Assignment 1: Processes and Threads

Due Date: **9th September 2014**

Weighting: **20%**

Group or Individual: **Individual/Group of Two**

1 Tasks

There are two tasks in the assignment. INB365 students need to attempt Task 1 only; INN365 students need to attempt both Task 1 and Task 2.

1.1 Task 1 (for both INB365 students and INN365 students)

Your task is to develop a simulator for an airport, which has only one runway, on which one plane can land or one plane can take off at a time. A plane cannot land at the same time another plane is taking off. Planes arrive at the airport or take off from the airport at random times. When a plane lands it parks in a randomly allocated landing bay that is not currently occupied (and not just in the next free available space). Of course only one plane can occupy a single landing bay at a time. The capacity of the airport is 10, which means it can accommodate up to 10 landed planes.

The simulator should be implemented in the C programming language using POSIX Threads, which is usually referred to as Pthreads, on the Linux operating system. This is the same environment used for the weekly practicals. The simulator is to run in a command line window, reading input from the keyboard and writing output to the screen.

The airport simulation program will have four threads:

1. The main thread that spawns the following three threads;
2. A landing thread (producer) that simulates planes landing at the airport;
3. A taking-off thread (consumer) that simulates planes taking off from the airport;
4. A monitor thread that accepts input from the keyboard.

The main thread starts the airport simulation program and displays the initial banner (see the example output below). Once the simulation begins it creates the other three threads as listed above and then waits for them to terminate.

The landing thread randomly generates arriving planes which need to land at the airport. If the airport is full, the landing thread prints “The airport is full” and will block until the airport is not full. The landing thread does not continually generate arriving planes. Instead, it sleeps for half a second (0.5 seconds) before it generates the next arriving plane. Planes are identified by a randomly generated six-character code such as “QF1234” consisting of two upper case letters and four digits. When a plane, let’s say QF1234, starts to land, the landing thread will display “Plane QF1234 is landing …” and after the airplane has landed, the landing thread will display “Plane QF1234 parked in landing bay 8”. The duration of landing for all planes is two seconds (the time taken for a plane to land once it has announced it is landing).

The taking-off thread randomly picks a landed plane in the airport to take off. If the airport is empty, then the taking-off thread prints “The airport is empty” and will block until the airport is not empty. The taking-off thread does not continually pick up a plane to take off. Instead, it sleeps for half a second (0.5 seconds) before it picks up the next plane in the airport to take off. When a plane, let’s say QF1234, has been selected to take off, the taking-off thread will display the taking off state and the information about the plane. For example, “After staying at bay 5 for 4.50 seconds, plane QA1234 is taking off …”, and after the taking off is complete, the taking-off thread displays “Plane QA1234 has finished taking off”. The duration of taking off for all planes is two seconds (the time required for a plane to take off).

The monitor thread reads input from the keyboard. Valid inputs include 'p', 'P', 'q' and 'Q'. The input 'p' or ‘P’ indicates that the current state of the airport needs to be printed to the screen. When printing the airport state, each and every landed airplane in the airport must be listed with the information about the plane as well as the empty landing bays. For example if plane QF1234 landed 1.5 seconds ago, then display “2: QF1234 (has parked for 1.50 seconds)” where “2:” represents the landing bay occupied by the plane (see the example output below). Remember the state of each and every landed bay must be written to the console even if the landing bay is empty as displayed in the example below. The input 'q' or ‘Q’ indicates that the simulation is to be terminated. However, before the airport simulation terminates, it must display the current state of the airport (as shown in the example output below) listing every plane currently occupying the airport as well as the empty landing bays. If a landing bay is vacant then the simply write to the console as an example “3: Empty” where “3” represents the landing bay. Note that the simulation must end gracefully without calling exit() or killing the threads.

The chance of the landing thread generating a plane must be passed at the command line as must the chance of the taking-off thread randomly selecting a landed plane to take off. This means you must pass three (3) parameters at the command line. The first parameter should be program name, the second parameter is the chance that the landing thread will produce a plane and the third parameter should be the chance that a randomly selected plane takes off. The format of the command line to run your program should be:

<***program name***> <***probability of landing thread producing a plane***> <***probability of randomly selected landed plane taking off***>

e.g. “***./AirportSimulator 30 50***”.

This means there is a 30% chance the landing thread will produce a plane to land and a 50% chance that a randomly selected plane will take off. Your program should not allow a probability of less than 1% or greater than 90% to be passed at the command line (i.e. Not less than 1% or greater than 90% for parameter 2, not less than 1% or greater than 90% for parameter 3).

