.
INFO: Created new booth with id 2.
INFO: Created new booth with id 3.
INFO: Created new waiting line with id 0 in port 1.
INFO: Created new waiting line with id 1 in port 1.
INFO: Created new waiting line with id 2 in port 1.
INFO: Created new ferry with id 0 in port 0.
INFO: Created new ferry with id 1 in port 1.
INFO: Initialization done.
INFO: Creating threads.
UPDATE: Motorcycle (0) approaches to Booth3 on Port 0.
UPDATE: Motorcycle (1) approaches to Booth2 on Port 0.
UPDATE: Motorcycle (2) approaches to Booth3 on Port 1.
UPDATE: Motorcycle (3) approaches to Booth1 on Port 1.
UPDATE: Motorcycle (6) approaches to Booth0 on Port 1.
UPDATE: Motorcycle (0) enters Line0 on Port 0.
UPDATE: Motorcycle (3) enters Line0 on Port 1.
UPDATE: Motorcycle (1) enters Line0 on Port 0.
UPDATE: Motorcycle (5) approaches to Booth1 on Port 1.
UPDATE: Motorcycle (7) approaches to Booth2 on Port 0.
UPDATE: Motorcycle (2) enters Line0 on Port 1.
UPDATE: Motorcycle (4) approaches to Booth3 on Port 1.
UPDATE: Motorcycle (6) enters Line0 on Port 1.
UPDATE: Motorcycle (5) enters Line0 on Port 1.
UPDATE: Motorcycle (7) enters Line0 on Port 0.
UPDATE: Motorcycle (4) enters Line0 on Port 1.
UPDATE: Motorcycle (3) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (0) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (1) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (2) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (7) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (6) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (5) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (4) is loaded to Ferry1 on Port 1.
UPDATE: Ferry1 is moving to Port 0.
UPDATE: Ferry0 is moving to Port 1.
UPDATE: Ferry1 arrived at Port 0.
UPDATE: Motorcycle (3) unloaded on Port 0.
UPDATE: Motorcycle (2) unloaded on Port 0.
UPDATE: Motorcycle (6) unloaded on Port 0.
UPDATE: Motorcycle (5) unloaded on Port 0.
```

UPDATE: Motorcycle (4) unloaded on Port 0.
UPDATE: Ferry1 is fully unloaded on Port 0.
UPDATE: Motorcycle (2) approaches to Booth2 on Port 0.
UPDATE: Motorcycle (2) enters Line0 on Port 0.
UPDATE: Ferry0 arrived at Port 1.
UPDATE: Motorcycle (0) unloaded on Port 1.
UPDATE: Motorcycle (2) is loaded to Ferry1 on Port 0.
UPDATE: Motorcycle (3) approaches to Booth2 on Port 0.
UPDATE: Motorcycle (1) unloaded on Port 1.
UPDATE: Motorcycle (7) unloaded on Port 1.
UPDATE: Motorcycle (3) enters Line0 on Port 0.
UPDATE: Motorcycle (5) approaches to Booth2 on Port 0.
UPDATE: Ferry0 is fully unloaded on Port 1.
UPDATE: Motorcycle (3) is loaded to Ferry1 on Port 0.
UPDATE: Motorcycle (5) enters Line0 on Port 0.
UPDATE: Motorcycle (5) is loaded to Ferry1 on Port 0.
UPDATE: Ferry1 is moving to Port 1.
UPDATE: Motorcycle (1) approaches to Booth3 on Port 1.
UPDATE: Motorcycle (1) enters Line0 on Port 1.
UPDATE: Motorcycle (1) is loaded to Ferry0 on Port 1.
UPDATE: Motorcycle (6) approaches to Booth2 on Port 0.
UPDATE: Motorcycle (6) enters Line0 on Port 0.
UPDATE: Ferry0 is moving to Port 0.
UPDATE: Motorcycle (7) approaches to Booth0 on Port 1.
UPDATE: Motorcycle (4) approaches to Booth1 on Port 0.
UPDATE: Motorcycle (7) enters Line0 on Port 1.
UPDATE: Motorcycle (4) enters Line0 on Port 0.
UPDATE: Motorcycle (0) approaches to Booth1 on Port 1.
UPDATE: Motorcycle (0) enters Line0 on Port 1.
UPDATE: Ferry0 arrived at Port 0.
UPDATE: Motorcycle (1) unloaded on Port 0.
