CIS657 Spring 2019

Lab Assignment 3

Submitted by –

Name – Anish Nesarkar

SUID – 368122582

Subject – Operating Systems

Lab 3 Assignment

CIS657 Spring 2019

Assignment Disclosure Form

Assignment #: 3

Name: Anish Nesarkar

1. Did you consult with anyone other than instructor or TA/grader on parts of this assignment?

   If Yes, please give the details.

2. Did you consult an outside source such as an Internet forum or a book on parts of this assignment?

   If Yes, please give the details.

I assert that, to the best of my knowledge, the information on this sheet is true.

Signature:_____Anish Nesarkar_____ Date : 3/1/2019

Design And Implementation:

A hotel reservation system is being simulated for 11 business days. The hotel has 30 rooms which has a unique room number. The objective is to design a system that keeps track of availability of the rooms and information of the guests. The goal of the system is to handle reservation requests from guests checking in the hotel for rooms, and assigning the rooms according to the requests.

The following are the requirements and constraints for the system:

a. The hotel has 30 rooms that are equally preferable to the guests ranging from 1 to 30.
b. Each guest has a unique ID that is generated sequentially.
c. The number of rooms for each guest is generated randomly from 1 to 5.
d. The number of nights a guest can stay is generated randomly from 1 to 4.

The following explanation talks about the design and implementation of the system:

Guest.cc

```
#include "guest.h"
#include "list.h"
#include<iostream>
#include <cstdlib>

//Guest constructor
guest::guest() {

}

//return the ID of the guest
int guest::getId()
{
    return id;
}

//set the ID of the employee
void guest::setId(int Id)
{
    id = Id;
}


//get the assigned rooms of the guest
List<int> *guest::getRooms()
{
```

```
    return assignedRooms;
}

//set checkIn date for the guest
void guest::setCheckInDate(int cInDate)
{
    checkInDate = rand() % (11 - cInDate) + cInDate;
}

//get the checkIn date of the guest
int guest::getCheckInDate()
{
    return checkInDate;
}

//set checkOut date of the guest
void guest::setCheckOutDate(int cOutDate)
{
    checkOutDate = cOutDate;
}

//get checkOut date of the guest
int guest::getCheckOutDate()
{
    return checkOutDate;
}

//get number of rooms
int guest::getNumberRooms()
{
    return rooms;
}

//set number of nights
void guest::setNights()
{
    NumNights = (rand() % 4) + 1;
}

//get number of nights
int guest::getNumNights()
{
    return NumNights;
}
```

```
//set number of rooms
void guest::setNumberRooms()
{
    rooms = (rand() % 5) + 1;
}
```

Guest.h

```
#include "string.h"
#include "list.h"
class guest {
    private:
        int id; // ID of the guest
        int rooms; //number of rooms occupied by the guest
        int NumNights; //number of nights stayed by the guest
        int checkInDate; //check in date of the guest
        int checkOutDate; //check out date of the guest
        int numberOfNights; //number of nights stayed by the guest
    public:
        guest(); //Guest Constructor
        ~guest(){
            delete assignedRooms;
        } //Guest Destructor

        void setId(int Id); //Set guest ID
        int getId(); //Return guest ID

        List<int> *getRooms(); //return number of rooms
        List<int> *assignedRooms = new List<int>(); // list of room numbers

        void setCheckInDate(int cInDate); //set checkin date of the guest
        int getCheckInDate(); //get check in date of the guest

        void setCheckOutDate(int cOutDate); //set check out date of the guest
        int getCheckOutDate(); //get checkout date of the guest

        void setNumberRooms(); //set number of rooms for the guest
        int getNumberRooms(); //get number of rooms of the guest

        void setNights(); //set number of nights for the guest
        int getNumNights(); //get number of nights for the guest
};
```

- A guest class is created for the guest which has setter and getter functions for the following information
    - ID of the guest
    - Number of rooms needed for the guest (1 – 5)
    - List of Assigned room numbers
    - CheckIn date of the guest
    - CheckOut Date of the guest
    - Number of nights stayed by the guest
- A corresponding guest header file is created for the guest class.

ThreadTest.cc

```
//------------< Defined function that creates child thread for executing program
>--------------
void myFunction()
{
    //create a new thread and fork it to worker
    Thread *t = new Thread("Conceirge thread");
    t -> Fork((VoidFunctionPtr) worker, (void *) 1);
}
```

- Threadtest function is invoked from the Main.cc
- The threadtest function forks the concierge thread which is the worker thread.
- The simulation is started in the worker thread.

Concierge thread

```
//starting concierge thread
void worker()
{

//initializing bitmap
for(int i=1; i <= 11; i++)
    {
        rooms[i] = new Bitmap(30);
    }
cout << endl;
cout << ">=========================================< Simulating Hotel
Reservation System >=========================================<" << endl <<
endl;
```

```
//simulating each day
    while(simulatedTime < 11)
    {

        //confirm list iteration
        ListIterator<Thread*> *confirmedListIterator = new
ListIterator<Thread*>(confirmedList);
        while(!confirmedListIterator->IsDone())
        {
            Thread *t = confirmedListIterator->Item(); //"t" pointing to current
thread
            kernel->scheduler->ReadyToRun(t); // putting the thread 't' to
ReadyToRun state
            confirmedListIterator->Next();
        }

        //iterating staying list
        ListIterator<Thread*> *stayingListIterator = new
ListIterator<Thread*>(stayingList);
        while(!stayingListIterator->IsDone())
        {
            Thread *t = stayingListIterator->Item(); //"t" pointing to current
thread
             kernel->scheduler->ReadyToRun(t); // putting the thread 't' to
ReadyToRun state
             stayingListIterator->Next();
        }

        cout << ">=====================================================< Day : "
<< simulatedTime << "
>=====================================================<"<<endl << endl;
        for(int i = 0; i<5;i++)
        {
            Thread *t = new Thread("guest request");
            t->Fork((VoidFunctionPtr) guestRequest, (void *) 1);
        }


        kernel->currentThread->Yield(); //yielding concierge thread

        displayOccupancy(simulatedTime); //displaying occupancy rate

        simulatedTime++; //incrementing the days
     }
     cout << endl << endl;
```

```
     cout << ">========================================================= Day : " <<
simulatedTime << "
>================================================================<"<<endl;
    //checking out remaining guests on Day 11
    ListIterator<Thread*> *stayingListIteratorRemaining = new
ListIterator<Thread*>(stayingList);
        while(!stayingListIteratorRemaining->IsDone())
        {
            Thread *t = stayingListIteratorRemaining->Item();
            kernel->scheduler->ReadyToRun(t);
            stayingListIteratorRemaining->Next();
        }
    kernel->currentThread->Yield();
    float x = rooms[simulatedTime]->NumClear(); //occupancy rate on day 11
    cout <<
"================================================================================"<<endl;
    cout << "Occupancy rate on Day: " << simulatedTime << " is " << ((30 -
x)/30)*100 << "%" << endl;
    cout <<
"================================================================================"<<endl;
    cout << endl;


    cout << endl << endl;
    cout << ">=================================================< Summary of
people who stayed >=================================================< " << endl
<< endl;


    //total requests generated
    totalRequestsGenerated();

    //total granted requests
    grantedRequests();

    //discarded requests
    discardedRequests();

cout << ">=============================================< End Simulation
>=================================================< " <<endl;

    //clearing check out list
    clearCheckOutList();

    //clearing discarded list
```

```
    clearDiscardedList();

    //deleting bitmap
    clearBitMapHeap();

    //finishing the concierge thread
    kernel->currentThread->Finish();
}
```

