Middle East Technical University Department of Computer Engineering

CENG 334

Introduction to Operating Systems

Spring 2023-2024

Homework 2 - Car Simulator

Due date: 06 05 2024, Monday, 23:59

1 Objective

This assignment will familiarize you with developing multi-threaded applications and synchronization using C or C++. Your task is implementing a car simulator with road connectors (RC). Towards this end, you will be using mutexes, semaphores, and condition variables to synchronize them.

Keywords— Thread, Semaphore, Mutex, Condition Variable

2 Problem Definition

We want to simulate cars following predetermined paths containing RCs. The RCs regulate the flow of traffic based on their rules. Normal roads will enable transfer between RCs. The RCs are all connected, and cars can travel between them freely with no traffic. The travel time between connectors will be provided to you for each car. There are three types of RC: `Narrow Bridge`, `Ferry` and `Crossroads`. They are described below:

- **Narrow Bridge** is a connector that connects two opposing lanes into a single lane. It allows only cars from one side to pass at any time. This means two opposing lanes merge into the bridge, and only one way can pass at any time. The cars that are going in the opposite direction must wait. However, the waiting lane has a maximum time limit. As soon as the time limit is reached, the traffic flow reverts to the other side and the time limit begins for the other lane. If there is a car currently passing the bridge, it should finish passing before the reversal begins. The cars on the active lane pass the bridge one after the other with a slight predetermined delay and they start passing according to the order they have arrived in. You can see an example in Figure 1.

- **Ferry** is a connector that allows multiple cars to be transported from one side to the other. Ferries wait on both sides of the sea. A ferry carries all the loaded cars to the other side when the number of loaded cars reaches the capacity, or the first loaded car waits for the maximum wait time. You can see an example in Figure 2.

- **Crossroad** is a connector that connects four roads. Only cars from one road can enter the crossroad at a time. All other roads wait for no car left on the active road or a time limit is reached. As soon as the first car starts waiting for the crossroad a timer

Credit: OTAGO IMAGES/OTAGO DAILY TIMES. Creator: Sean Nugent

Figure 1: Narrow bridge example

starts and when it expires, the active road is closed and traffic is given to the first busy road available. When more than one road is busy (there are waiting cars) the traffic will be given to the next road based on the road numbering (i.e. 0,1,2,3,0,1,2,3...). After switching to a different active road, the timer is reset for other waiting roads. The cars on the active road pass the bridge one after the other with a slight predetermined delay and they start passing according to the order they have arrived in. You can see an example in Figure 3.

3 Implementation Details

You shall implement the cars as threads and synchronize their activities. The synchronization is done using RCs which is the critical part of the homework. The implementation of car threads is trivial comparatively. The number of cars, narrow bridges, ferries, and crossroads along their attributes and the paths that the cars will take will be provided to you (for details, see Input Specifications). They all have IDs starting from 0 and are assigned by their input order. The threads will be created at the beginning of the simulation. After starting threads, you will wait for them to finish by joining them and, finally exit.

3.1 Car Details

Cars have two main parameters: the travel time between connectors and the predetermined path containing road connectors and travel directions on those connectors. For each connector

Photo by Franz Wender on Unsplash

Figure 2: Ferry example

on the path, the car will travel to that connector, and try to pass the connector along the direction indicated by the path. Each connector will block or allow traffic to flow based on its rules. They will be detailed in the following subsections. The pseudo-code of car threads is given in the following algorithm:

Algorithm 1: Car thread main routine

Data: ID, TravelTime, Path
for *Connector,* Direction *in Path* **do**
  CurrentConnector $\leftarrow$ *Connector*
  CurrentConnectorID $\leftarrow$ (CurrentConnector $\rightarrow$ ID)
  CurrentConnectorType $\leftarrow$ (CurrentConnector $\rightarrow$ $Type$)
  CurrentConnectorTravelTime $\leftarrow$ (CurrentConnector $\rightarrow$ $TravelTime$)
  `WriteOutput`(*ID,* CurrentConnectorType, CurrentConnectorID, TRAVEL)
  Sleep for `TravelTime` milliseconds
  `WriteOutput`(*ID,* CurrentConnectorType, CurrentConnectorID, ARRIVE)
  `Pass` (CurrentConnector, Direction)
end

The functions are explained below:

- `WriteOutput`: The function is used for printing information for this homework. It will be provided to you.
- `Pass`: Waits on the connector until it can pass based on the connector rules and other threads waiting on that connector. After passing it will sleep to simulate passing the connector and notify the other waiting threads. Each connector type has its own rules and will be explained in detail in the following subsections.