UPDATE: Ferry1 arrived at Port 1.
UPDATE: Motorcycle (2) unloaded on Port 1.
UPDATE: Ferry0 is fully unloaded on Port 0.
UPDATE: Motorcycle (3) unloaded on Port 1.
UPDATE: Motorcycle (5) unloaded on Port 1.
UPDATE: Ferry1 is fully unloaded on Port 1.
UPDATE: Motorcycle (6) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (4) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (7) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (0) is loaded to Ferry1 on Port 1.
UPDATE: Ferry0 is moving to Port 1.
UPDATE: Ferry1 is moving to Port 0.
UPDATE: Ferry1 arrived at Port 0.
UPDATE: Motorcycle (7) unloaded on Port 0.
UPDATE: Motorcycle (0) unloaded on Port 0.
UPDATE: Ferry1 is fully unloaded on Port 0.
UPDATE: Ferry0 arrived at Port 1.
UPDATE: Motorcycle (6) unloaded on Port 1.
UPDATE: Motorcycle (4) unloaded on Port 1.
INFO: Vehicle threads are done. Waiting for ferries..
UPDATE: Ferry0 is fully unloaded on Port 1.
INFO: Ferry threads are done. Testing completeness..
INFO: 8/8 checks are complete. Every vehicle has made a round trip. Success!

Appendix 3

```
$ ./program
INFO: Initialization begun.
INFO: Created new port with id 0.
INFO: Created new booth with id 0.
INFO: Created new booth with id 1.
INFO: Created new booth with id 2.
INFO: Created new booth with id 3.
INFO: Created new waiting line with id 0 in port 0.
INFO: Created new waiting line with id 1 in port 0.
INFO: Created new waiting line with id 2 in port 0.
INFO: Created new port with id 1.
INFO: Created new booth with id 0.
INFO: Created new booth with id 1.
INFO: Created new booth with id 2.
INFO: Created new booth with id 3.
INFO: Created new waiting line with id 0 in port 1.
INFO: Created new waiting line with id 1 in port 1.
INFO: Created new waiting line with id 2 in port 1.
INFO: Created new ferry with id 0 in port 0.
INFO: Created new ferry with id 1 in port 1.
INFO: Initialization done.
INFO: Creating threads.
UPDATE: Motorcycle (0) approaches to Booth2 on Port 1.
UPDATE: Motorcycle (1) approaches to Booth1 on Port 0.
UPDATE: Motorcycle (4) approaches to Booth1 on Port 1.
UPDATE: Motorcycle (5) approaches to Booth0 on Port 1.
UPDATE: Motorcycle (6) approaches to Booth0 on Port 0.
UPDATE: Car (10) approaches to Booth2 on Port 0.
UPDATE: Motorcycle (0) enters Line0 on Port 1.
UPDATE: Motorcycle (1) enters Line0 on Port 0.
UPDATE: Car (15) approaches to Booth2 on Port 1.
UPDATE: Motorcycle (2) approaches to Booth1 on Port 0.
UPDATE: Motorcycle (5) enters Line0 on Port 1.
UPDATE: Motorcycle (6) enters Line0 on Port 0.
UPDATE: Motorcycle (4) enters Line0 on Port 1.
UPDATE: Motorcycle (7) approaches to Booth0 on Port 1.
UPDATE: Car (10) enters Line0 on Port 0.
UPDATE: Car (14) approaches to Booth0 on Port 0.
UPDATE: Car (8) approaches to Booth1 on Port 1.
UPDATE: Car (13) approaches to Booth2 on Port 0.
UPDATE: Car (15) enters Line0 on Port 1.
UPDATE: Motorcycle (2) enters Line0 on Port 0.
UPDATE: Motorcycle (3) approaches to Booth1 on Port 0.
UPDATE: Motorcycle (7) enters Line0 on Port 1.
UPDATE: Car (14) enters Line0 on Port 0.
UPDATE: Car (9) approaches to Booth0 on Port 1.
UPDATE: Car (8) enters Line0 on Port 1.
UPDATE: Car (13) enters Line0 on Port 0.
UPDATE: Motorcycle (3) enters Line0 on Port 0.
UPDATE: Car (12) approaches to Booth1 on Port 0.
UPDATE: Car (9) enters Line0 on Port 1.
```

UPDATE: Car (11) approaches to Booth0 on Port 1.