- The worker thread creates a 2D bitmap for all the simulated days.
- The worker thread then starts the simulation from day 1.
- The worker thread forks 5 new guest threads every simulated day till day 10.
- After creating guest requests, the worker thread then puts all the staying guest list of threads to ready to run to check if they want to check out. If not, it puts all the threads back to sleep. If, the guest wants to check out on that simulated date, the guest thread puts that guest object to check out list and finishes the thread.
- The worker thread then iterates through the confirmed list. If the reservation date is equal to the current date, it puts the thread to staying list and removes it from the confirmed list. If reservation date is not same as current date, the thread goes to sleep again.
- On the 11th day, the worker thread puts all the staying guests to check out list, prints out the summary of the simulation and clears the heap of memory created by lists and bitmap.

Guest thread

```
void guestRequest()
{
    guest *g = new guest(); //creating guest object pointer
    g->setId(uID); //setting ID to the guest
    uID++;
    g->setCheckInDate(simulatedTime); //setting checkin date
    g->setNumberRooms(); //setting number of rooms
    g->setNights(); //setting number of nights
    g->setCheckOutDate(g->getCheckInDate() + g->getNumNights()); //setting
checkout date
    selectRooms(g); //selecting available rooms
    bool isRoomAvailable = getAvailableRooms(g); //check if rooms are available

//put the guest to discarded list when rooms are not available
if(!isRoomAvailable)
{
```

```
    cout << ">-------------------------------------------------< Request
Discarded >-------------------------------------------------<" << endl << endl;
    cout << "Rooms Not available" << endl;
    printDiscarded(g);
    discardedList->Append(g);
    kernel->currentThread->Finish();
}

//else check if guest wants to check in or confirm for future date
else
{
    //check if current date is equal to check in date
    if(g->getCheckInDate() == simulatedTime)
    {
        //dispaly check in information of the guest
        cout << ">-------------------------------------------------< Reservation
made!!! >-------------------------------------------------<" << endl <<
endl;
        cout << "Guest Checking In " << endl;
        printCheckInGuestInfo(g);
        cout << endl;
        //set the bitmap
        setBitMap(g);
        //display available rooms
        cout << "Number of availabe rooms : " << rooms[g->getCheckInDate()]-
>NumClear() << endl;
        //append the current thread to staying list
        stayingList->Append(kernel->currentThread);
        //sleep the thread
        kernel->currentThread->Sleep(FALSE);
        //put the thread to sleep again if check out date is not equal to current
date
        while(g->getCheckOutDate() != simulatedTime)
        {
            if(g->getCheckOutDate() > 11 && (simulatedTime == 11))
                break;
            kernel->currentThread->Sleep(FALSE);
        }

        //adding guest to check out list
        checkOutList->Append(g);

        //display checkout list
        printCheckOutGuest(g);
```

```
    //clear the bit map
    clearBitMap(g);
    //remove the thread from the staying list
    stayingList->Remove(kernel->currentThread);
    //finish the thread
    kernel->currentThread->Finish();

    }
    //put guest to confirmed list
    else
    {
        //display confirm guest information
        cout << ">----------------------------------< Reservation made for
future date : " << g->getCheckInDate() << " >----------------------------------
--<" << endl << endl;

        printCheckInGuestInfo(g);
        setBitMap(g);
        //available rooms
        cout << "Number of availabe rooms : " << rooms[g->getCheckInDate()]-
>NumClear() << endl;

        //append the thread to confirmed list and put the thread to sleep
        confirmedList->Append(kernel->currentThread);


        kernel->currentThread->Sleep(FALSE);

        //until checkin date is not equal to current date, put back the thread to
sleep
        while(g->getCheckInDate() != simulatedTime)
        {
            kernel->currentThread->Sleep(FALSE);
        }

        //transfer this thread to staying list thread
        stayingList->Append(kernel->currentThread);
        confirmedList->Remove(kernel->currentThread);

        //put the thread to sleep again if check in date is not equal to current
date
        while(g->getCheckInDate() != simulatedTime)
        {
            kernel->currentThread->Sleep(FALSE);
        }
```

```
        //display check in guest information
        cout << endl;
        cout << ">-------------------------------------------------------< Check
In >-------------------------------------------------------<"<< endl << endl;
        cout << "Guest Checking In " << endl;
        printCheckInGuestInfo(g);

        //put the thread to sleep again if check out date is not equal to current
date
        while(g->getCheckOutDate() != simulatedTime)
        {
            if(g->getCheckOutDate() > 11 && (simulatedTime == 11))
                break;
            kernel->currentThread->Sleep(FALSE);
        }

        //adding guest to check out list
        checkOutList->Append(g);

        //display checkout list
        printCheckOutGuest(g);

        //clear the bitmap
        clearBitMap(g);
        //remove the thread from the staying list
        stayingList->Remove(kernel->currentThread);
        //finish the thread
        kernel->currentThread->Finish();
    }

}

}
```

- 5 guest threads are spawned every day by the concierge thread.
- A guest pointer object is created and all its attributes are set by the setter functions.
- The guest thread then checks if there are available rooms for the guests according to the requirements.
- If the rooms are not available, the guest object pointer is added to discarded list and the guest is finished.
- If the check In date is equal to the current date, the guest is checking In and the guest thread is put into staying list.

