CENG334 - Introduction to Operating Systems

Spring 13/14

Programming Assignment 2 (Section 2)

Synchronization – Airport Simulation

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

Due To: 05.05.2014 - 23:55:50

1. Objectives

In this assignment you are going to practice on synchronization between threads using C or C++. Your task is to implement an airport simulation system to synchronize the ground movements of planes which may land or take off. 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: *Airport Simulation, Thread, Semaphore, Mutex, Condition Variable*

2. Problem Definition

In airport simulation, there are planes that may land or take off using a runway in the airport. The runways are used for both landing and take-off. There are also park areas for the planes to stay in and taxi nodes for the connection between park areas and runways. The specifications about the airport simulation are given below:

- There are greater than or equal to one runways in the airport. Both landing and take-off can be operated on each runway. Only one plane can operate on a runway at a time.
- Airport has park areas for the planes to stay in. There are taxi nodes between the runways and the park areas. The pathway between the taxi nodes are in one-way direction: either runway to park direction or park to runway direction.
- The taxi nodes in runway to park direction do not intersect or connect with the taxi nodes in park to runway direction. And there is no circular connection between the taxi nodes.
- The taxying operation between the runways and the park areas is done by passing through the taxi nodes in the path. There can be only one plane on a taxi node at a time.
- At the beginning, a plane can request landing or taking off.
- The operations of the landing for a plane start with landing on a runway and finish when it reaches to a park area. The operations of the taking off for a plane start with the taxying to the runway and finish when it takes off from a runway.
- Airport has a plane capacity which is equal to the maximum number of the planes that are allowed in the ground (runways and taxi nodes) at an instance of time. If the capacity is full, the planes cannot make request for landing or taking off. If a plane finishes its operations, the free capacity will be increased.

- When a plane finishes its landing or its take-off on a runway, that runway will become available immediately.
- The current taxi node will become available when the plane enters to the next taxi node in the path.
- There are some exceptional taxi nodes that will be considered. These are listed as follows:
    - ♦ After the plane finishes landing on a runway, it will start its taxying by entering to the first taxi node. Since there can be only one plane on a taxi node at a time, there may be more than one landed planes which wait for the first taxi node. Do not consider this implicit queue in your solution.
    - ♦ When the plane leaves the last taxi node of the path in runway to park direction, the taxying to the corresponding park area will finish. And this last taxi node will become available immediately. There may be more than one plane stay in the same park area. Contents of the park areas are irrelevant in this homework.
    - ♦ When the plane leaves the last taxi node of the path in park to runway direction, the taxying to the corresponding runway will finish. And this last taxi node will become available immediately. There may be more than one plane which wait for the runway to become available for taking off. Do not consider this implicit queue in your solution.
- An example illustration of the airport is given in Figure1. The circles represent the taxi nodes, the string that starts with "P" represents the parks and the string that starts with "R" represents the runways.