UPDATE: Car (12) enters Line0 on Port 0.
UPDATE: Car (11) enters Line0 on Port 1.
UPDATE: Motorcycle (1) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (0) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (6) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (5) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (4) is loaded to Ferry1 on Port 1.
UPDATE: Car (10) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (2) is loaded to Ferry0 on Port 0.
UPDATE: Car (14) is loaded to Ferry0 on Port 0.
UPDATE: Car (15) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (7) is loaded to Ferry1 on Port 1.
UPDATE: Car (13) is loaded to Ferry0 on Port 0.
UPDATE: Car (8) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (3) is loaded to Ferry0 on Port 0.
UPDATE: Car (9) is loaded to Ferry1 on Port 1.
UPDATE: Car (12) is loaded to Ferry0 on Port 0.
UPDATE: Car (11) is loaded to Ferry1 on Port 1.
UPDATE: Ferry1 is moving to Port 0.
UPDATE: Ferry0 is moving to Port 1.
UPDATE: Ferry1 arrived at Port 0.
UPDATE: Motorcycle (0) unloaded on Port 0.
UPDATE: Motorcycle (5) unloaded on Port 0.
UPDATE: Motorcycle (4) unloaded on Port 0.
UPDATE: Car (15) unloaded on Port 0.
UPDATE: Motorcycle (7) unloaded on Port 0.
UPDATE: Car (8) unloaded on Port 0.
UPDATE: Car (9) unloaded on Port 0.
UPDATE: Car (11) unloaded on Port 0.
UPDATE: Ferry1 is fully unloaded on Port 0.
UPDATE: Ferry0 arrived at Port 1.
UPDATE: Motorcycle (1) unloaded on Port 1.
UPDATE: Motorcycle (6) unloaded on Port 1.
UPDATE: Car (10) unloaded on Port 1.
UPDATE: Motorcycle (2) unloaded on Port 1.
UPDATE: Car (14) unloaded on Port 1.
UPDATE: Car (13) unloaded on Port 1.
UPDATE: Car (8) approaches to Booth2 on Port 0.
UPDATE: Motorcycle (3) unloaded on Port 1.
UPDATE: Car (12) unloaded on Port 1.
UPDATE: Car (8) enters Line0 on Port 0.
UPDATE: Ferry0 is fully unloaded on Port 1.
UPDATE: Car (8) is loaded to Ferry1 on Port 0.
UPDATE: Ferry1 is moving to Port 1.
UPDATE: Car (14) approaches to Booth0 on Port 1.
UPDATE: Car (14) enters Line0 on Port 1.
UPDATE: Car (12) approaches to Booth2 on Port 1.
UPDATE: Car (11) approaches to Booth2 on Port 0.
UPDATE: Car (11) enters Line0 on Port 0.
UPDATE: Car (12) enters Line0 on Port 1.
UPDATE: Car (14) is loaded to Ferry0 on Port 1.
UPDATE: Car (12) is loaded to Ferry0 on Port 1.
UPDATE: Motorcycle (5) approaches to Booth1 on Port 0.
UPDATE: Motorcycle (5) enters Line0 on Port 0.

UPDATE: Car (13) approaches to Booth1 on Port 1.
UPDATE: Car (9) approaches to Booth3 on Port 0.
UPDATE: Motorcycle (3) approaches to Booth2 on Port 1.
UPDATE: Car (13) enters Line0 on Port 1.
UPDATE: Car (9) enters Line0 on Port 0.
UPDATE: Motorcycle (3) enters Line0 on Port 1.
UPDATE: Car (13) is loaded to Ferry0 on Port 1.
UPDATE: Motorcycle (3) is loaded to Ferry0 on Port 1.
UPDATE: Motorcycle (0) approaches to Booth3 on Port 0.
UPDATE: Motorcycle (1) approaches to Booth0 on Port 1.
UPDATE: Motorcycle (4) approaches to Booth0 on Port 0.
UPDATE: Motorcycle (0) enters Line0 on Port 0.
UPDATE: Car (15) approaches to Booth1 on Port 0.
UPDATE: Ferry0 is moving to Port 0.
UPDATE: Motorcycle (1) enters Line0 on Port 1.
UPDATE: Motorcycle (4) enters Line0 on Port 0.
UPDATE: Car (15) enters Line0 on Port 0.
UPDATE: Motorcycle (7) approaches to Booth2 on Port 0.
