COMP 304: Project 2
Spacecraft Control with Pthreads

Due: April 30th 2020, 11.59 pm

Notes: The project can be done **individually or in teams of 2**. You may discuss the problems with other teams and post questions to the OS discussion forum but the submitted work must be your own work. This assignment is worth 15% of your total grade.

Contact TA: Ilyas Turimbetov (ENG 230)
mailto:iturimbetov18@ku.edu.triturimbetov18@ku.edu.tr

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.

Spacecraft Controller

You have been hired as an engineer to a company that focuses on reusable spacecraft named SpaceY. Your job is to implement a collision and deadlock-free simulator of a spacecraft launch site. The launch site has 2 landing pads: pad A and pad B, where pieces of the launched rockets land. Since the spacecraft is reusable, these pieces can be reassembled. The space center also has a controller tower, that manages and schedules all the landing, launching and reassembly of spacecraft. Launching can be done from pad A, while reassembly of the spacecraft is done on pad B.

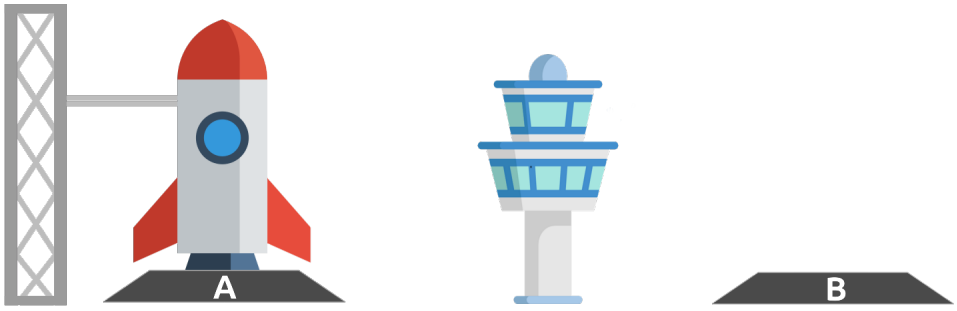

Figure 1: SpaceY launch site

Part I

(50 points) Here is how spacecraft uses the launch site. When a rocket needs to depart, a piece of a rocket needs to land, or a team of engineers and mechanics are ready to start assembly, they will contact the control tower and notify the tower about it. The tower will acknowledge the requesting entity about its position (e.g. 3rd place to land). The tower contacts (signals/wakes up) the rocket/piece/engineers that is in the first position in the corresponding queue and asks it to get ready to launch/land/assemble. Here are more 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.
- Pad A can be used for launching or landing. Pad B for assembly or landing.
- There is only one job being done at each pad at a time.
- After a rocket takes off or lands or has been assembled, it has no consequence on the simulation.
- Each spacecraft piece takes $1t$ secs to land. A spacecraft takes $2t$ secs to be launched and $6t$ secs to be assembled.
- Launching rockets/landing pieces/assembly teams have to notify the tower and ask for permission to use a pad.
- Rocket launch and assembly both happen with probability *p/2* at every t secs and notify the tower about readiness.
- A rocket piece becomes ready to land with probability *1-p* at every t secs and notifies the tower that it is ready.
- Since there are more pieces landing than the rockets being launched, you will be asked to use p=0.2, making the probability of launch and assembly 0.1 each and probability of landing 0.8.
- It is not allowed to perform any action without the acknowledgement of the tower.
- The technology doesn't allow the landing pieces of aircraft to wait in the air, so the landing is favored and the pieces land until there are none of them left in the air. The tower will let only one ground job to begin at each pad (one launch and one assembly) and start favouring the landing pieces again.
- At time zero, there is one rocket waiting to take off.
- 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 hours/days but assume t is 2 sec for the purpose of the computer simulation.

Part II

(30 points) Part I may cause starvation to the jobs waiting on the ground. Now assume SpaceY came up with a technology that allows the landing pieces to wait in the orbit to solve the issue. In this section, we will add a maximum wait-time and counter for the jobs on the ground to avoid this situation. The control tower favors the landing pieces and lets the waiting pieces in the orbit to land until one of following conditions hold

- (a) No more pieces are waiting to land,
- (b) 3 or more launches or 3+ assemblies are waiting to be scheduled on the ground.

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

Part III

(10 points)
Now, we will add an emergency landing for a spacecraft. Assume that every $40t$ a spacecraft crashes, causing 2 pieces at once to request emergency landing. Give this plane immediate access to both landing pads under the control of the tower.

Keeping Logs

(10 points)
You are required to keep a log for the events and for the tower, which will help you debug and test your code. The events.log should keep the event no (ID), its status (landing (L), departing (D), assembly (A), or emergency (E)), the request time to use the launch site, the end time (of launch site usage), turnaround time (end time - request time), pad (A or B).

EventID	Status	Request Time	End Time	Turnaround Time	Pad
1	D	0	4	4	A
2	L	1	3	2	B
4	A	2	15	13	B

Output the snapshot of the waiting landing/launch/assembly jobs 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 job IDs waiting in the queue at time n. Feel free to come up with a better representation or GUI.

At 30 sec landing : 16, 18, 20, 21, 22, 25, 27, 28, 29
At 30 sec launch : 17, 23, 24
At 30 sec assembly : 19, 26

Implementation

- The starter code given to you outlines a possible structure of your project, but you are free to make changes.
- You may want to keep a queue for each job type, add jobs to queues at the appropriate times. The queue code is provided to you, but you can edit it if you want to.
- You can use random number generator to generate jobs at the specified probability. For easy debugging, use the same seed.
- You should represent each job as a thread. The control tower should have its own separate thread.
- Start simple. For example simulate first landing only. Add complexity as you are sure the current implementation works (no deadlock, no collusion).
- To sleep pthreads, please use the code provided to you. 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/
- Revisit the condition variable example we covered in the class.

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 120 sec simulation, with random seed 10 and p=0.2.
- any supplementary files for your implementation if needed (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, but keep it clear and short.
- Finally you may perform a demo if requested by the TA if there are issues with your implementation.

GOOD LUCK.