Photo by Sasha Kaunas on Unsplash

Figure 3: Crossroad example. Shanghai

3.2 Narrow Bridge Details

Narrow bridges have two main parameters: the travel time to cross the bridge and the maximum wait time for the waiting lane. When a car tries to pass the narrow bridge, it should act based on the following algorithm:

Algorithm 2: Narrow bridge wait routine

```
Data: ID, TravelTime, MaximumWaitLimit
Car is the car associated with the calling thread
Function Pass(Connector, Direction):
    CurrentConnectorID ← (CurrentConnector → ID)
    CurrentConnectorType ← (CurrentConnector → Type)
    CurrentConnectorTravelTime ← (CurrentConnector → TravelTime)
    if Currently the passing lane is the same Direction then
        if There is car passing then
            Wait
        else if Direction has no cars left that have arrived before it then
            Sleep for PASS_DELAY milliseconds if a car has passed before
            WriteOutput(ID, CurrentConnectorType, CurrentConnectorID,
             START_PASSING)
            Notify the next waiting cat to pass
            Sleep for CurrentConnectorTravelTime milliseconds
            WriteOutput(ID, CurrentConnectorType, CurrentConnectorID,
             FINISH_PASSING)
        else
            Wait
    else if The MaximumWaitLimit is reached for the current Direction then
        Switch the passing lane to the opposite Direction
        Notify the cars on the opposite Direction
        Goto beginning
    else if There are no more cars left on the passing lane Direction then
        Switch the passing lane to the opposite Direction
        Notify the cars on the opposite Direction
        Goto beginning
    else
        Wait
```

Important points:

- After getting notified, the car thread may need to perform the same checks again.
- Maximum wait time is not triggered by a car thread. It should trigger automatically once the time limit is reached.
- `PASS_DELAY` is defined in the `helper.c` file and it is in milliseconds.

3.3 Ferry Details

Ferries have three main parameters: The travel time between the two stops of the ferry, the maximum wait time after the first car is loaded into the ferry and the capacity. The algorithm for it is given below:

Algorithm 3: Ferry wait routine

Data: ID, TravelTime, MaximumWaitLimit, Capacity
Car is the car associated with the calling thread
Function `Pass(`*Connector,* Direction`):`
    CurrentConnectorID $\leftarrow$ (CurrentConnector $\rightarrow ID$)
    CurrentConnectorType $\leftarrow$ (CurrentConnector $\rightarrow Type$)
    CurrentConnectorTravelTime $\leftarrow$ (CurrentConnector $\rightarrow TravelTime$)
    The following conditions will be done for each Direction
    Load the Car on the ferry
    if *There is no more room on the ferry* **then**
        `WriteOutput(`*ID,* CurrentConnectorType*,* CurrentConnectorID*,* START_PASSING`)`
        Notify other cars on the ferry to pass as well
        Sleep for CurrentConnectorTravelTime milliseconds
        `WriteOutput(`*ID,* CurrentConnectorType*,* CurrentConnectorID*,* FINISH_PASSING`)`
    else if *If the maximum wait time is reached* **then**
        Notify the waiting cars on the ferry to pass
    else
        Wait

Important Points:

- You can assume that loading and unloading cars from the Ferry is done instantly. Every car is unloaded at the same time.
- You can assume there are an infinite number of ferries. All cars coming to the ferry will be loaded on a new one once the current ferry departs. Cars will not wait for ferries to arrive.
- Maximum wait time is not triggered by a car thread. It should trigger automatically once the time limit is reached.

3.4 Crossroad Details

Crossroads has three main parameters: The travel time to pass the crossroad, and the maximum wait time of the other lanes. The algorithm for it is given below:

Algorithm 4: Crossroad wait routine

```
Data: ID, TravelTime, MaximumWaitLimit
Car is the car associated with the calling thread ;
Function Pass(Connector, Direction):
    CurrentConnectorID ← (CurrentConnector → ID);
    CurrentConnectorType ← (CurrentConnector → Type);
    CurrentConnectorTravelTime ← (CurrentConnector → TravelTime);
    if Currently the passing lane is the same Direction then
        if There is car passing then
            Wait;
        else if Direction has no cars left that have arrived before it then
            Sleep for PASS_DELAY milliseconds if a car has passed before;
            WriteOutput(ID, CurrentConnectorType, CurrentConnectorID,
             START_PASSING);
            Notify the next waiting cat to pass;
            Sleep for CurrentConnectorTravelTime milliseconds;
            WriteOutput(ID, CurrentConnectorType, CurrentConnectorID,
             FINISH_PASSING);
        else
            Wait;
    else if The MaximumWaitLimit is reached for the passing lane Direction then
        Switch the passing lane to the next non-empty Direction;
        Notify the cars on the next Direction;
        Go to beginning;
    else if There are no more cars left on the current Direction then
        Switch the passing lane to the next non-empty Direction;
        Notify the cars on the next Direction;
        Go to beginning;
    else
        Wait;
```

Important Points:

- After getting notified, the car thread may need to perform the same checks again.
- Maximum wait time is not triggered by a car thread. It should trigger automatically once the time limit is reached.
- Next active direction is determined by incrementing numbers. Direction details will be explained at the end of this section.

The simulation will be subjected to these constraints.

1. Directions for car path determined by two numbers representing $(from, to)$ for connectors. For narrow bridges and ferries, it can either be $(0, 1)$ or $(1, 0)$. For crossroads, it can be one of the sixteen values in the form of $([0, 3], [0, 3])$. However, when considering lanes, only the `from` value is considered. This means that crossroads have four passing lanes.
2. All cars waiting on the ferry should pass at the same time.