- If the check In date is future date, the reservation for the guest is made and the guest thread is put into the confirmed list.
- When the guest threads in staying list are put to ReadyToRun state by the Concierge thread, if the check out date is equal to current date, the guests are put into check out list and the thread is finished.

Display functions

```
//displaying discarded list of guests
void printDiscarded(guest *g)
{
    cout << "Guest ID : " << g->getId() << endl;
    cout << endl << endl;
}

//displaying list of guests checked In
void printCheckInGuestInfo(guest *g)
{
    cout << "Guest : " << g->getId() << endl;
    cout << "Check In Date : " << g->getCheckInDate() << endl;
    cout << "Check Out Date : " << g->getCheckOutDate() << endl;
    cout << "Rooms Needed : " << g->getNumberRooms() << endl;
    cout << "Rooms : ";
    ListIterator<int> roomListIterator = g->getRooms(); //get the list of room
numbers
    while(!roomListIterator.IsDone())
    {
        cout << (roomListIterator.Item()) << " ";
        roomListIterator.Next();
    }
    cout << endl << endl;
}

//displaying list of guests that checked out
void printCheckOutGuest(guest *g)
{
    cout << " >------------------< Guest Checked Out >-----------------<" <<
endl;
    cout << "Guest ID : " << g->getId() << endl;
    cout << "Check Out Date : " << g->getCheckOutDate() << endl;
    cout << "Rooms Vacated : ";
    ListIterator<int> roomListIterator = g->getRooms();
    while(!roomListIterator.IsDone())
    {
        cout << (roomListIterator.Item()) << " ";
```

```
        roomListIterator.Next();
    }
    cout << endl << endl;

    if(g->getCheckOutDate() < 11)
        cout << "Number of available rooms : " << rooms[g->getCheckOutDate()]-
>NumClear();
    else
        cout << "Number of available rooms : " << rooms[11]->NumClear();
    cout << endl << endl;

}

//displaying list of granted requests
void grantedRequests()
{
    //checkout list iterator
    ListIterator<guest*> *checkoutListIterator = new
ListIterator<guest*>(checkOutList);
    cout << " >-------------------------< Granted Requests >--------------------
--------<" << endl << endl;
    guest *g = new guest();
    int x = 1;
    while(!checkoutListIterator->IsDone())
    {
        g = checkoutListIterator->Item();
        cout << ">------------------< Granted Request " << x << " >--------------
------< " << endl << endl;
        cout << "Guest : " << g->getId() << endl;
        cout << "Check In Date : " << g->getCheckInDate() << endl;
        cout << "Check Out Date : " << g->getCheckOutDate() << endl;
        cout << "Number of Rooms : " << g->getNumberRooms() << endl;
        cout << "Rooms stayed : ";
        ListIterator<int> roomListIterator = g->getRooms(); //get the list of
rooms
        //display room numbers
        while(!roomListIterator.IsDone())
        {
            cout << (roomListIterator.Item()) << " ";
            roomListIterator.Next();
        }
        cout << endl << endl;
        x++;
        checkoutListIterator->Next();
    }
```

```
}

//displaying list of discarded requests
void discardedRequests()
{
    //discarded list iterator
    ListIterator<guest*> *discardedListIterator = new
ListIterator<guest*>(discardedList);
    guest *g = new guest();
    int x = 1;
    while(!discardedListIterator->IsDone())
    {
        g = discardedListIterator->Item();
        cout << ">------------------< Discarded Request " << x << " >------------
--------< " << endl << endl;
        cout << "Guest : " << g->getId() << endl;
        cout << "Check In Date : " << g->getCheckInDate() << endl;
        cout << "Check Out Date : " << g->getCheckOutDate() << endl;
        cout << "Number of Rooms Needed : " << g->getNumberRooms() << endl;
        cout << endl;
        x++;
        discardedListIterator->Next();
    }
    cout << endl << endl;

}


//displaying total requests generated in simulation
void totalRequestsGenerated()
{
    //list iterators for checkout and discarded list
    ListIterator<guest*> *checkoutListIterator = new
ListIterator<guest*>(checkOutList);
    ListIterator<guest*> *discardedListIterator = new
ListIterator<guest*>(discardedList);
    cout << " >-------------------------< Total requests Generated >------------
----------------<" << endl << endl;
    guest *g = new guest();
    int x = 1;
    while(!checkoutListIterator->IsDone())
    {
        g = checkoutListIterator->Item();
```

```
        cout << ">------------------< Request " << x << " >--------------------<
" << endl << endl;
        cout << "Guest : " << g->getId() << endl;
        cout << "Check In Date : " << g->getCheckInDate() << endl;
        cout << "Check Out Date : " << g->getCheckOutDate() << endl;
        cout << "Number of Rooms : " << g->getNumberRooms() << endl;
        cout << "Rooms stayed : ";
        ListIterator<int> roomListIterator = g->getRooms();
        //display room numbers
        while(!roomListIterator.IsDone())
        {
            cout << (roomListIterator.Item()) << " ";
            roomListIterator.Next();
        }
        cout << endl << endl;
        x++;
        checkoutListIterator->Next();
    }
    //display discarded guest requests
    while(!discardedListIterator->IsDone())
    {
        g = discardedListIterator->Item();
        cout << ">------------------< Request " << x << " >--------------------<
" << endl << endl;
        cout << "Guest : " << g->getId() << endl;
        cout << "Check In Date : " << g->getCheckInDate() << endl;
        cout << "Check Out Date : " << g->getCheckOutDate() << endl;
        cout << "Number of Rooms : " << g->getNumberRooms() << endl;
        cout << endl << endl;
        x++;
        discardedListIterator->Next();
    }
    cout << endl << endl;
}

//display occupancy rate of rooms each day at the end of the day
void displayOccupancy(int stime)
{
    cout << endl;
    //number of available rooms on current date at the end of the day
    float x = rooms[stime]->NumClear();
    cout <<
"========================================================================="<<endl;
    cout << "Occupancy rate on Day: " << stime << " is " << ((30 - x)/30)*100 <<
"%" << endl;
```

```
    cout <<
"=========================================================================="<<endl <<
endl;
}
```

- The above functions print the guest information for Check In process, checkout process, discarded guest request, occupancy rates and the summary of the guests i.e. total requests generated, granted requests and discarded requests.