UPDATE: Motorcycle (7) enters Line0 on Port 0.
UPDATE: Motorcycle (6) approaches to Booth1 on Port 1.
UPDATE: Motorcycle (6) enters Line0 on Port 1.
UPDATE: Car (10) approaches to Booth0 on Port 1.
UPDATE: Car (10) enters Line0 on Port 1.
UPDATE: Motorcycle (2) approaches to Booth0 on Port 1.
UPDATE: Motorcycle (2) enters Line0 on Port 1.
UPDATE: Ferry1 arrived at Port 1.
UPDATE: Ferry0 arrived at Port 0.
UPDATE: Car (8) unloaded on Port 1.
UPDATE: Car (14) unloaded on Port 0.
UPDATE: Ferry1 is fully unloaded on Port 1.
UPDATE: Car (12) unloaded on Port 0.
UPDATE: Car (13) unloaded on Port 0.
UPDATE: Motorcycle (3) unloaded on Port 0.
UPDATE: Ferry0 is fully unloaded on Port 0.
UPDATE: Motorcycle (1) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (6) is loaded to Ferry1 on Port 1.
UPDATE: Car (10) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (2) is loaded to Ferry1 on Port 1.
UPDATE: Car (11) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (5) is loaded to Ferry0 on Port 0.
UPDATE: Car (9) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (0) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (4) is loaded to Ferry0 on Port 0.
UPDATE: Car (15) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (7) is loaded to Ferry0 on Port 0.
UPDATE: Ferry1 is moving to Port 0.
UPDATE: Ferry0 is moving to Port 1.
UPDATE: Ferry1 arrived at Port 0.
UPDATE: Motorcycle (1) unloaded on Port 0.
UPDATE: Motorcycle (6) unloaded on Port 0.
UPDATE: Car (10) unloaded on Port 0.
UPDATE: Motorcycle (2) unloaded on Port 0.
UPDATE: Ferry1 is fully unloaded on Port 0.
UPDATE: Ferry0 arrived at Port 1.
UPDATE: Car (11) unloaded on Port 1.

UPDATE: Motorcycle (5) unloaded on Port 1.
UPDATE: Car (9) unloaded on Port 1.
UPDATE: Motorcycle (0) unloaded on Port 1.
UPDATE: Motorcycle (4) unloaded on Port 1.
UPDATE: Car (15) unloaded on Port 1.
UPDATE: Motorcycle (7) unloaded on Port 1.
UPDATE: Ferry0 is fully unloaded on Port 1.
INFO: Vehicle threads are done. Waiting for ferries..
INFO: Ferry threads are done. Testing completeness..
INFO: 16/16 checks are complete. Every vehicle has made a round trip. Success!

Appendix 4

```
$ ./program
INFO: Initialization begun.
INFO: Created new port with id 0.
INFO: Created new booth with id 0.
INFO: Created new booth with id 1.
INFO: Created new booth with id 2.
INFO: Created new booth with id 3.
INFO: Created new waiting line with id 0 in port 0.
INFO: Created new waiting line with id 1 in port 0.
INFO: Created new waiting line with id 2 in port 0.
INFO: Created new port with id 1.
INFO: Created new booth with id 0.
INFO: Created new booth with id 1.
INFO: Created new booth with id 2.
INFO: Created new booth with id 3.
INFO: Created new waiting line with id 0 in port 1.
INFO: Created new waiting line with id 1 in port 1.
INFO: Created new waiting line with id 2 in port 1.
INFO: Created new ferry with id 0 in port 0.
INFO: Created new ferry with id 1 in port 1.
INFO: Initialization done.
INFO: Creating threads.
UPDATE: Motorcycle (0) approaches to Booth3 on Port 0.
UPDATE: Motorcycle (1) approaches to Booth0 on Port 1.
UPDATE: Motorcycle (5) approaches to Booth1 on Port 1.
UPDATE: Motorcycle (6) approaches to Booth1 on Port 0.
UPDATE: Car (8) approaches to Booth2 on Port 1.
UPDATE: Car (15) approaches to Booth0 on Port 0.
UPDATE: Bus (17) approaches to Booth3 on Port 1.
UPDATE: Truck (27) approaches to Booth2 on Port 0.
UPDATE: Motorcycle (1) enters Line0 on Port 1.
UPDATE: Motorcycle (0) enters Line0 on Port 0.
