CSE445/598 Assignment 2 Requirement Document – 50 Points
Spring 2024

Assignment 2 due: Sat., February 10, 2024, by 11:59pm (Arizona Time), plus a one-day grace period.

Introduction

The purpose of this assignment is to make sure that you understand and are familiar with the concepts covered in the lectures, including distributed computing, multithreading, thread definition, creation, management, synchronization, cooperation, event-driven programming, client and server architecture, service execution model of creating a new thread for each request, the performance of parallel computing, and the impact of multi-core processors to multithreading programs with complex coordination and cooperation. Furthermore, you can apply these concepts in a programming assignment.

Assignment 2 is individual assignment. Each student must perform and submit independent work. No cooperation is allowed in completing the individual assignments.

Not for this assignment, but for the following assignments 3, 4, 6 and 7, a formal team-building process is required and a document is given separately. We need to know the teams in advance before we can start assignment 3, because we need to create a server site for each team.

Section I Preparation and Practice Exercises (No submission required)

No submission is required for this section of exercises. However, doing these exercises can help you better understand the concepts and thus help you in quizzes, exams, as well as the assignment questions.

1. Reading: Textbook Chapter 2.

2. Answer the multiple-choice questions in text section 2.8. Studying the material covered in these questions can help you prepare for the lecture exercises, quizzes, and the exams.

3. Study for questions 2 through 20 in text section 2.8. Make sure that you understand these questions and can briefly answer these questions. Study the material covered in these questions can help you prepare for the exams and understand the homework assignment.

4. Test the programs given in questions 24 and 25 in text section 2.8. Identify the problems in the program and give correct versions of the programs.

5. If you want solve a more challenging problem in multithreading, you can do question 26 in text section 2.8.

6. **Tutorial**. To help you complete the assignment in Section II, you may want go through the tutorial given in the textbook chapter 2, which consists of

   6.1 Reading the case study in text section 2.6.3.

6.2 Implementing and testing the program given in the case study. The program can be used as the starting point for your assignment in Section II.

6.3 Extending the program based on the requirement in Section II.

7. For debugging multithreading programs, please also read:

https://docs.microsoft.com/en-us/visualstudio/debugger/debug-multithreaded-applications-in-visual-studio?view=vs-2019

Section II Assignment (Submission required)

Warning: This is a long programming assignment designed for a study load for two weeks of estimated 2*8 = 16 hours. It is challenging at both the conceptual level and the implementation level. It requires solid object-oriented computing concepts and the detailed materials covered in the lectures. The assignment document is long, and you need to spend time to read it carefully. You must distribute the load in the given two weeks. You will not have enough time to complete the assignment if you start the assignment only in the last week before the assignment is due.

Purpose of this assignment is to exercise the concepts learned in this chapter. It is not the purpose of this assignment to create realistic services and applications. We will create more realistic services, and applications in the subsequent assignments. In this assignment, you must use a C# console application to implement the requirements, and the program will be graded using **Gradescope**. You must follow the instructions and the given template precisely.

Description: Many hotels have unsold rooms. You decide to create an e-business that better makes use of the unsold hotel rooms: A new hotel room block booking system that involves travel agents and hotels. The system consists of multiple travel agents (clients) and multiple hotels (servers). The travel agents can buy in quantity (block) of hotel rooms from the hotels with lower (discounted) prices, and then resell the tickets to their customers (end users) at competitive prices. A simplified workflow and the major components of the system are shown in the diagram in Figure 1.