You may choose to use any data structure to represent the airport such as an array or a linked list. Concurrent access to this shared data structure must be synchronised using POSIX threads synchronisation primitives, such as a mutex and semaphores.

Example Output:

```
./AirportSim 50 50 --- input from the user

Welcome to the airport simulator.
Press p or P followed by return to display the state of the airport.
Press q or Q followed by return to terminate the simulation.

Press return to start the simulation.
Plane YQ7510 is landing ...
Plane YQ7510 parked in landing bay 3.
After staying at bay 3 for 0.00 seconds, plane YQ7510 is taking off ...
Plane YQ7510 has finished taking off.
The airport is empty.

Plane KA5437 is landing ...
Plane KA5437 parked in landing bay 0.
Plane ZB4266 is landing ...
Plane ZB4266 parked in landing bay 1.
After staying at bay 0 for 4.00 seconds, plane KA5437 is taking off ...
Plane KA5437 has finished taking off.
Plane ZD8160 is landing ...
Plane ZD8160 parked in landing bay 7.

P<enter>  --- input from the user
Airport state:
0: Empty
1: ZB4266 (has parked for 6.01 seconds)
2: Empty
3: Empty
4: Empty
5: Empty
6: Empty
7: ZD8160 (has parked for 2.00 seconds)
8: Empty
9: Empty

After staying at bay 1 for 7.50 seconds, plane ZB4266 is taking off ...
Plane ZB4266 has finished taking off.
Plane WR4863 is landing ...
Plane WR4863 parked in landing bay 8.
After staying at bay 7 for 8.01 seconds, plane ZD8160 is taking off ...
Plane ZD8160 has finished taking off.
Plane AG2430 is landing ...
Plane AG2430 parked in landing bay 9.
After staying at bay 8 for 8.51 seconds, plane WR4863 is taking off ...
Plane WR4863 has finished taking off.
Plane VY1381 is landing ...
Plane VY1381 parked in landing bay 5.
After staying at bay 9 for 8.01 seconds, plane AG2430 is taking off ...
Plane AG2430 has finished taking off.

p<enter>  --- input from the user
Airport state:
0: Empty
1: Empty
2: Empty
3: Empty
```

```
4: Empty
5: VY1381 (has parked for 4.00 seconds)
6: Empty
7: Empty
8: Empty
9: Empty

Plane QY2681 is landing ...
Plane QY2681 parked in landing bay 2.
After staying at bay 2 for 8.00 seconds, plane QY2681 is taking off ...
Plane QY2681 has finished taking off.
Plane UJ0334 is landing ...
Plane UJ0334 parked in landing bay 8.
After staying at bay 8 for 4.00 seconds, plane UJ0334 is taking off ...
Plane UJ0334 has finished taking off.
Plane EE5378 is landing ...
Plane EE5378 parked in landing bay 1.
After staying at bay 1 for 4.00 seconds, plane EE5378 is taking off ...
Plane EE5378 has finished taking off.

Q<enter>  --- input from the user
Airport state:
0: Empty
1: Empty
2: Empty
3: Empty
4: Empty
5: VY1381 (has parked for 16.51 seconds)
6: Empty
7: Empty
8: Empty
9: Empty
```

1.2 Task 2 (for INN365 students only)

Discuss how to extend the airport simulation program that you developed for Task 1 to simulate an airport which has two runways, on each of which one plane can land or one plane can take off at a time. But, one plane can take off or land on one runway while another plane taking off or landing on the other runway.

You do not need to actually develop another airport simulator. All you need to do is to discuss what new issues you would need to consider when extending your airport simulation program and how you would address these new issues.

2 Requirements

- Your solution to Task 1 must be written in C programming language using POSIX Threads on the Linux operating system. This is the same environment used in the weekly practicals.
- Your airport simulation program must be a console/command-line application, reading input from the keyboard and output to the screen.
- The producer/consumer pattern must be used in your airport simulation program.
- Concurrent access to any shared data in a critical section must be synchronised using POSIX threads synchronisation primitives.
- Your program *must* be able to compile at the command line of the Linux Operating System

3 Submission

Your assignment solution will be submitted electronically via Blackboard before 11:59pm on Tuesday, 9th September 2014. Your submission should consist of the following:

A Zip file including:

- All source code of your solution.
- **README.txt** file with instructions on how to compile and run your program.
- The ***make*** file that you used for generating the executable file, if any.
- **A report in Word Document format which includes:**
    a. A statement of completeness indicating the tasks you attempted, any deviations from the assignment specification, and problems and deficiencies in your solution.
    b. A description of the data structures in your implementation, as well as a description of the threads and how they interact with each other.
    c. Your answer to Task 2 (**for INN365 students only**).