UPDATE: Motorcycle (5) enters Line0 on Port 1.
UPDATE: Motorcycle (2) approaches to Booth0 on Port 1.
UPDATE: Car (8) enters Line0 on Port 1.
UPDATE: Motorcycle (6) enters Line0 on Port 0.
UPDATE: Motorcycle (3) approaches to Booth3 on Port 0.
UPDATE: Motorcycle (7) approaches to Booth1 on Port 1.
UPDATE: Car (10) approaches to Booth2 on Port 1.
UPDATE: Bus (23) approaches to Booth1 on Port 0.
UPDATE: Car (15) enters Line0 on Port 0.
UPDATE: Bus (16) approaches to Booth0 on Port 0.
UPDATE: Bus (17) enters Line0 on Port 1.
UPDATE: Truck (28) approaches to Booth3 on Port 1.
UPDATE: Truck (27) enters Line0 on Port 0.
UPDATE: Truck (30) approaches to Booth2 on Port 0.
UPDATE: Motorcycle (2) enters Line0 on Port 1.
UPDATE: Motorcycle (3) enters Line0 on Port 0.
UPDATE: Car (11) approaches to Booth0 on Port 1.
UPDATE: Motorcycle (4) approaches to Booth3 on Port 0.
UPDATE: Bus (23) enters Line0 on Port 0.
```

UPDATE: Car (10) enters Line0 on Port 1.
UPDATE: Truck (24) approaches to Booth1 on Port 0.
UPDATE: Bus (16) enters Line0 on Port 0.
UPDATE: Car (12) approaches to Booth2 on Port 1.
UPDATE: Motorcycle (7) enters Line0 on Port 1.
UPDATE: Truck (25) approaches to Booth0 on Port 0.
UPDATE: Car (9) approaches to Booth1 on Port 1.
UPDATE: Truck (28) enters Line0 on Port 1.
UPDATE: Truck (30) enters Line0 on Port 0.
UPDATE: Car (11) enters Line0 on Port 1.
UPDATE: Motorcycle (4) enters Line0 on Port 0.
UPDATE: Bus (18) approaches to Booth0 on Port 1.
UPDATE: Car (12) enters Line0 on Port 1.
UPDATE: Car (13) approaches to Booth2 on Port 1.
UPDATE: Truck (24) enters Line1 on Port 0.
UPDATE: Truck (26) approaches to Booth1 on Port 0.
UPDATE: Truck (25) enters Line1 on Port 0.
UPDATE: Car (9) enters Line1 on Port 1.
UPDATE: Truck (29) approaches to Booth0 on Port 0.
UPDATE: Bus (19) approaches to Booth1 on Port 1.
UPDATE: Bus (18) enters Line1 on Port 1.
UPDATE: Bus (21) approaches to Booth0 on Port 1.
UPDATE: Car (13) enters Line1 on Port 1.
UPDATE: Car (14) approaches to Booth2 on Port 1.
UPDATE: Truck (29) enters Line1 on Port 0.
UPDATE: Bus (19) enters Line1 on Port 1.
UPDATE: Bus (20) approaches to Booth1 on Port 1.
UPDATE: Truck (26) enters Line1 on Port 0.
UPDATE: Bus (21) enters Line1 on Port 1.
UPDATE: Truck (31) approaches to Booth0 on Port 1.
UPDATE: Car (14) enters Line1 on Port 1.
UPDATE: Bus (20) enters Line1 on Port 1.
UPDATE: Bus (22) approaches to Booth1 on Port 1.
UPDATE: Truck (31) enters Line2 on Port 1.
UPDATE: Bus (22) enters Line2 on Port 1.
UPDATE: Motorcycle (0) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (1) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (5) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (6) is loaded to Ferry0 on Port 0.
UPDATE: Car (8) is loaded to Ferry1 on Port 1.
UPDATE: Car (15) is loaded to Ferry0 on Port 0.
UPDATE: Truck (27) is loaded to Ferry0 on Port 0.
UPDATE: Bus (17) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (3) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (2) is loaded to Ferry1 on Port 1.
UPDATE: Bus (23) is loaded to Ferry0 on Port 0.
UPDATE: Car (10) is loaded to Ferry1 on Port 1.
UPDATE: Bus (16) is loaded to Ferry0 on Port 0.
UPDATE: Truck (30) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (4) is loaded to Ferry0 on Port 0.
