CENG334 - Introduction to Operating Systems

Spring 14/15

Project Assignment (Section 1)

Synchronization – Railway Simulation

Özcan Dülger – odulger(at)ceng.metu.edu.tr

Due: 03.05.2015 - 23:55:00

1. Objectives

In this assignment you are going to practice on synchronization between threads using C++. Your task is to implement a railway simulation system to synchronize the trains on the opposite positions. You are allowed to use semaphores, condition variables and mutexes for the synchronization between threads. You can use one of them or all of them.

Keywords: *Railway Simulation, Thread, Semaphore, Mutex, Condition Variable*

2. Problem Definition

In railway simulation, there is a single railway connection between two cites. The trains in the railway move in west direction or east direction but not both. When a train moves in the railway, the trains on the opposite direction can not enter the railway and wait for the railway becomes empty. But the trains in the same direction can enter and follow each other. The order of the trains is important. The trains in the same direction can enter the railway in any order but the trains should exit from the railway as in their entering order to the railway. When the railway is empty, the trains in any direction can enter. There is a limit in the railway to avoid the trains in opposite direction waiting the railway forever. When the first train in opposite direction starts to wait the railway, number of trains on the railway plus trains may follow them in current direction later is limited by this number. After this limit is reached, no other train will be allowed in current direction, and when the railway is empty a train in opposite direction will enter the railway.

The following is a summary of checks for entering and exiting a rail for a train:

```
for enter(Direction):
    1. if railway is empty → enter
    2. if train(s) in opposite direction →
              activate limit counter as #of trains on railway
              wait until railway is empty
    3. if train(s) in same direction →
        1. if no train in opposite direction is waiting → enter
        2. else if limit-counter >= limit →
              wait until opposite direction train(s) enter
              and railway is empty back again
        3. else → increment limit-counter and enter

for exit():
    1. if not following a train → exit
    2. else wait for train to exit, then exit
```

3. Tasks

Your task is to write a simple railway simulation. In this simulation, the group of train threads on the railway will be synchronized. You are allowed to use only pthread.h library and semaphore.h library for the threads, semaphores, condition variables and mutexes. Your solution should not busy wait. To implement the solution, you have to complete the following implementations as well:

- Create a thread for each train and synchronize the trains with a proper way. The grades of bad solutions will be cut off. Create the threads in detached mode in order to release their resources without waiting all the threads to finish their job.
- Read the railway limit data from the command line as. /project limit.
- Read the information of the trains from stdin (standard input) each on a separate line. Its content is explained in section 4 in detail.
- Write the output of each operation to the stdout (standard output). A function namely writeOutput will be given in writeOutput.h file to write the outputs of the simulation. Do not modify this file. Read the comment lines in the function carefully and set the parameters for the appropriate output.
- The content template for project.cpp file will be provided. There is a Rail class which consists of data members and procedures for the railway operations. The trains will call Enter function before entering the railway. The trains will call Exit function before they exit the railway. The limit data member is used for the limit property of the railway. You will assign the value of the limit in the constructor of the Rail class. Do not modify the definitions of public functions and the private limit data member of the Rail class. But you can add any function or data members as a private member to the Rail class.
- Arriving operation of the trains on the railway will take (arriveTime*1000) microseconds. Moving operation on the railway will take (duration*10000) microseconds. Use usleep function that is defined in unistd.h library. It gets its input as a microsecond value.
- Read and use the underlined information carefully; otherwise your evaluation may end up with unexpected results.
- Pseuodo-code for simulation can be given as:

```
train thread:
writeOutput(0, trainCode, direction)
Enter(direction, trainCode)
usleep(duration*10000)
Exit(direction, trainCode)
finish thread

main thread:
read railway data
while (read train request r)
{
        usleep(arriveTime*1000)
        start a train thread for r
}
end of request input
wait until last train finishes
exit
```

4. Inputs

The limit parameter of the railway will be given in the command line as. /project *limit*.
You will read the train information from stdin (standard input) as a single line. An example train input is as follows:

```
0 1 WEST 1
0 2 EAST 1
1 3 WEST 2
1 4 WEST 1
3 5 WEST 3
1 6 EAST 1
1 7 EAST 1
2 8 WEST 1
2 9 WEST 2
1 10 WEST 3
1 11 EAST 4
2 12 EAST 1
4 13 WEST 1
5 14 WEST 2
1 15 WEST 1
2 16 EAST 1
```

The parameters are separated with a single space character. The first parameter is the relative arrival time (arriveTime) of the train. The second parameter is the train code of the train. This is a unique integer value. The third parameter is the direction of the train: either WEST or EAST. The fourth parameter is the duration of the train on the railway.

