**CIS 657 Lab 3**

Submit three files:

* Create your Lab 3 document (doc or pdf), follow the instruction and then submit it to the **Turnitin link** on the blackboard (under Assignments).
  + You can submit multiple times before the due.
  + After the due, you cannot resubmit newer version.
* 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.
* Save your simulation output to text file naming **Lab3Output\_yourNetId.txt (Note: plain text file)**. Submit the zipped file to the **Assignment Output Submission link** of Blackboard.
* You have to make sure your submission is correctly made
  + If you don’t have a confirm email, you should check again.
* **You should raise an appeal within a week after grade is posted.**

Due: Sep. 27 (Friday, end of the day)

Late submission: you will have 2d penalty of your late days.

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

**Overview**

You are going to design a railway ticket reservation system which will help the passengers to book their tickets, search train between two stations and get details of particular train schedules including their fare details. After passengers enter the departure and destination station name, the system will be able to provide the list of all available trains for particular destination and their arrival and departure time and date. The system will handle requests from passengers boarding at station Syracuse, Rochester, Buffalo, … on the train whose route would be loaded from the train route/fair information file. Station X is the destination of some passengers and they will get off, and their seats will be available again. Then the system will assign these available seats according to new requests.

We are assuming the followings:

* Each train has its unique id and route.
  + Train schedule information includes the following minimum information:
    - Train number
    - Route (for example, station NYC (Penn st), … , Syracuse, … , Toronto), Arrival, departure time at each station
    - Fares per class, between departure and destination stations
    - Number of seats per class
* There are two types of seats: business and coach.
  + Among the same class, all seats are equally preferable to the passengers (NO seat selection)
    - 20 seats for business class and 40 seats for coach class
  + Each seat has unique number
* There are 20 stations in this region.
  + Each station id is unique
* 5 requests every 10 min. (simulation time unit)
  + For each request,
    - Unique request id
    - Departure/Destination station should be randomly generated (cannot be same as the departure destination)
    - Departure time should be randomly generated (cannot be earlier than current time)
    - the class of seat should be randomly generated (Business or Coach)
    - the number of passengers should be randomly generated between 1 and 8
    - Each passenger in the same request has the same itinerary; all passengers in the same party will get off at the same station
    - The requests are granted in first come first serve basis
  + The system will display all available trains for the request
    - One of them will be randomly selected
  + If there is no match for departure and destination stations in any route of the system, the request will not be served (refused)
  + If there are not enough seats to satisfy a request, that request will not be served (rejected)
* For a refused request,
  + It will be discarded and next request will be examined until there is neither any seat nor request left

Run the railway ticket reservation simulation from 6 am to 10 pm using the following:

* For each 10 minutes of the day:
  + If a new request is made,
    - Check all train schedule and display matched trains
    - If there are multiple available trains, select a random train and enqueue the granted itinerary
  + When a train arrives at a station,
    - Passengers reaching their destination get off the train
  + When a train departs from a station,
    - Passengers with the granted itinerary of this station take the train
* When the simulation begins, a schedule file is read and the system runs all trains according to the schedule

**Implementation Requirements:**

* Design/define classes, which contains all the information you need for this project
  + Implement all necessary class in the new files (declaration in .h file and definition in .cc file)
  + The member variables have to be private. Implement public getter/setter functions for accessing these variables
  + Change code/build.linux/Makefile to include the new files to the NachOS compilation process
* Call rand() function to randomly generate a number.
* Use the **List class** to store requests in the following categories:
  1. All granted requests
  2. All refused requests
  3. Currently on a train
* You can provide your own operations as new public member functions for List class.
* Use the **Bitmap class** under code/userprog/ to keep track of the availability of 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., 0’s in your bitmap.
  3. Do not change Bitmap class
* Use threads to simulate this train reservation system.
  1. **Admin thread** is responsible for simulating the reservation system, creating Train threads, and creating Reservation threads.
     + One thread
  2. **Reservation thread** is responsible for generating requests, assigning seats to a request, taking a train and getting a request off. Each Reservation thread deals with one request.
     + 5 threads created per simulation time
  3. **Train thread** is responsible for operating trains in accordance to their schedule, and making passengers to get in or out of the train
     + N number of threads based on your scheduled trains from the file
* 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.**For example,
  1. Starting in ThreadTest(), one Admin thread can be created and forked.
     + Read your train schedule for the day
     + Create all Train threads
  2. The Admin thread does its job (such as printing) and creates 5 Reservation threads. It calls Yield to give CPU to other 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 train. Then the current thread calls Sleep to give up CPU.
  4. When a train arrives at a station as scheduled, its Train thread will call Scheduler::ReadyToRun() on all the threads in a boarding list for this station to wake them up, and they will get their passengers in the train.
  5. When a train departs at a station as scheduled, its Train thread will call Scheduler::ReadyToRun() on all the threads in a list for this station to wake them up, and they will get their passengers off the train.
* **You need to make your own train schedule running at least 5 trains using 20 stations for the given simulation time.** Each train may have a different route, arrival & departure time for each station for the route.

**Output:**

You should print out appropriate messages in a good including the following information:

* Request per simulation time
  + Granted and refused
* Train operation information per simulation time
  + # of itinerary, # of passengers for boarding at a station of running trains at the time
  + # of itinerary, # of passengers for getting off at a station at the time
* Simulation summary
  + Total # of requests
  + Total # of granted requests
  + Train operation summary
    - Total served itinerary
    - Total passengers
    - Busiest section

**Lab3 document must include the followings:**

* Cover Page
* Disclosure Form
* How much time did you spend to do:
  + Analyze the problem, determine specifications, and create your design
  + Implement the design
    - write the program
  + Test/debug the program
    - Find/fix errors
    - Create test cases, and try them out
* Design/your solution for each requirement
  + We are looking for your own answers
* Implementation of your solution (Code snippet with good comments)
  + Do not include Nachos original code
  + We want to see your own code/modification
    - You need to modify threadtest.cc
    - Your new classes
  + Should be text (no image/photo of code)
* Testing
  + How to run your tests
  + What each test case is testing (you need to prove your implementation of each requirement)

**Testing:**

**We will build and run your Nachos on the VM.** We will test your program with different random seed. You must test your program on the VM and also should provide proper test scenario in the Lab 3 document.

**Grading:**

* Syntax/Runtime/Logic Errors with proper Makefile [-50, -15]
* Proper Thread Scheduling [-30, -20]
* Class declaration/definition respectively [-20, -10]
* Satisfy all implementation requirements [-50, -5]
* Input/output design [-10, -5]
* Output(by Students)/Test(by TAs) [-50]