Bitmap operations

```
//returns "true" if there are available rooms otherwise "false"
bool getAvailableRooms(guest *g)
{
    ListIterator<int> roomListIteratorempty = g->getRooms(); //To check if the
assigned rooms are empty
    if(roomListIteratorempty.IsDone())
        return false;
    else
        return true;
}

//function to clear the bitmap
void clearBitMap(guest *g)
{
    //iterating through the number of days
    for(int j = 0; j <= g->getNumNights();j++)
    {
            //list iterator for the number of rooms
            ListIterator<int> roomListIteratorclear = g->getRooms();
            //clear all rooms on Day 11
            if(g->getCheckOutDate() > 11 && (simulatedTime == 11))
            {
                while(!roomListIteratorclear.IsDone())
                {
                    rooms[11]->Clear(roomListIteratorclear.Item() - 1);
                    roomListIteratorclear.Next();
                }
            }
                //clearing rooms on other days

                while(!roomListIteratorclear.IsDone())
                {
```

```
                    if((g->getCheckInDate() + j) <= 11)
                        rooms[g->getCheckInDate() + j]-
>Clear(roomListIteratorclear.Item() - 1);
                    roomListIteratorclear.Next();
                }

    }
}

//function to select the availabe rooms
void selectRooms(guest *g)
{
    //temporary variables
    int count = 0;
    int countRoom = 0;
    //check if there are available rooms
    if(rooms[g->getCheckInDate()]->NumClear() > g->getNumberRooms())
    {
            if((g->getCheckInDate()) < 11)
            {
                int i = 0;
                //iterate through all the rooms and check which are empty
                while(i < 30)
                {
                    count = 0;
                    //iterate through the number of days and check if that
particular room is empty
                    for(int j = 0; j < g->getNumNights(); j++)
                    {
                        if((g->getCheckInDate() + j) < 12)
                        {
                            if(rooms[g->getCheckInDate() + j]->Test(i) == TRUE)
                                continue;
                            count++;
                        }
                    }
                    //mark the room in the bitmap if its empty for all the days
                    if(count == g->getNumNights())
                    {
                        rooms[g->getCheckInDate()]->Mark(i);
                        g->assignedRooms->Append(i + 1);
                        countRoom++;
                    }
                    //logic to mark the rooms in the bitmap when checkout date
exceeds 11
```

```
                    if(count == (12 - g->getCheckInDate()))
                    {
                        if(rooms[g->getCheckInDate()]->Test(i) == FALSE)
                        {

                            rooms[g->getCheckInDate()]->Mark(i);
                            g->assignedRooms->Append(i + 1);
                            countRoom++;
                        }
                    }

                    if(countRoom == g->getNumberRooms())
                        break;
                    i++;
                }
                //if rooms not availabe, remove them from the list
                if(countRoom < g->getNumberRooms() && countRoom > 0)
                {
                    clearBitMap(g);
                    while(countRoom != 0)
                    {
                        g->assignedRooms->RemoveFront();
                        countRoom--;
                    }
                }
            }
    }
}


//setting the bitmap
void setBitMap(guest *g)
{
    for(int j = 1; j <= g->getNumNights();j++)
    {
        //iterate through the rooms to mark the particular index in the bitmap
        ListIterator<int> roomListIteratorset = g->getRooms();
        while(!roomListIteratorset.IsDone())
        {
            if((g->getCheckInDate() + j) < 12)
            {
                rooms[g->getCheckInDate() + j]->Mark(roomListIteratorset.Item() -
1);
            }
            roomListIteratorset.Next();
```

```
        }
    }
}
```

- The above functions selects the available rooms, sets the bitmap for the available rooms, clears the bitmap when the guest Checks out.

Clearing memory

```
//clearing Checkout list
void clearCheckOutList()
{
    ListIterator<guest*> *checkoutListIterator = new
ListIterator<guest*>(checkOutList);
    guest *g = new guest();
    int count = 0;
    //deep deleting guest from destructor
    while(!checkoutListIterator->IsDone())
    {
        g = checkoutListIterator->Item();
        delete g;
        count++;
        checkoutListIterator->Next();
    }
    //removing guest object pointer from the checkout list
    while(count != 0)
    {
        checkOutList->RemoveFront();
        count--;
    }

}

//clearing discarded list
void clearDiscardedList()
{
    //list iterator for discarded list
    ListIterator<guest*> *discardedListIterator = new
ListIterator<guest*>(discardedList);
    guest *g = new guest();
    int count = 0;
    while(!discardedListIterator->IsDone())
    {
```

```
        g = discardedListIterator->Item();
        delete g;
        count++;
        discardedListIterator->Next();
    }

    while(count != 0)
    {
        discardedList->RemoveFront();
        count--;
    }
}
//deleting bitmap from heap
void clearBitMapHeap()
{
    for(int i=1; i <= 11;i++)
    {
        delete rooms[i];
    }
}
```

- The above functions delete the guest object pointers present in checkout and discarded list and the bitmap pointers.