UPDATE: Truck (24) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (7) is loaded to Ferry1 on Port 1.
UPDATE: Truck (25) is loaded to Ferry0 on Port 0.
UPDATE: Truck (28) is loaded to Ferry1 on Port 1.
UPDATE: Car (11) is loaded to Ferry1 on Port 1.

UPDATE: Car (12) is loaded to Ferry1 on Port 1.
UPDATE: Car (9) is loaded to Ferry1 on Port 1.
UPDATE: Bus (18) is loaded to Ferry1 on Port 1.
UPDATE: Car (13) is loaded to Ferry1 on Port 1.
UPDATE: Bus (19) is loaded to Ferry1 on Port 1.
UPDATE: Ferry1 is moving to Port 0.
UPDATE: Ferry0 is moving to Port 1.
UPDATE: Ferry1 arrived at Port 0.
UPDATE: Motorcycle (1) unloaded on Port 0.
UPDATE: Motorcycle (5) unloaded on Port 0.
UPDATE: Car (8) unloaded on Port 0.
UPDATE: Bus (17) unloaded on Port 0.
UPDATE: Motorcycle (2) unloaded on Port 0.
UPDATE: Car (10) unloaded on Port 0.
UPDATE: Motorcycle (7) unloaded on Port 0.
UPDATE: Truck (28) unloaded on Port 0.
UPDATE: Car (11) unloaded on Port 0.
UPDATE: Car (12) unloaded on Port 0.
UPDATE: Car (9) unloaded on Port 0.
UPDATE: Bus (18) unloaded on Port 0.
UPDATE: Car (13) unloaded on Port 0.
UPDATE: Bus (19) unloaded on Port 0.
UPDATE: Ferry1 is fully unloaded on Port 0.
UPDATE: Motorcycle (2) approaches to Booth0 on Port 0.
UPDATE: Motorcycle (2) enters Line0 on Port 0.
UPDATE: Ferry0 arrived at Port 1.
UPDATE: Motorcycle (0) unloaded on Port 1.
UPDATE: Motorcycle (2) is loaded to Ferry1 on Port 0.
UPDATE: Motorcycle (6) unloaded on Port 1.
UPDATE: Car (15) unloaded on Port 1.
UPDATE: Truck (27) unloaded on Port 1.
UPDATE: Bus (17) approaches to Booth0 on Port 0.
UPDATE: Truck (29) is loaded to Ferry1 on Port 0.
UPDATE: Motorcycle (3) unloaded on Port 1.
UPDATE: Bus (23) unloaded on Port 1.
UPDATE: Bus (17) enters Line0 on Port 0.
UPDATE: Truck (26) is loaded to Ferry1 on Port 0.
UPDATE: Bus (16) unloaded on Port 1.
UPDATE: Truck (30) unloaded on Port 1.
UPDATE: Motorcycle (4) unloaded on Port 1.
UPDATE: Truck (24) unloaded on Port 1.
UPDATE: Truck (25) unloaded on Port 1.
UPDATE: Ferry0 is fully unloaded on Port 1.
UPDATE: Bus (17) is loaded to Ferry1 on Port 0.
UPDATE: Car (9) approaches to Booth0 on Port 0.
UPDATE: Bus (18) approaches to Booth1 on Port 0.
UPDATE: Car (9) enters Line0 on Port 0.
UPDATE: Bus (18) enters Line0 on Port 0.
UPDATE: Car (9) is loaded to Ferry1 on Port 0.
UPDATE: Bus (18) is loaded to Ferry1 on Port 0.
UPDATE: Motorcycle (1) approaches to Booth0 on Port 0.
UPDATE: Car (15) approaches to Booth2 on Port 1.
UPDATE: Car (15) enters Line0 on Port 1.
UPDATE: Ferry1 is moving to Port 1.
UPDATE: Motorcycle (1) enters Line0 on Port 0.

UPDATE: Car (10) approaches to Booth0 on Port 0.
UPDATE: Truck (25) approaches to Booth3 on Port 1.
UPDATE: Car (10) enters Line0 on Port 0.
UPDATE: Motorcycle (7) approaches to Booth0 on Port 0.
UPDATE: Car (12) approaches to Booth1 on Port 0.
UPDATE: Motorcycle (7) enters Line0 on Port 0.
UPDATE: Car (13) approaches to Booth3 on Port 0.