3. On narrow bridges and crossroads, cars in the same direction pass together with a slight delay. For a given lane, the cars should start passing in the same order that they have arrived in.

4 Implementation Specifications

1. You should call `InitWriteOutput` function before creating threads in your main thread.
2. Main thread should wait for every thread to finish before exiting.
3. Simulator should only use `WriteOutput` function to output information, and no other information should be printed.
4. You can use `sleep_milli` function in `helper.h` for sleeping in certain amount of milliseconds.
5. The slight delay, applied to the cars passing in the same direction on narrow bridges and crossroads, is given as `PASS_DELAY` in `helper.h` file.
6. `WriteOutput` takes a single character as a connector type. They should be the letter `N` for narrow bridges, `F` for ferries and `C` for crossroads.

5 Input Specifications

Information related to simulation agents will be given through standard input.

The first line contains the number of narrow bridges (N_N) in the simulation. The following N_N lines contain the properties of the narrow bridge with i^{th} ID (All IDs start from 0) in the following format:
- N_T N_M

- N_T is an integer representing the travel time of the narrow bridge in milliseconds.
- N_M is an integer representing the maximum wait time of the narrow bridge in milliseconds.

The next line contains the number of ferries (F_N) in the simulation. The following F_N lines contain the properties of the ferries with i^{th} ID in the following format:
- F_T F_M F_C where

- F_T is an integer representing the travel time of the ferry in milliseconds.
- F_M is an integer representing the maximum wait time of the ferry in milliseconds.
- F_C is an integer representing the capacity of the ferry.

The next line contains the number of crossroads (C_N) in the simulation. Following C_N lines contain the properties of the crossroads with i^{th} ID in the following format:
- C_T C_M

- C_T is an integer representing the travel time of the crossroad in milliseconds.
- C_M is an integer representing the maximum wait time of the crossroad in milliseconds.

The next line contains the number of cars (CA_N) in the simulation. Following $CA_N \times 2$ lines contain the properties of the cars with i^{th} ID in the following (two line) format:
- CA_T CA_P
- P_{C1} F_{C1} T_{C1} ... P_{CN} F_{CN} T_{CN}

- CA_T is an integer representing the travel time of the car between connectors in milliseconds.
- CA_P is an integer representing the path length. Then the next line contains CA_P number of path objects in the following format:
  - P_{CM} is the string representing the M^{th} connector on the path with the format: `{N or F or C for connector type}{ConnectorID}`. Ex:
    `N2` → Narrow bridge with the ID 2
  - F_{CM} is the from direction. For ferry and narrow bridges, it can be either one or zero. For crossroads can be valued in $[0, 3]$.
  - T_{CM} is the to direction. For ferry and narrow bridges, it can be either one or zero. For crossroads can be valued in $[0, 3]$. From and to directions can be the same.

Example:

```
1
10 5
1
10 10 5
1
10 10
4
10 3
N0 0 1 F0 1 0 C0 2 3
8 2
N0 1 0 F0 0 1
9 3
N0 1 0 F0 1 0 C0 1 0
5 4
N0 0 1 F0 0 1 C0 1 3 N0 1 0
```

NOTE: There will only be a single space separation between values.

6 Specifications

- Your code must be written in C or C++ on Linux. No other platforms and languages will be accepted.
- You can only use pthread libraries for the threads, semaphores, condition variables, mutexes and monitors (Onur Hoca's implementation). I have also added the `timedwait` function to monitor implementation and will provide you with the updated file. Your Makefile should not contain any library flag other than -lpthread. It will be separately controlled.
- You are **NOT** allowed to use C++ `threads` library.
- Your solution should not employ busy wait.
- Submissions will be evaluated with black box technique with different inputs. Consequently, output order and format are important. Please make sure that calls to the `WriteOutput` function are done in the correct step. Also, please do not modify `writeOutput.h`, `writeOutput.h`, and `helper.h` files as your submission for these files may be overwritten. You do not even have to include them in your submissions. However, make sure to include `monitor.h` file as you are free to modify it.
- There will be a penalty for bad solutions. Non-terminating (deadlocked or otherwise) simulations will get zero from the corresponding input.
- Your submission will be evaluated on lab computers (ineks).
- Everything you submit should be your work. Usage of source files and codes found on the internet is strictly forbidden.
- Please follow the course page on Cow and ODTUClass for updates and clarifications.
- Please ask your questions through ODTUClass instead of emailing teaching assistants if your question does not contain a solution or code.

7 Submission

Submission will be done via ODTUClass. Create a tar.gz file named `hw2.tar.gz` that contains all your source code files with your Makefile. Your tar file should not contain any folders. Your code should be able to compile and your executable should run using this command sequence.

```
> tar -xf hw2.tar.gz
> make all
> ./simulator
```

8 Grading

The tentative grading rubric is given below:

- Testcases containing only narrow bridges: 10 pts
- Testcases containing only ferries: 10 pts
- Testcases containing only crossroads: 10 pts
- Testcases containing narrow bridges and ferries: 15 pts
- Testcases containing narrow bridges and crossroads: 15 pts
- Testcases containing ferries and crossroads: 15 pts
- Testcases containing all connectors: 25 pts