Testing

How to run the Test

1. Copy the nachos folder from your local machine to the server.
2. Copy the header file guest.h and class file guest.cc to the location /nachos/code/threads/
3. Copy the modified Threadtest.cc from the local machine to the server to the location /nachos/code/threads/
4. Navigate to the directory nachos/code/build.linux
5. Execute the below commands
    a. make clean
    b. make depend
    c. make nachos
    d. ./nachos -K

Added Files:

guest.cc and guest.h to the location /nachos/code/threads/

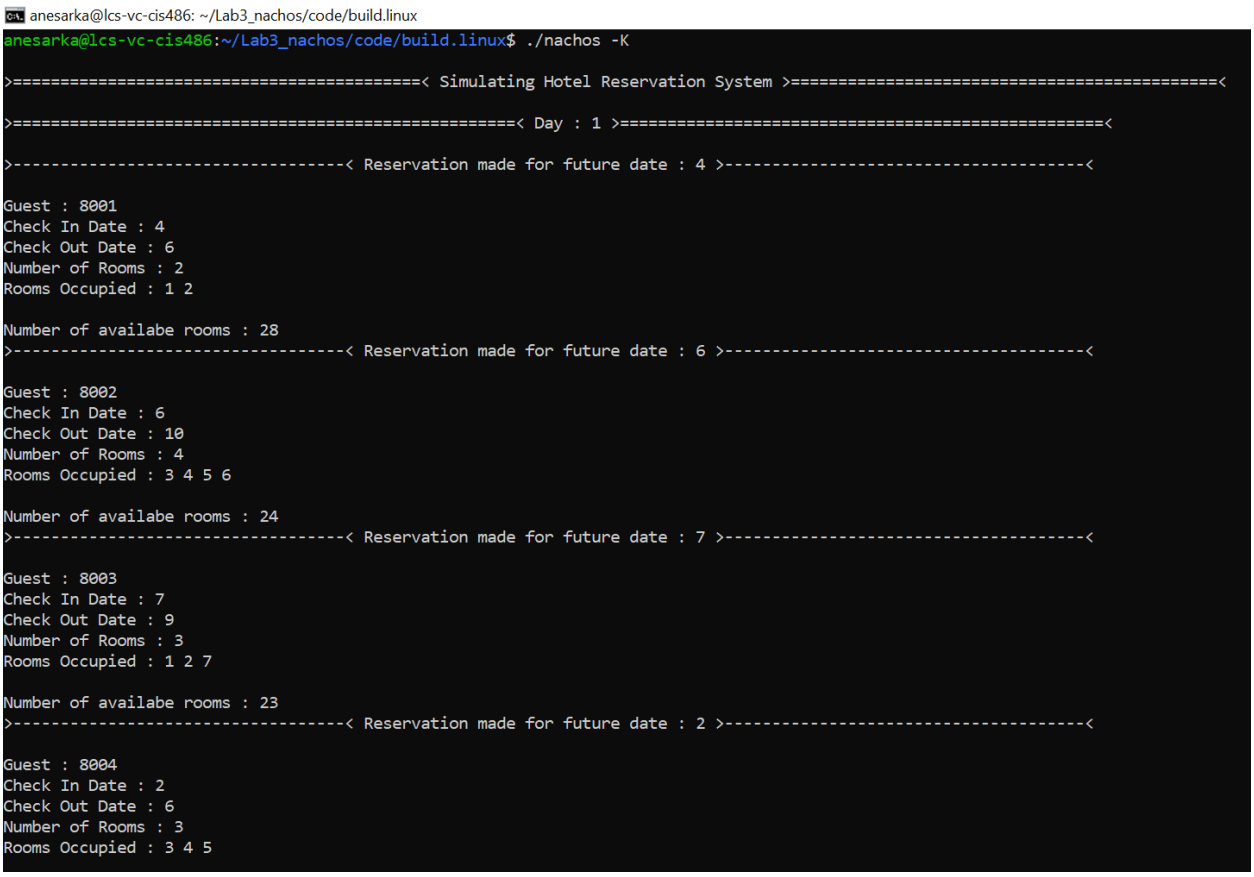
Modified Files:

Threadtest.cc in the location /nachos/code.threads/

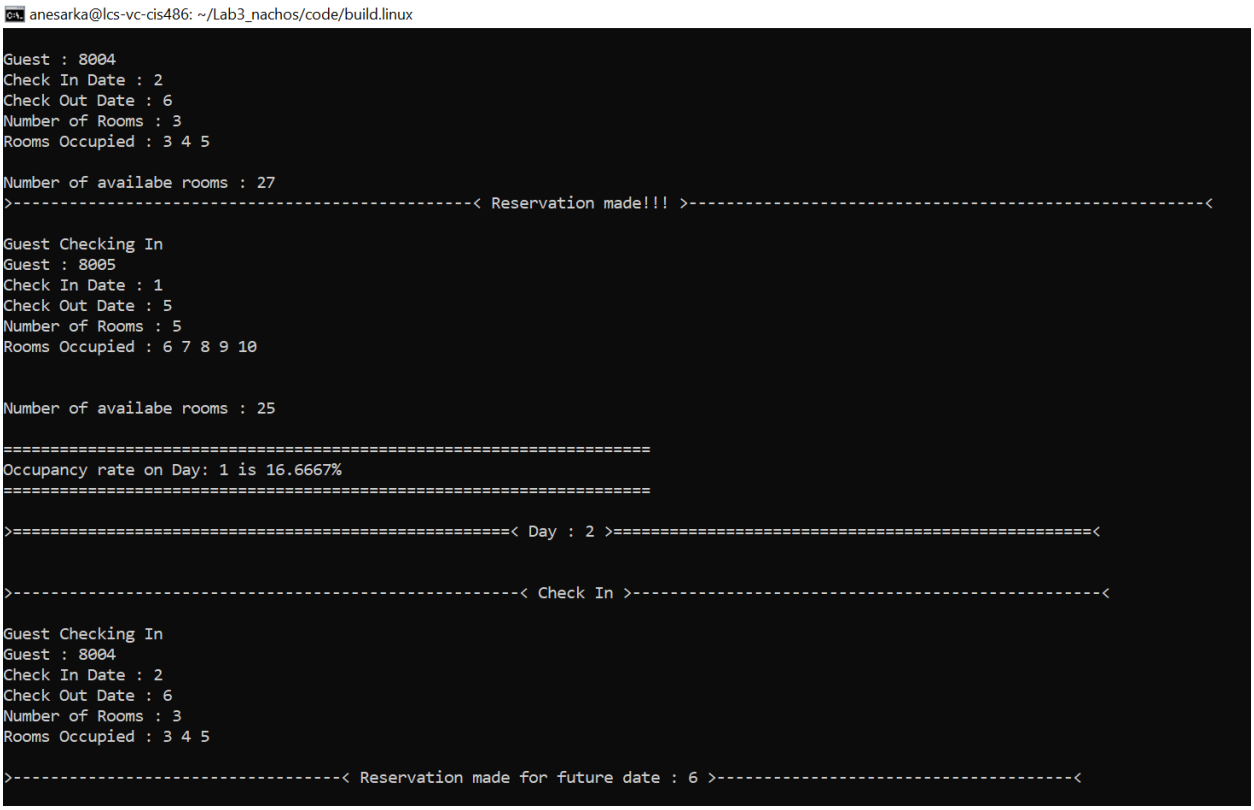
Makefile

```
THREAD_H = ../threads/alarm.h\
        ../threads/kernel.h\
        ../threads/main.h\
        ../threads/scheduler.h\
        ../threads/switch.h\
        ../threads/synch.h\
        ../threads/synchlist.h\
        ../threads/thread.h\
        ../threads/guest.h

THREAD_C = ../threads/alarm.cc\
        ../threads/kernel.cc\
        ../threads/main.cc\
        ../threads/scheduler.cc\
        ../threads/synch.cc\
        ../threads/synchlist.cc\
        ../threads/thread.cc\
        ../threads/threadtest.cc\
        ../threads/guest.cc

THREAD_O = alarm.o kernel.o main.o scheduler.o synch.o thread.o threadtest.o
guest.o
```