UPDATE: Bus (19) approaches to Booth0 on Port 0.
UPDATE: Car (12) enters Line0 on Port 0.
UPDATE: Car (13) enters Line0 on Port 0.
UPDATE: Bus (19) enters Line0 on Port 0.
UPDATE: Truck (25) enters Line0 on Port 1.
UPDATE: Bus (21) is loaded to Ferry0 on Port 1.
UPDATE: Car (14) is loaded to Ferry0 on Port 1.
UPDATE: Bus (20) is loaded to Ferry0 on Port 1.
UPDATE: Truck (31) is loaded to Ferry0 on Port 1.
UPDATE: Bus (22) is loaded to Ferry0 on Port 1.
UPDATE: Truck (27) approaches to Booth1 on Port 1.
UPDATE: Truck (27) enters Line0 on Port 1.
UPDATE: Car (15) is loaded to Ferry0 on Port 1.
UPDATE: Truck (25) is loaded to Ferry0 on Port 1.
UPDATE: Truck (27) is loaded to Ferry0 on Port 1.
UPDATE: Motorcycle (4) approaches to Booth1 on Port 1.
UPDATE: Truck (28) approaches to Booth3 on Port 0.
UPDATE: Car (11) approaches to Booth1 on Port 0.
UPDATE: Motorcycle (4) enters Line0 on Port 1.
UPDATE: Truck (28) enters Line0 on Port 0.
UPDATE: Car (11) enters Line0 on Port 0.
UPDATE: Motorcycle (4) is loaded to Ferry0 on Port 1.
UPDATE: Motorcycle (5) approaches to Booth1 on Port 0.
UPDATE: Ferry0 is moving to Port 0.
UPDATE: Motorcycle (5) enters Line0 on Port 0.
UPDATE: Car (8) approaches to Booth1 on Port 0.
UPDATE: Car (8) enters Line0 on Port 0.
UPDATE: Truck (24) approaches to Booth2 on Port 1.
UPDATE: Truck (24) enters Line0 on Port 1.
UPDATE: Bus (23) approaches to Booth2 on Port 1.
UPDATE: Bus (16) approaches to Booth1 on Port 1.
UPDATE: Bus (23) enters Line0 on Port 1.
UPDATE: Truck (30) approaches to Booth3 on Port 1.
UPDATE: Bus (16) enters Line0 on Port 1.
UPDATE: Truck (30) enters Line0 on Port 1.
UPDATE: Motorcycle (0) approaches to Booth0 on Port 1.
UPDATE: Motorcycle (6) approaches to Booth1 on Port 1.
UPDATE: Motorcycle (0) enters Line0 on Port 1.
UPDATE: Motorcycle (3) approaches to Booth2 on Port 1.
UPDATE: Motorcycle (6) enters Line0 on Port 1.
UPDATE: Motorcycle (3) enters Line0 on Port 1.
UPDATE: Ferry0 arrived at Port 0.
UPDATE: Bus (21) unloaded on Port 0.
UPDATE: Car (14) unloaded on Port 0.
UPDATE: Bus (20) unloaded on Port 0.
UPDATE: Ferry1 arrived at Port 1.
UPDATE: Motorcycle (2) unloaded on Port 1.
UPDATE: Truck (31) unloaded on Port 0.

UPDATE: Truck (29) unloaded on Port 1.
UPDATE: Bus (22) unloaded on Port 0.
UPDATE: Truck (26) unloaded on Port 1.
UPDATE: Car (15) unloaded on Port 0.
UPDATE: Bus (17) unloaded on Port 1.
UPDATE: Truck (25) unloaded on Port 0.
UPDATE: Car (9) unloaded on Port 1.
UPDATE: Bus (18) unloaded on Port 1.
UPDATE: Truck (27) unloaded on Port 0.
UPDATE: Ferry1 is fully unloaded on Port 1.
UPDATE: Motorcycle (4) unloaded on Port 0.
UPDATE: Ferry0 is fully unloaded on Port 0.
UPDATE: Truck (29) approaches to Booth2 on Port 1.
UPDATE: Truck (29) enters Line1 on Port 1.
UPDATE: Truck (24) is loaded to Ferry1 on Port 1.
UPDATE: Car (14) approaches to Booth0 on Port 0.
UPDATE: Bus (23) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (1) is loaded to Ferry0 on Port 0.
UPDATE: Car (10) is loaded to Ferry0 on Port 0.