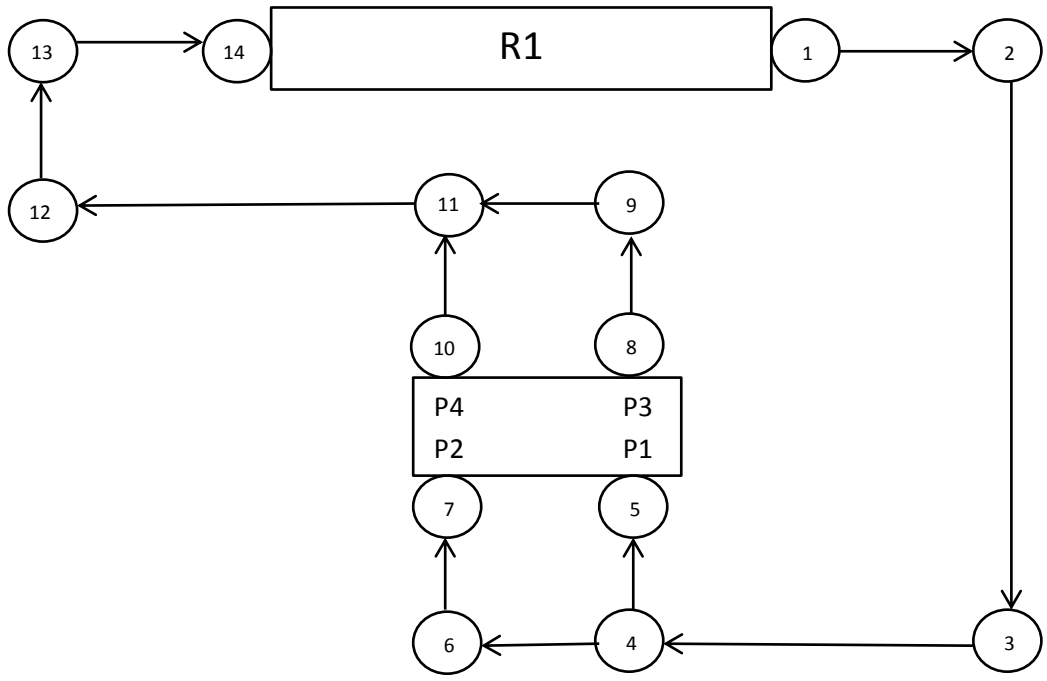

Figure 1: An illustration of the airport

In this airport, for example, if a plane requests landing on R1 and parking on P2, it should finish its landing on R1 and enter to the taxi node 1, 2, 3, 4, 6 and 7. When it leaves the taxi node 7, the taxying operation to the park P2 will finish and this taxi node will become available immediately. If a

plane requests taking off from R1 and stays in P3, it should enter to taxi node 8, 9, 11, 12, 13 and 14. When the plane leaves the taxi node 14, the plane will start to wait until the R1 becomes available to take off and this taxi node will become available immediately. After finishing the take-off from R1, the operations for the plane will finish.

3. Tasks

Your task is to write a simple airport simulation. In this simulation, the group of plane threads on the airport will be synchronized. You are allowed to use only pthread.h library and semaphore.h library for the threads, semaphores, condition variables and mutexes. To implement the solution, you have to complete the following implementations as well:

- Create a thread for each plane and synchronize the planes 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 airport data from a text file. Filename is given from the command line as ./hw2 airportData.txt . Its content is explained in section 4 in detail.
- Read the landing/take-off requests of planes 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 writeToFile will be given in homework files to write the outputs of the simulation. Read the comment lines in the function carefully and set the parameters for the appropriate output.
- Landing operation on a runway will take 10000 microsecond, take-off operation from a runway will take 50000 microsecond and taxying operation waits on a single taxi node will take 10000 micro second. 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:

```
plane thread:
if (landing) {
        start landing
        start landing on runway
        sleep 10000 microseconds
        finish landing on runway
        start taxying
        for each taxi node t connecting runway to park: {
                enter node t
                sleep 10000 microseconds
        }
        finish landing
} else {
        start take-off
        start taxying
        for each taxi node t connecting park to runway: {
```

```
                enter node t
                sleep 10000 microseconds
        }
        start take off from runway
        sleep 50000 microseconds
        finish take off from runway
        finish take off
}
finish thread
```

main thread:

```
read airport data
while (read landing/take off request r) {
        start a plane thread for r
}
end of request input
wait until last plane finishes
exit
```

4. Inputs

The input parameters of the airport will be read from a text file. Filename is given from the command line as ./hw2 airportData.txt. As an example, the input parameters of the airport simulation that is given in Figure 1 are as follows:

```
1
4
14
2
R1P1 1 2 3 4 5 -1
R1P2 1 2 3 4 6 7 -1
P3R1 8 9 11 12 13 14 -1
P4R1 10 11 12 13 14 -1
-1
```

You will read the parameters until you reach to the line that starts with -1. The first parameter represents the number of runways in the airport. The runway ids will start with 'R' character and will be in order. Assume that the airport has three runways; they are represented as R1, R2 and R3. The second parameter represents the total park number in the airport. The park ids will start with 'P' character and will be in order. Assume that the airport has four parks; they are represented as P1, P2, P3 and P4. The third parameter represents the total taxi nodes in the airport. The taxi nodes will be integer value and will be in order. Assume that the airport has five taxi nodes; they are represented as 1,2,3,4 and 5. The fourth parameter represents the plane capacity of the airport.