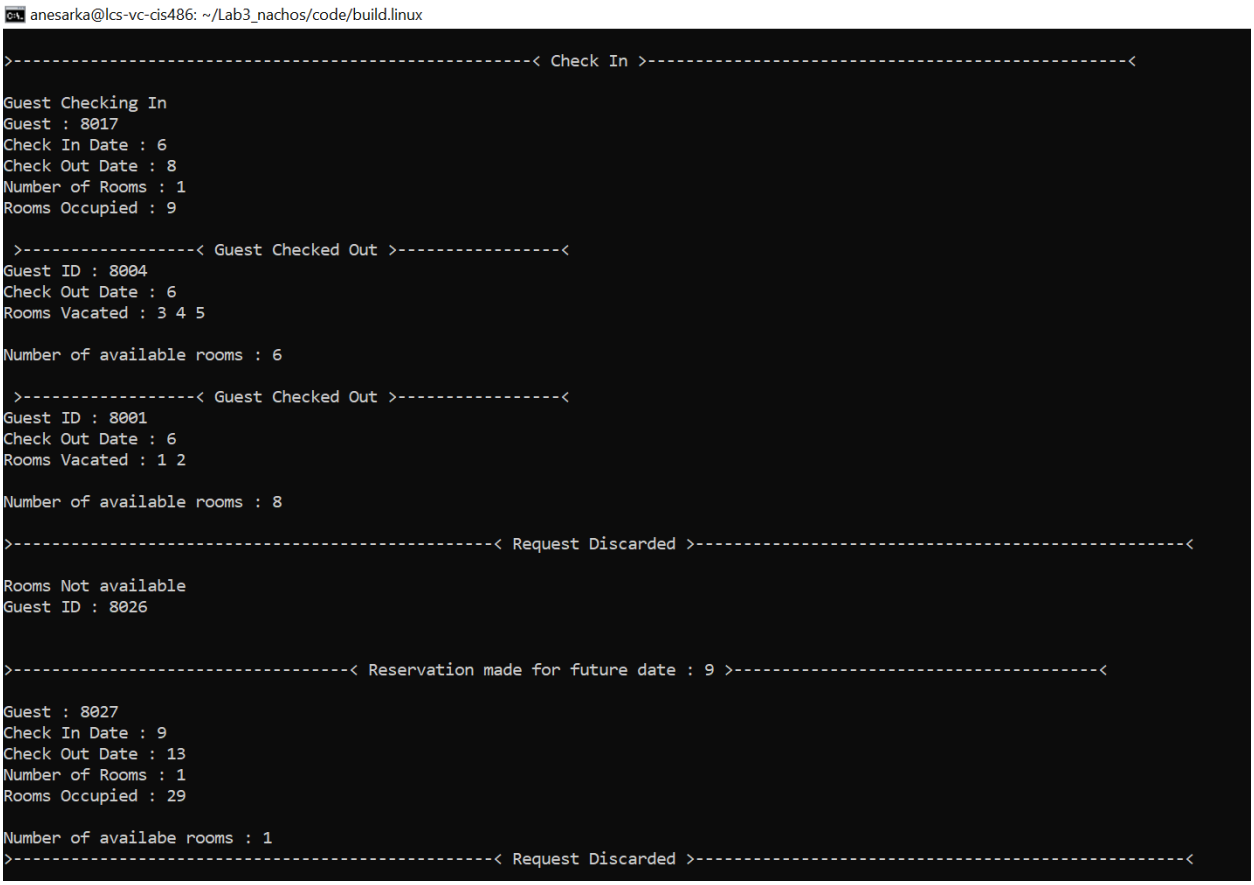
OUTPUT



- The above output displays Day 1 information
- It displays reservation made for future date.



- The above output shows CheckIn process when current date is equal to Check In date.(Guest ID 8005).
- It also shows check In process of reserved guests. (Guest ID – 8004)
- It also shows occupancy rate on Day 1. i.e. 16.67%



- The above output shows CheckOut guest information. (guest ID – 8004)
- It also shows discarded guest. (guest ID – 8026)

