**CIS 657 Lab 3**

Submit two files:

* Create your Lab 3 document -- readme (doc or pdf) including the **disclosure form**, **screenshot(s) of your output**, **your solution code (filename including full path and your own code (created or modified by you), not Nachos original code) with description (you need to explain your solution)**  to the **Turnitin link** on the blackboard (under Assignments).
* Do a “make clean” in code/build.linux, then compress your Nachos folder on the VM, download the compressed file into your local machine and name the compressed file **Lab3\_yourNetId.zip (Note: Zip only!!)**. Submit the zipped file to the **Assignment Code Submission link** of Blackboard.

Due: Oct. 5 (Friday, end of the day)

Late submission: you have total 72 hours throughout the semester.

*Follow the Lab1 instruction and create a new fresh Nachos folder.*

**Overview**

You are going to design a flight reservation system in Nachos. The system will handle requests from passengers boarding in City A, B, C and D for seats on the flight whose route as follows: City A 🡪 City B 🡪 City C 🡪 City D 🡪 City E. City E is the last destination and some passengers will get off in the intermediate cities (B, C and D), and the system will assign these available seats according to the requests. We are assuming the followings:

* The aircraft has **only 20 seats**, which are equally preferable to the passengers (No seat selection)
  + Each seat has unique number (for example, 1 to 20)
* 5 requests at each city A, B, C and D.
  + For each request, the number of passengers should be randomly generated between 1 and 5.
  + Every passenger in the same request has the same itinerary; all passengers in the same party will get off at the same city.
  + The requests are granted in first-come-first-serve basis.
* At each intermediate city, some passenger(s) will get off and their seats will become available.
* If there are not enough seats to satisfy a request, that request will be discarded and next one will be examined until there is neither any seat nor request left.

**Implementation Requirements:**

* Define **Request class**, which contains all the information you need for a request, and use it to create 5 requests each city except City E.
  1. Implement the Request class in the new files (declaration in .h file and definition in .cc file)
  2. Define the class for holding necessary information:
     + unique ID for the request
     + number of passengers (number of needed seats)
       - randomly generated (1 to 5)
     + assigned seat numbers to all passengers
     + departure city
     + destination city of the request
       - randomly generated by your own probability
       - for example, a request made in City A would have a probability of getting off: City B (10%), City C(20%), City D(30%) and City E(40%)
     + any other information you want to add
  3. The member variables have to be private. Implement necessary functions including getter/setter functions for accessing these variables
  4. Change code/build.linux/Makefile to include the new files to the NachOS compilation process
* Use the **Bitmap class** under code/userprog/ to keep track of the availability of 20 seats.
  1. If a seat is taken, it is set to 1 in your bitmap.
  2. If passengers of a request get off, all the seats taken become available, i.e., 1’s are back to 0’s in your bitmap.
  3. Do not change Bitmap class
* Use the **List class** to store requests in the following categories:
  1. Currently on the plane
  2. Discarded because there was no available seat
* You can provide your own operations as new public member functions for List class.
* Use threads to simulate this flight reservation system.
  1. **Flight thread** is responsible for simulating the flight(s) and creating Reservation threads. You can either use a single thread for all the flights or use individual thread for each flight.
  2. **Reservation thread** is responsible for generating requests, assigning seats to a request, getting a request off the plane. Each Reservation thread deals with one request
* Tips: A thread is a process in Nachos. Each thread is assigned a function to run when **Thread::Fork()** is called. The calling thread will be put at the end of the ready queue (need to check, implemented as a FIFO queue).
* We assume that there is no interrupt, so each thread will run till completion or its calling of **Thread::Yield()** or **Thread::Sleep()**.
  1. A thread calling of Thread::Yield() will give up the CPU and go back to the end of the ready queue. A thread calling of Thread::Sleep() will give up the CPU. By calling **Scheduler::ReadyToRun**(the sleeping thread) you can put the sleeping thread to the end of ready queue.
* **You need to create threads and call Yield or Sleep at correct location to ensure that threads run in your desired order.**
  1. Starting in ThreadTest(), one Flight thread can be created and forked.
  2. The Flight thread does its job (such as printing) and creates 5 Reservation threads. It finishes (by calling Thread::Finish() if you have multiple Flight threads) or calls Yield to give CPU to the Reservation threads.
  3. Reservation thread can generate and process a request. If the request cannot be granted, then the thread finishes by calling Finish. Otherwise, the request is granted and the current (Reservation) thread will be stored in a list corresponding to a destination city. Then the current thread calls Sleep to give up CPU. After all the five Reservation threads, the Flight thread runs again (if you used Yield in the step 2). If you used Finish in the step 2, you need to create a new Flight thread.
  4. When a destination city is reached, the Flight thread will call Scheduler::ReadyToRun() on all the threads in the list for this city to wake them up, and they will get their request off the plane.

**Output:**

For each city (A, B, C, D and E)

* If it is a destination city for any Reservation thread (request), passengers of the request get off.
  + Print out ID of the request, the seat numbers returned, and available numbers after the return
* After getting passengers off at their destination city, print out IDs of all the requests currently on the plane and occupancy rate of seats.
* If it is not the last city (City E)
  + When a request is generated, print out ID, number of seats needed, and the destination city.
  + After assigning seats for a request, print out ID, the seat numbers assigned, and total number of available seats and available seat numbers after the assignment.

After simulating threads, you need to print out the followings:

* All the requests discarded throughout the simulation due to lack of available seat.

**Testing:**

We will build and run your Nachos on the VM.

**Grading:**

* Well written and commented in source code and readme [-10]
* Syntax/Runtime/Logic Errors with proper Makefile [-50, -15]
* Class declaration/definition respectively [-20, -10]
* Satisfy all implementation requirements [-50, -5]
* Thread manipulation (Flight and Reservation threads) [-30,-5]
* Outputs contain all the required information [-20, -5]
* Output(by Students)/Test(by TAs) [-50]