In the fifth line, the taxi node connection matrixes between the runways and the parks are started. First, runway to park connection matrix is given. The first value in a line gives information about row id and column id. For example, R1P1 means that the row id is R1 and the column id is P1. The cells of the matrix are given in row major. The values of the cells are list of taxi nodes in a pathway from the

corresponding runway to the corresponding park. The taxi node information begins just after the first single space character up to the -1. You will parse the line to get the taxi node information. Taxi node ids are separated by a single space. The park to runway connection matrix starts with the first line that starts with the 'P' character. In this matrix, the row represents the park areas and the columns represent the runways. The cells of the matrix are given in row major. The values of the cells are taxi node information from the corresponding park to the corresponding runway. The parsing operation of this matrix is same with the parsing operation of the runway to park connection matrix. The runways and park areas are fully connected in both directions. Assume you will always be given the complete input.

You will read the plane information from stdin (standard input) as a single line. The plane information comes as a string and you will parse them to get the information. An example plane input is as follows:

12354 L 1 2
18765 T 1 3
17564 T 1 3
21654 L 1 1
16785 T 1 4

The parameters are separated with a single space character. The first parameter is the flight code of the plane. This is a unique integer value. The second parameter is the flight type of the plane: either L (landing) or T (take-off). The third parameter is the runway id where the plane will land or take off. The fourth parameter is the park id where the plane taxying after landing or stay in at the beginning. In the example, 12354 L 1 2 means the plane with 12354 flight code will land on runway 1 and taxy to park 2 to stay in. 18765 T 1 3 means the plane with 18765 flight code which is stay in park 3 will take off from runway 1.

5. Outputs

Write the output of each operation to the stdout (standard output). A function namely writeToFile will be given in homework files 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 writeToFile function in the correct step with the appropriate parameters. An example output of the input that is given section 4 is as follows:

Plane = 12354 started landing
Plane = 12354 started its landing on runway 1
Plane = 18765 started take-off
Plane = 18765 entered to taxi node 8
Plane = 18765 entered to taxi node 9
Plane = 12354 finished its landing on runway 1
Plane = 12354 entered to taxi node 1
Plane = 18765 entered to taxi node 11
Plane = 12354 entered to taxi node 2
Plane = 18765 entered to taxi node 12
Plane = 12354 entered to taxi node 3
Plane = 18765 entered to taxi node 13
Plane = 12354 entered to taxi node 4

Plane = 18765 entered to taxi node 14
Plane = 12354 entered to taxi node 6
Plane = 18765 finished its taxying to the runway 1
Plane = 18765 started its take off from runway 1
Plane = 12354 entered to taxi node 7
Plane = 12354 finished its taxying to the park 2
Plane = 17564 started take-off
Plane = 17564 entered to taxi node 8
Plane = 17564 entered to taxi node 9
Plane = 17564 entered to taxi node 11
Plane = 17564 entered to taxi node 12
Plane = 18765 finished its take off from runway 1
Plane = 21654 started landing
Plane = 21654 started its landing on runway 1
Plane = 17564 entered to taxi node 13
Plane = 21654 finished its landing on runway 1
Plane = 21654 entered to taxi node 1
Plane = 17564 entered to taxi node 14
Plane = 21654 entered to taxi node 2
Plane = 17564 finished its taxying to the runway 1
Plane = 17564 started its take off from runway 1
Plane = 21654 entered to taxi node 3
Plane = 21654 entered to taxi node 4
Plane = 21654 entered to taxi node 5
Plane = 21654 finished its taxying to the park 1
Plane = 16785 started take-off
Plane = 16785 entered to taxi node 10
Plane = 16785 entered to taxi node 11
Plane = 17564 finished its take off from runway 1
Plane = 16785 entered to taxi node 12
Plane = 16785 entered to taxi node 13
Plane = 16785 entered to taxi node 14
Plane = 16785 finished its taxying to the runway 1
Plane = 16785 started its take off from runway 1
Plane = 16785 finished its take off from runway 1

6. Specifications

- Please read the specifications carefully!
- Your homework must be written in C or 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 solutions will be evaluated as a black box technique by using differently many inputs. Output order is important, so be sure that you call writeToFile function in the correct step with the appropriate parameters.
- The grades of the bad solutions will cut off %50. The non-terminated solutions will get zero from the corresponding input.
- 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 gcc/g++ and run on the department inek machines. No other platforms/gcc/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 hw2.tar.gz that contains hw2.cpp or hw2.c and a makefile. Makefile should create an executable named hw2. Your implementations will be compiled using the command chain:

```
$ tar -xf hw2.tar.gz
$ make
```

The tar file should not contain any directories!!

May it be easy.