anesarka@lcs-vc-cis486: ~/Lab3_nachos/code/build.linux

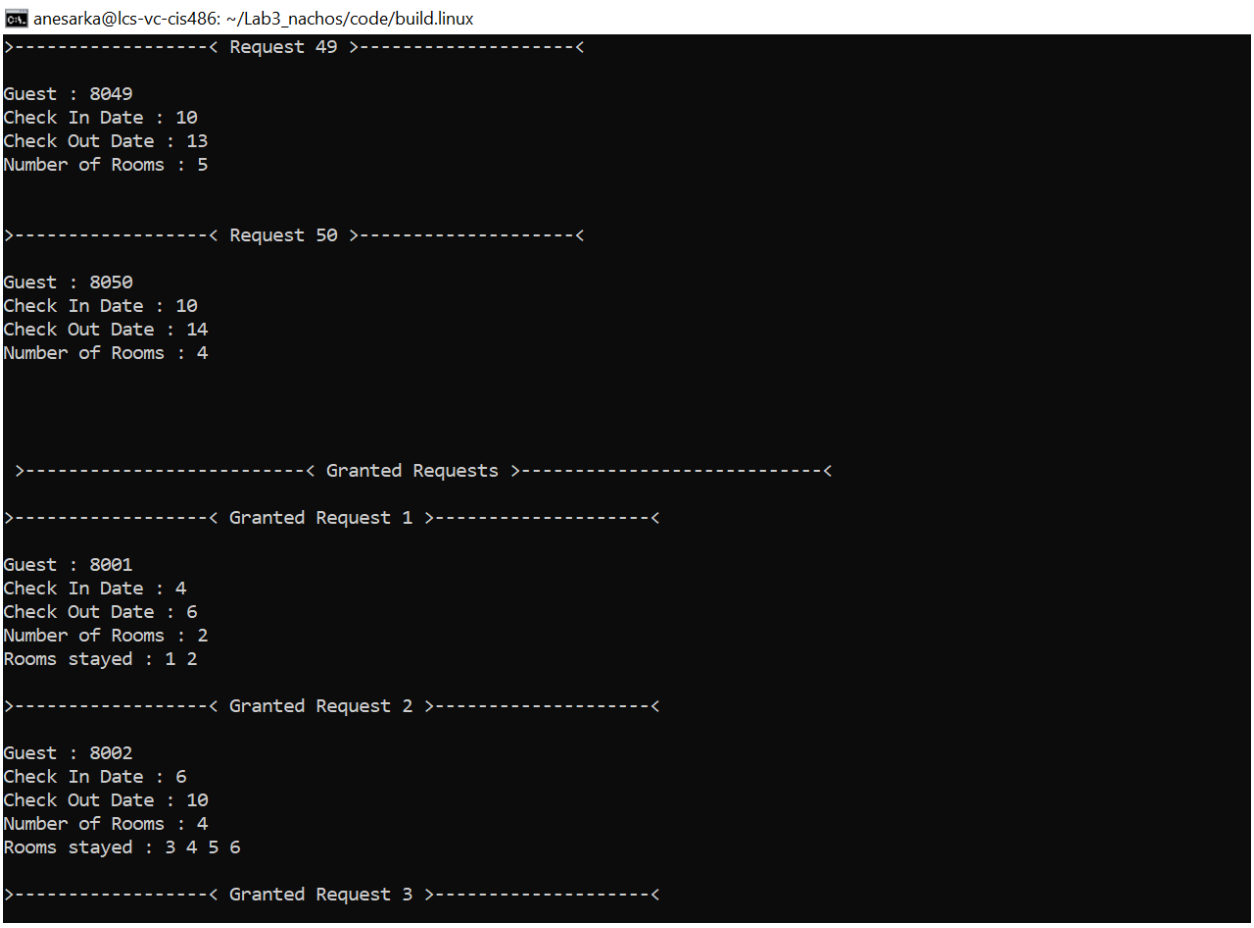
```
>========================================================= Day : 11 >=================================================================<
 >------------------< Guest Checked Out >-----------------<
Guest ID : 8007
Check Out Date : 13
Rooms Vacated : 9 10

Number of available rooms : 6

 >------------------< Guest Checked Out >-----------------<
Guest ID : 8027
Check Out Date : 13
Rooms Vacated : 29

Number of available rooms : 7

 >------------------< Guest Checked Out >-----------------<
Guest ID : 8008
Check Out Date : 12
Rooms Vacated : 1 2 7

Number of available rooms : 10

 >------------------< Guest Checked Out >-----------------<
Guest ID : 8009
Check Out Date : 13
Rooms Vacated : 8

Number of available rooms : 11

 >------------------< Guest Checked Out >-----------------<
Guest ID : 8024
Check Out Date : 14
Rooms Vacated : 14 15 16 17 18

Number of available rooms : 16
```

- The above output shows all Guests checking out on Day 11.

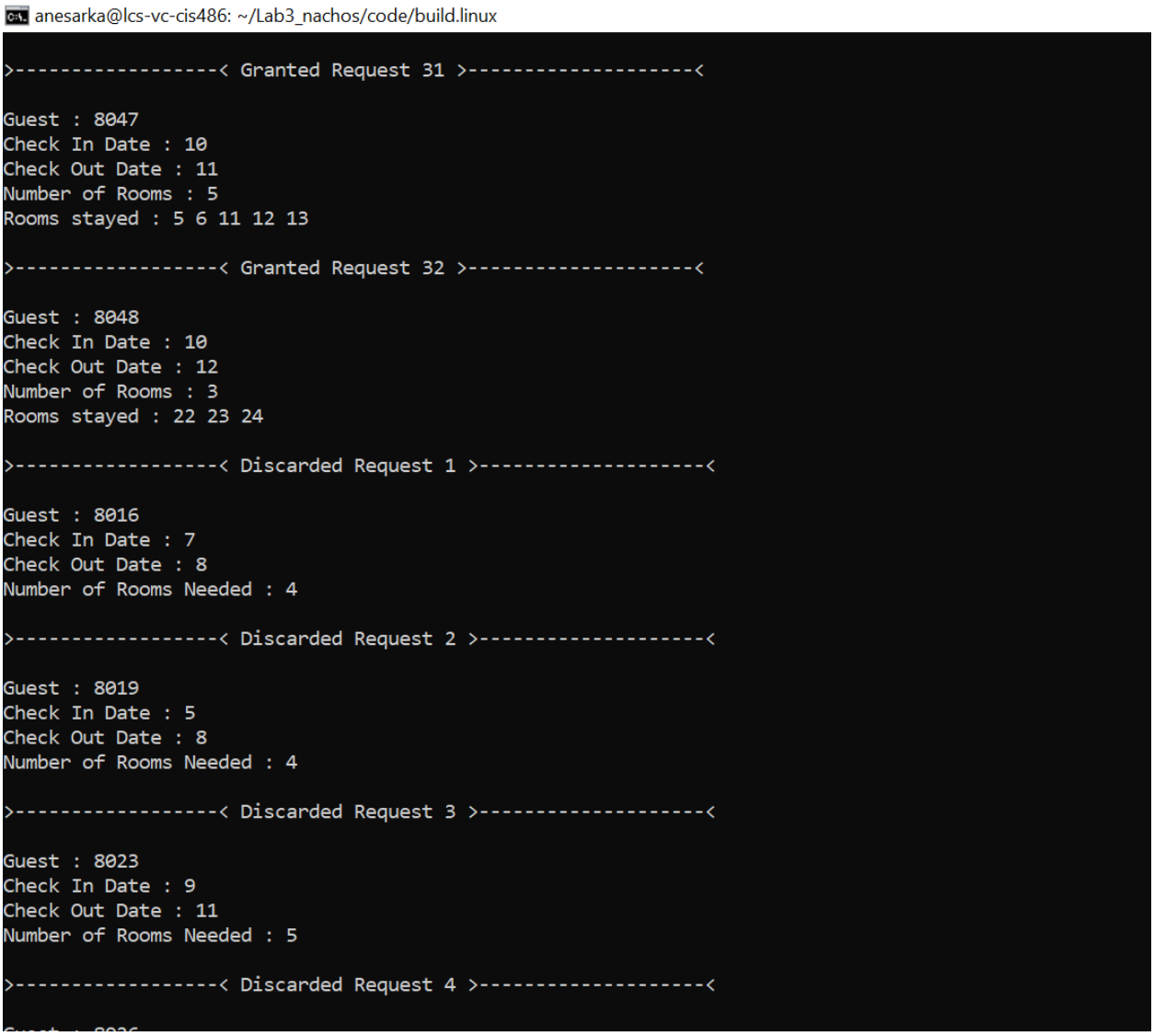
anesarka@lcs-vc-cis486: ~/Lab3_nachos/code/build.linux

```
>================================================< Summary of people who stayed >================================================<

 >-------------------------< Total requests Generated >--------------------------<

>-----------------< Request 1 >--------------------<

Guest : 8001
Check In Date : 4
Check Out Date : 6
Number of Rooms : 2
Rooms stayed : 1 2

>-----------------< Request 2 >--------------------<

Guest : 8002
Check In Date : 6
Check Out Date : 10
Number of Rooms : 4
Rooms stayed : 3 4 5 6

>-----------------< Request 3 >--------------------<

Guest : 8003
Check In Date : 7
Check Out Date : 9
Number of Rooms : 3
Rooms stayed : 1 2 7

>-----------------< Request 4 >--------------------<

Guest : 8004
Check In Date : 2
Check Out Date : 6
Number of Rooms : 3
Rooms stayed : 3 4 5
```

