Project 2

Necla Mutlu 64032

30 April 2020

1 Non-Working Parts

- A problem that I realized was while creating emergency planes, rather than making a newly created landing plane an emergency plane, it creates two planes at the same moment, one of which is a landing plane and the other one is emergency plane.

2 Part 1

2.1 Different planes and queues

Run simulation function calls **create plane** and **create emergency plane** functions in appropriate times. Planes created are grouped under three categories. Each category has their own condition and queue.

- Landing planes:
  They are stored in **landing planes** queue. Each landing plane's thread starts to run **notify tower landing** function. In this function, planes adds themselves to the **landing planes** queue. The access to this queue is controlled by **landing queue mutex**. A newly coming landing plane waits on **runway permission landings** condition. When the traffic controller tower broadcasts the threads waiting on this condition, each thread checks whether the plane that they belong to is the first plane waiting on the queue. If it is they are permitted to use runway, otherwise they go back to waiting on the condition. The traffic controller sleeps for 2 seconds for each plane that completes and after that it removes the plane from the queue.

- Departing planes:
  They are stored in **taking off planes** queue. Each departing plane's thread starts to run **notify tower taking off** function. In this function, planes adds themselves to the **taking off planes** queue. The access to this queue is controlled by **taking off queue mutex**. A newly coming taking off plane waits on **runway permission taking offs** condition. When the traffic controller tower broadcasts the threads waiting on this

condition, each thread checks whether the plane that they belong to is the first plane waiting on the queue. If it is they are permitted to use runway, otherwise they go back to waiting on the condition. The traffic controller sleeps for 2 seconds for each plane that completes and after that it removes the plane from the queue.

- Emergency planes:
  They are stored in **emergency** queue. Each emergency plane's thread starts to run **notify tower emergency** function. In this function, planes adds themselves to the **emergency** queue. The access to this queue is controlled by **emergency mutex**. A newly coming emergency waits on **runway permission emergencies** condition. When the traffic controller tower broadcasts the threads waiting on this condition, each thread checks whether the plane that they belong to is the first plane waiting on the queue. If it is they are permitted to use runway, otherwise they go back to waiting on the condition. The traffic controller sleeps for 2 seconds for each plane that completes and after that it removes the plane from the queue.

2.2 The Traffic Controller Tower Mechanism

When created, the traffic controller thread starts to run **check notifications** function. This function has some helper functions:

- **let emergencies**: It lets emergencies to use the runway until there is no plane left in the emergency queue.
- **let one to take off**: If number of planes waiting in the taking off queue is less than 5, by using this function the traffic controller lets one of the planes in the taking off queue to use the runway.
- **let one to land**: If there is a plane in the landing planes queue, it lets one of them to use the runway.
- **let landings**: By using this function, the traffic controller lets planes waiting in the landing queue until the queue becomes empty or more than 5 planes start to wait in the taking off queue. When one of those conditions become true, it lets one of the departing planes to use the runway and returns.
- **let taking offs**: While 5 or more planes waiting in the taking off queue, this function lets them use the runway one by one and after each departing plane's completion, it lets one of the planes waiting in the landing planes queue to land in order to avoid starvation.

The traffic controller handles giving permissions in this way: It gets the runway mutex, and it broadcasts threads waiting on related condition, then releases the lock. Each thread waken up by tower checks whether it is the first one in the

related queue. If it is not it goes for a wait on condition. If it is, it sets the **completed action** field of its plane to true. The traffic controller sleeps for 2 seconds and checks the first plane in the related queue and if its **completed action** field is true, it pops it from the related queue.

3 Part 2

In this part, starvation can occur for landing planes because if the number of departing planes is too much, after some point, the number of waiting departing planes will always be more than 5. This means that no landing plane will have a chance to use the runway. To avoid this situation, when the number of waiting departing planes is higher than 5, the traffic controller lets one of the landings to use runway after each taking off plane in my system. This continues until the number gets lower than 5. After that, the tower continues to prioritize landings.

4 Part 3

I explained which mutexes, queues etc. were used to schedule those planes in part 1. To guarantee that those planes get immediate access to runway, in each of those helper functions, **let emergencies** function is called first. So that, the traffic controller checks whether there is a plane waiting in the emergency queue and lets them first.

5 Keeping Logs

Logs are written by the traffic controller thread. After popping planes from their queues subsequently to their completion, it calls **print log** function. To report the planes on ground and air, I have a separate thread that runs **report planes** function. This thread busy waits until the nth second comes and then prints the planes in the queues by creating local copies of the queues.