5. Outputs

Write the output of three operations: arrive, enter and exit to the stdout (standard output). A function namely writeOutput will be given in writeOutput.h file to write the outputs of the simulation. Read the comment lines in the function carefully and set the parameters for the appropriate output. Output order is important, so be sure that you call the writeOutput function in the correct step with the appropriate parameters. Do not modify writeOutput.h file. An example output of the input that is given section 4 is as follows (The value of the limit is 5):

```
Train = 1 arrived the railway towards the west
Train = 1 entered the railway towards the west
Train = 2 arrived the railway towards the east
Train = 3 arrived the railway towards the west
Train = 3 entered the railway towards the west
Train = 4 arrived the railway towards the west
Train = 4 entered the railway towards the west
Train = 5 arrived the railway towards the west
Train = 5 entered the railway towards the west
Train = 6 arrived the railway towards the east
Train = 7 arrived the railway towards the east
Train = 8 arrived the railway towards the west
Train = 8 entered the railway towards the west
Train = 1 exited the railway towards the west
Train = 9 arrived the railway towards the west
Train = 10 arrived the railway towards the west
Train = 11 arrived the railway towards the east
Train = 12 arrived the railway towards the east
Train = 13 arrived the railway towards the west
Train = 3 exited the railway towards the west
Train = 4 exited the railway towards the west
Train = 14 arrived the railway towards the west
Train = 15 arrived the railway towards the west
Train = 5 exited the railway towards the west
Train = 16 arrived the railway towards the east
Train = 8 exited the railway towards the west
Train = 2 entered the railway towards the east
Train = 6 entered the railway towards the east
Train = 7 entered the railway towards the east
Train = 11 entered the railway towards the east
Train = 12 entered the railway towards the east
Train = 2 exited the railway towards the east
Train = 6 exited the railway towards the east
Train = 7 exited the railway towards the east
Train = 11 exited the railway towards the east
Train = 12 exited the railway towards the east
Train = 9 entered the railway towards the west
Train = 13 entered the railway towards the west
Train = 10 entered the railway towards the west
Train = 15 entered the railway towards the west
Train = 14 entered the railway towards the west
Train = 9 exited the railway towards the west
Train = 13 exited the railway towards the west
Train = 10 exited the railway towards the west
Train = 15 exited the railway towards the west
Train = 14 exited the railway towards the west
Train = 16 entered the railway towards the east
Train = 16 exited the railway towards the east
```

6. Specifications

- Please read the specifications carefully!
- Your homework must be written in standard C++. No other platforms are accepted.
- You are allowed to use only pthread.h library and semaphore.h library for the threads, semaphores, condition variables and mutexes. Your solution should not busy wait.
- Your solutions will be evaluated as a black box technique by using differently many inputs. Output order is important, so be sure that you call writeOutput function in the correct step with the appropriate parameters.
- The non-terminated solutions will get zero from the corresponding input.
- Partial grades will be given as:
    1. Only trains in exclusive direction is implemented → Max 40 pts
    2. In addition, trains following entrance order on exit implemented → Max 75 pts
    3. In addition to 1 or 2, waiting limit is implemented → + 25 pts.
- Everything you submit should be your own work.
- Please follow the course page on newsgroup (cow) for any update and clarification.

- Please ask your questions related to the homework on cow instead of email in order to your friends, who may face with same problem, can see your questions and answers.
- Your programs will be compiled with g++ and run on the department inek machines. No other platforms/g++ versions etc. will be accepted so check that your code works on ineks before submitting it.
- Sharing and copying any piece of code from each other and INTERNET is strictly forbidden. Both the sharing one and the copying one will be counted as cheated.
- Use sufficient comment lines in your algorithm in order to explain your solution clearly.

7. Submission

Submission will be done via COW. Create a tar.gz file named project.tar.gz that contains project.cpp and a makefile. Makefile should create an executable named project. Your implementations will be compiled using the command chain:

```
$ tar -xf project.tar.gz
$ make
```

The tar file should not contain any directories!!

May it be easy.