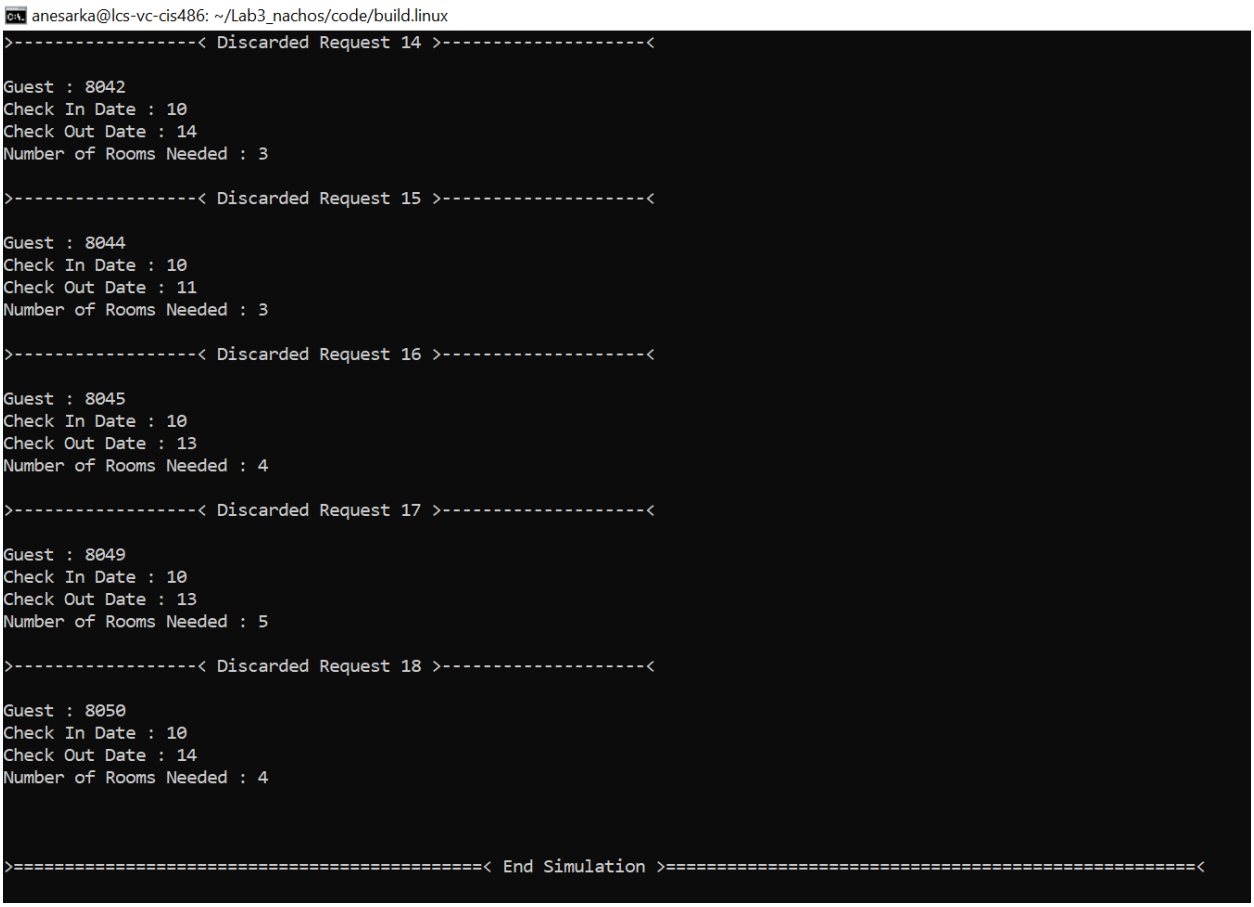
- The above output shows summary of guests who stayed in the hotel.
- There are 50 guests requests generated for 10 days.



- The above output shows the granted requests sequentially.



- The above output shows there were 32 granted request.
- It also shows discarded requests.



- The above output shows that there were 18 discarded guest requests.
- This is End of Simulation.