CSCI 530 – Operating Systems
Project Guidelines
Fall 2022

In this project we want to simulate the operation of a simple airport. Our airport has one runway that will be used to service both planes landing and departing. The runway can service one plane at a time: either to land the plane or to depart. We want to write a C++ program to simulate this airport. In order to do that, you need to maintain two queues, one for landing and the other for takeoff. The following are the specifications of the problem:

We are going to implement an airport simulator using threads and semaphores. Our program will have:

1) *main* process (runs the *main* function): responsible for displaying the menu, starting the simulation, stopping the simulation, and displaying statistics about the simulation.
2) Plane Creation: this thread executes a function that continuously creates a plane that is either departing or arriving. This thread runs a loop that repeats until the user from the *main* process decides to end the simulation. The loop does the following:
    a. first sleeps a random time between 0 and 5 seconds.
    b. Randomly decides whether the next plane should be arriving or departing.
    c. Creates a plane.
    d. Adds the new plane to the correct queue (landing or departing queue).
3) Arrival Service: this thread through a loop continuously checks for planes that are waiting for arrival. The loop stops when the user from the *main* process decides to end the simulation.
    a. first sleeps a random time between 10 and 20 seconds.
    b. Checks whether the runway is empty.
    c. If it is, take the plane at the front of the landing queue and have it land on the runway. Displays the following on the screen: *Flight# A-- arrived*

4) Departure Service: this thread through a loop continuously checks for planes that are waiting to depart. The loop stops when the user from the *main* process decides to end the simulation.
    a. first sleeps a random time between 10 and 25 seconds.
    b. Checks whether the runway is empty.
    c. If it is, take the plane at the front of the departing queue and have it flyoff. Displays the following on the screen: *Flight# D-- departed*

Since our program uses multiple threading, then we need to handle concurrency and manage the shared resource in our program. We are going to use semaphores to satisfy mutual exclusion when a thread accesses a shared resource. In our simulation problem we need to worry about the following shared resources:

- The runway
- Integer variables accessed by the multiple threads
- Queues used in the program

- *cout* statements (optional)

Semaphores in C++

We are going to use C's library (POSIX implementation) for semaphores. Therefore, we need to include the following library:

```
#include <semaphore.h>
```

We declare and initialize a semaphore using the following instruction:

```
sem_t sem;
sem_init(&sem, 1, 1);
```

The 1st instruction declares the semaphores in a variable called *sem*. The second instruction initializes the semaphore by sending the address of *sem*, sets its sharing option to global (can be shared by all threads), and gives it an initial count value of 1. The *sem_wait* and *sem_post* instructions stand for calling the semaphore to wait, and signal.

```
sem_wait(&sem);
sem_post(&sem);
```

Finally, if we do not need the semaphore anymore then we can dispose of it using the *sem_destroy* instruction.

```
sem_destroy(&sem);
```

Example

Here is an example run of the simulator. The main function starts the simulation and displays the menu. At this time, all three processes are running. When the user prints the statistics, we can see that there is one plane waiting in the queue to depart. Here the user input is highlighted in blue. Next, our service threads display that 3 plane were serviced: a plane landed (A1), another departed (D1), and another landed (A2). Service messages are highlighted in green. The user chooses to print the simulation statistics, then stops the simulation. The main function waits for the threads to finish, where 4 additional planes are serviced, then the program terminates.

```
Choose an option:
(1) Print Statistics
(2) Stop Simulation
1
Planes Serviced:
----------------
Departure Serviced: 0
Arrival Serviced: 0
Waiting Queue:
----------------
Departure Queue: 1
Arrival Queue: 0

Choose an option:
(1) Print Statistics
```

```
(2) Stop Simulation
Flight# A1 arrived
Flight# D1 departed
Flight# A2 arrived
1
Planes Serviced:
----------------
Departure Serviced: 1
Arrival Serviced: 2
Waiting Queue:
----------------
Departure Queue: 3
Arrival Queue: 3
Choose an option:
(1) Print Statistics
(2) Stop Simulation
2
Flight# D2 departed
Flight# A3 arrived

Flight# A4 arrived
Flight# D3 departed

Process returned 0 (0x0)   execution time : 45.328 s
Press any key to continue.
```

Allowed Libraries

In this project we are **only allowed to use** the following libraries:

- Queue
  - https://cplusplus.com/reference/queue/queue/?kw=queue
- sleep_for
  - https://cplusplus.com/reference/thread/this_thread/sleep_for/
- C++ thread library
  - https://cplusplus.com/reference/thread/
- C semaphore library
- C++ *rand* function

C++ IDE

http://www.codeblocks.org/downloads/binaries/

If you are using Windows, make sure to choose *mingw-setup.exe* if your machine does not have a C++ compiler.

Project Deliverables:

The project should either be done in groups of 2 or individually. You should prepare a 10 to 15 minutes video presentation to demonstrate the project. If you are a group of two, then both members should present equally. Main things to focus on in this presentation:

- Explain the code
- Explain clearly how did you handle mutual exclusion
- Demo run of the airport simulation
- Any challenges faced during implementation

What to submit:

- The presentation video as a YouTube link
- The C++ code file/s

Deadline:

- Tuesday of **Week 15**.