**Overview**

You are going to design a **hotel reservation system** in Nachos and run a simulation. You are the owner of a hotel of 30 rooms, which has a unique room number. Guests come and leave frequently. Each guest may ask for x number of rooms for y number of nights. After spending z number of nights (0 <= z <= y), the guest leaves and all of her/his rooms become available again.

You need to design a system to keep track of the availability of the rooms and information of the guests. The system will handle reservation requests from guests checking in the hotel for rooms, and assign rooms according to the requests. We are assuming the followings:

* The hotel has **30 rooms**, which are equally preferable to the guests (No room selection)
  + Each room has unique number (for example, 1 to 30)
* Each guest has his/her unique ID (can be generated sequentially).
  + For each guest, the number of rooms should be randomly generated between 1 and 5.
  + For each guest, check-in & check-out dates should be randomly generated with random number of nights between 1 and 4.
  + The guest’s request is granted in first-come-first-serve basis.
* Everyday some guest(s) will leave and their rooms will become available.
* If there are not enough rooms to satisfy a request, that request will be discarded and next one will be examined until there is neither any room nor request left.

**Requirements & Tips**

* We will do the simulation with 11 business days (10 nights).
  + The last day (11th day) should check out all staying guests.
  + No new quest’s request on that day.
* Each guest has a unique ID and each room has a unique room number.
* You must keep track of the availability of the 30 rooms for 10 business day nights.
  + Tip: Bitmap class defined in Nachos will be very useful
    - If a room is taken, it is set to 1 in your bitmap.
    - If a guest leaves, all the rooms taken become available, i.e., 1’s are back to 0’s in your bitmap.
    - Do not change Bitmap class
  + You can keep track of the availability with List class
* If there are not enough rooms to satisfy a guest’s request, the request cannot be accommodated.
* Define **Guest class**, which contains all the information you need for a request.
  + Implement the Guest class in the new files (declaration in .h file and definition in .cc file)
  + Define the class for holding necessary information:
    - unique ID for the guest
    - number of rooms needed
      * randomly generated (1 to 5)
    - assigned room numbers
    - check-in date/check-out date
      * number of nights should be randomly generated (1 to 4)
      * check-in date must be equal or later than current simulation date (random)
    - any other information you want to add
  + The member variables have to be private. Implement necessary functions including getter/setter functions for accessing these variables.
  + Implement under /threads folder.
  + Change code/build.linux/Makefile to include the new files to the NachOS compilation process.
* You must keep track of the information of the guests.
  + Use the **List class** to store requests in the following categories:
    - Staying (Checking in)
    - Checking out
    - Confirmed (Not checking in yet)
    - Discarded because there were not enough available rooms
    - Tip: Staying list – sorted by check-out date, Confirmed list – sorted by check-in date
    - You can modify this List class for your purpose.
* Use threads to simulate this online hotel reservation system.
  + **Concierge thread** is responsible for simulating the hotel and creating Guest threads.
  + **Guest thread** is responsible for generating guests, assigning rooms to a guest’s request, checking in/out. Each guest 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()**.
  + 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 Concierge thread can be created and forked.
  2. The Concierge thread does its daily job (explained in Step 4 and 5) and creates 5 Guest threads. It calls Yield to give CPU to the Guest threads.
  3. Guest thread can generate and process a request. If the request cannot be granted (confirmed), then the thread finishes by calling Finish, and the request needs to be added into the Discarded list. Otherwise, the request is granted and the current (Guest) thread will be stored in a corresponding list (either Staying if check-in date is same as the simulation date or Confirmed). If the guest is in Staying, his/her checking in process follows. Then the current thread calls Sleep to give up CPU. After all the five Guest threads, the Concierge thread runs again.
  4. When a check-out date of staying guest is reached, the Concierge thread will call Scheduler::ReadyToRun() on all the threads in the Staying list for the day to wake them up, and they will leave the hotel by calling Finish. The guest needs to be moved to the Checking out list. After waking all guests up to check out, the Concierge thread need to call Yield for the checking out Guest threads.
  5. When a check-in date of confirmed guest is reached, the Concierge thread will call Scheduler::ReadyToRun() on all the threads matched among the Confirmed list for the day to wake them up, and they will check in the hotel followed by calling Sleep. The guest should be in the Staying list. After waking all guests up to check out, the Concierge thread need to call Yield for the checking out Guest threads.

**Output**

* When a guest makes a reservation, print out the information of the guest: ID, # of rooms, check-in/out dates, assigned room numbers if granted, and print out the available room numbers after the assignment.
* When a guest checks in the hotel, print out the guest info.
* When a guest leaves, print out the ID of the guest and room numbers he/she returned, and print out the available room numbers after the return.
* After 11 days simulation, print out the summary of guests and room availability.
  + For example, daily vacancy/occupancy rate, granted rate, …