COMP 304 - Operating Systems: Project 2
Air Traffic Control with Pthreads

Due: 23:59 June 5th, 2024

Hakan Ayral

Notes: The project can be done individually or as a team of 2. You may discuss the problems with other teams and post questions to the discussion forum, but the submitted work must be your own. **Any material you use from external sources such as the internet should be properly cited in your report**. This assignment counts for both Project 2 and Project 3 from the syllabus, therefore it is worth 25% of your total grade. (Project 1 was worth 15%)

Corresponding TAs: GÖRKAY AYDEMIR, ILYAS TURIMBETOV, MOHAMMAD KEFAH TAHA ISSA

Github Classroom link: https://classroom.github.com/a/JrSWID3Z

Description

In this project, you will get more familiar with the concepts of scheduling, synchronisation, multi-threading and deadlock prevention in operating systems by using POSIX threads (pthreads) API.

Air Traffic Controller

Air traffic control (ATC) is a service to prevent collisions of planes during take-off or landing. ATC provides information for pilots during landing or take-off. In this project, you will implement a simulator that simulates multiple independently acting planes in air traffic by modelling their take-off and landing, where the air traffic is controlled by the ATC tower. Your job is to implement the collision and deadlock-free simulator.

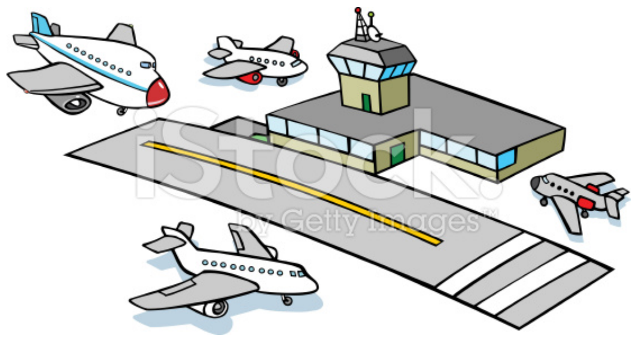

Figure 1: Air traffic directed by the control tower

Part I

(50 points) Here is how planes use the runway. When a plane needs to land or depart, it will contact with the tower and notify the tower that it is ready to land/depart. The tower will acknowledge the plane about its position (e.g. 3rd place to land). The tower contacts (signals/wakes up) the plane that is in the first position to land/depart and asks it to get ready to land/depart. Here are detailed rules for the simulation environment.

- The simulation should use real-time. Get the current time of the day, run the simulation until current time + simulation time.
- There is a single runway that is used both for take-off and landing.
- There is only one plane on the runway at any time, otherwise collision is assumed to occur.
- After a plane takes off or lands, it can head to any direction and has no consequence on the simulation.
- Each plane takes $2t$ secs to land or take-off (sleep the thread for $2t$ secs).
- Arriving or departing planes have to notify the tower and ask for permission to use the runway.
- A plane arrives to the airport with probability p at every t secs and notifies the tower that it is ready to land.
- A plane becomes ready to take off with probability $1\text{-}p$ at every t secs and notifies the tower that it is ready to take-off.
- It is not allowed for a plane to land without the acknowledgement of the tower (a plane cannot acquire a lock from another plane to land/depart).
- For fuel efficiency, the control tower favors the landing planes and lets the waiting planes in the air to land until no more planes is waiting to land. The tower will let only one plane to take off and start favouring the landing planes again.
- At time zero, there is one plane waiting to take off and one plane waiting to land.

- Use a command line argument *-s* to indicate the total simulation time (e.g. -s 200) and *-p* for probability.

In real life, t is usually in the order of minutes but assume t is 1 sec for the purpose of the computer simulation.

Part II

(35 points) Part I may cause starvation to the planes waiting on the ground. In this section, we will add a maximum wait-time and counter for the planes on the ground to avoid this situation. The control tower favors the landing planes and lets the waiting planes in the air to land until one of following conditions hold

- (a) No more planes is waiting to land,
- (b) 5 planes or more on the ground are lined up to take off.

Note that this solution now causes starvation to the planes in the air. Suggest and implement a solution to avoid starvation of waiting planes in the air. Briefly explain why starvation for landing planes occurs and how you solve it in your report.

Keeping Logs

(15 points)
You are required to keep a log for the planes and for the tower, which will help you debug and test your code. The planes.log should keep the plane no (ID), its status (landing (L) or departing (D) or emergency (E)), the request time to use the runway, the runway time (end of runway usage), turnaround time (runway time - request time). Use even IDs to indicate landing, odd IDs for the departing planes.

PlaneID	Status	Request Time	Runway Time	Turnaround Time
1	D	0	2	2
2	L	0	4	4

Output the snapshot of the waiting planes on the ground and in the air with their IDs in every second starting from n^{th} secs to the terminal, where n is also a command line argument. The numbers indicates the plane IDs waiting in the queue at time n on the ground and in the air. Feel free to come up with a better representation or GUI.

```
At 20 sec ground: 16, 18, 20
At 20 sec air   : 23, 25
```

Implementation

- You may want to keep a queue for landing planes and another queue for departing planes, add planes to queues at the appropriate times. C++ STL queues may help your implementation of the data structure.
- You can use random number generator to generate planes at the specified probability. For easy debugging, add a command line argument for a seed and feed the seed to the random number generator. (helps with debugging).
- **You should represent each plane as a thread. The air control tower should have its own separate thread.**
- Start simple. For example first simulate landing planes only. Add complexity as you are sure the current implementation works (no deadlock, no accident).
- **To sleep pthreads, please use the code that we provided on blackboard. Do not use sleep() system call**.
- For Pthread semaphores, mutexes and condition variables, also refer to the pthread tutorial online: **https://hpc-tutorials.llnl.gov/posix/**

Deliverables

You are required to submit the followings packed in a zip file (named your-username(s).zip) to blackboard :

- .c or .cpp source file that implements the simulation. Please comment your implementation.
- sample log files for 60 sec simulation for p=0.5.
- any supplementary files for your implementation (e.g. Makefile)
- a README (report) file describing your implementation, particularly which parts of your code work. Since we cannot hold a demo with everyone, document your report properly.
- Finally you may perform a demo if requested by the TA if there are issues with your implementation.

GOOD LUCK.