UPDATE: Car (14) enters Line0 on Port 0.
UPDATE: Bus (16) is loaded to Ferry1 on Port 1.
UPDATE: Motorcycle (7) is loaded to Ferry0 on Port 0.
UPDATE: Truck (30) is loaded to Ferry1 on Port 1.
UPDATE: Car (12) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (0) is loaded to Ferry1 on Port 1.
UPDATE: Car (13) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (6) is loaded to Ferry1 on Port 1.
UPDATE: Bus (19) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (3) is loaded to Ferry1 on Port 1.
UPDATE: Truck (28) is loaded to Ferry0 on Port 0.
UPDATE: Car (11) is loaded to Ferry0 on Port 0.
UPDATE: Motorcycle (5) is loaded to Ferry0 on Port 0.
UPDATE: Truck (29) is loaded to Ferry1 on Port 1.
UPDATE: Car (8) is loaded to Ferry0 on Port 0.
UPDATE: Car (14) is loaded to Ferry0 on Port 0.
UPDATE: Ferry1 is moving to Port 0.
UPDATE: Ferry0 is moving to Port 1.
UPDATE: Bus (21) approaches to Booth2 on Port 0.
UPDATE: Bus (21) enters Line0 on Port 0.
UPDATE: Bus (20) approaches to Booth3 on Port 0.
UPDATE: Truck (31) approaches to Booth2 on Port 0.
UPDATE: Bus (20) enters Line0 on Port 0.
UPDATE: Truck (26) approaches to Booth1 on Port 1.
UPDATE: Truck (31) enters Line0 on Port 0.
UPDATE: Truck (26) enters Line0 on Port 1.
UPDATE: Bus (22) approaches to Booth1 on Port 0.
UPDATE: Bus (22) enters Line0 on Port 0.
UPDATE: Ferry1 arrived at Port 0.
UPDATE: Truck (24) unloaded on Port 0.
UPDATE: Bus (23) unloaded on Port 0.
UPDATE: Bus (16) unloaded on Port 0.
UPDATE: Truck (30) unloaded on Port 0.
UPDATE: Motorcycle (0) unloaded on Port 0.
UPDATE: Motorcycle (6) unloaded on Port 0.
UPDATE: Motorcycle (3) unloaded on Port 0.

UPDATE: Truck (29) unloaded on Port 0.
UPDATE: Ferry1 is fully unloaded on Port 0.
UPDATE: Ferry0 arrived at Port 1.
UPDATE: Motorcycle (1) unloaded on Port 1.
UPDATE: Car (10) unloaded on Port 1.
UPDATE: Bus (21) is loaded to Ferry1 on Port 0.
UPDATE: Bus (20) is loaded to Ferry1 on Port 0.
UPDATE: Truck (31) is loaded to Ferry1 on Port 0.
UPDATE: Bus (22) is loaded to Ferry1 on Port 0.
UPDATE: Motorcycle (7) unloaded on Port 1.
UPDATE: Car (12) unloaded on Port 1.
UPDATE: Car (13) unloaded on Port 1.
UPDATE: Bus (19) unloaded on Port 1.
UPDATE: Truck (28) unloaded on Port 1.
UPDATE: Car (11) unloaded on Port 1.
UPDATE: Motorcycle (5) unloaded on Port 1.
UPDATE: Car (8) unloaded on Port 1.
UPDATE: Car (14) unloaded on Port 1.
UPDATE: Ferry0 is fully unloaded on Port 1.
UPDATE: Ferry1 is moving to Port 1.
UPDATE: Truck (26) is loaded to Ferry0 on Port 1.
UPDATE: Ferry0 is moving to Port 0.
UPDATE: Ferry1 arrived at Port 1.
UPDATE: Bus (21) unloaded on Port 1.
UPDATE: Bus (20) unloaded on Port 1.
UPDATE: Truck (31) unloaded on Port 1.
UPDATE: Bus (22) unloaded on Port 1.
UPDATE: Ferry1 is fully unloaded on Port 1.
UPDATE: Ferry0 arrived at Port 0.
UPDATE: Truck (26) unloaded on Port 0.
INFO: Vehicle threads are done. Waiting for ferries..
UPDATE: Ferry0 is fully unloaded on Port 0.
INFO: Ferry threads are done. Testing completeness..
INFO: 32/32 checks are complete. Every vehicle has made a round trip. Success!