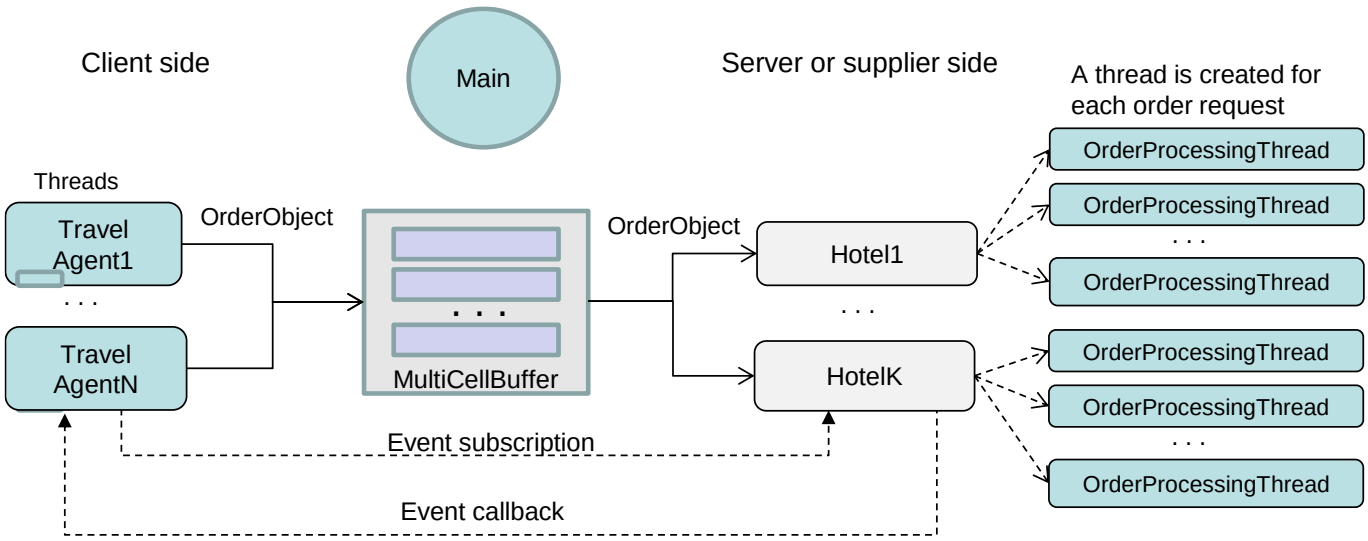

Figure 1 Simplified workflow of a new hotel room booking system in multithreading system

Figure 2 shows the architecture and the components of the program to be developed. In the diagram, solid lines show the class object instantiation (creating objects) and the dashed lines are function calls.

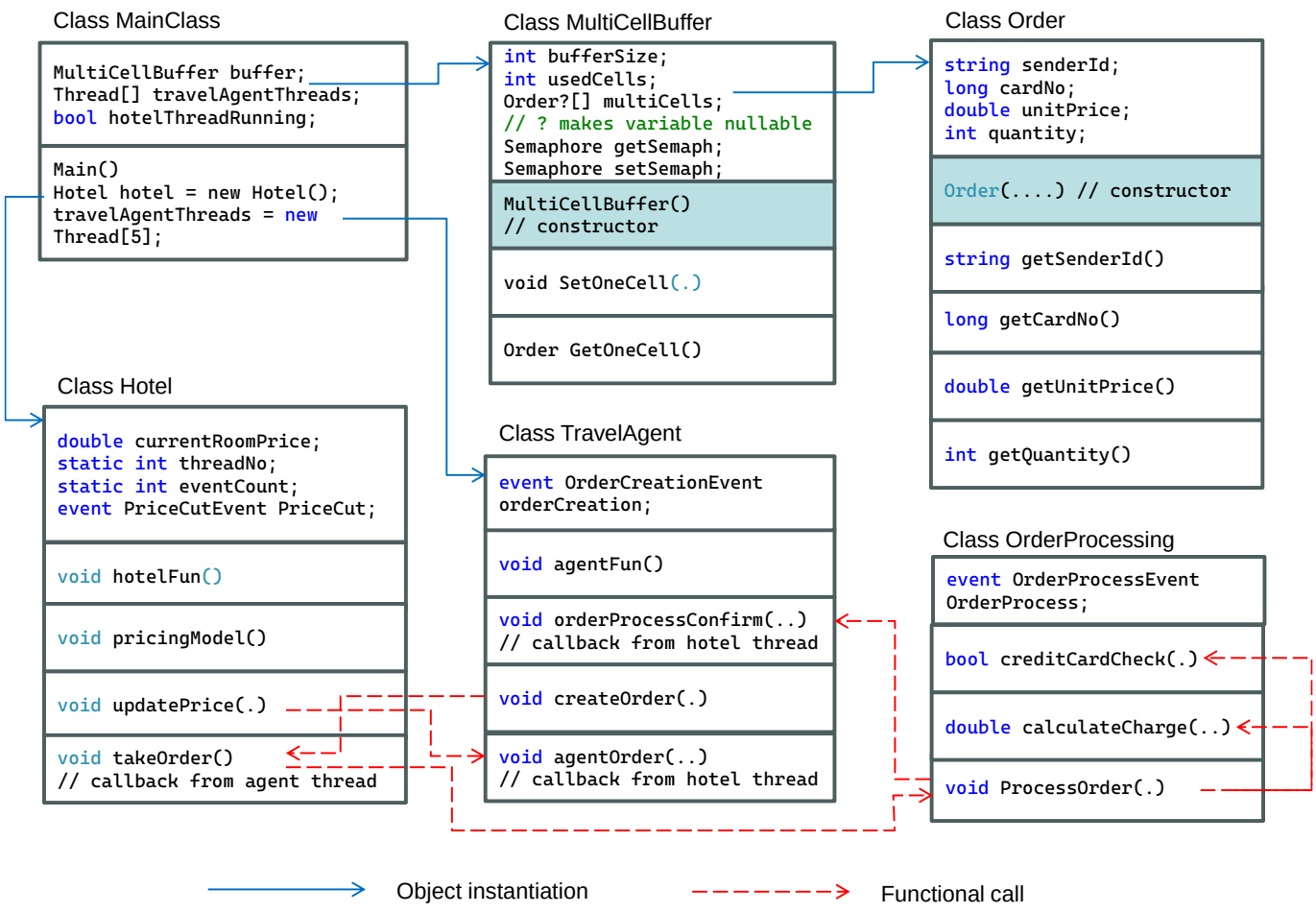

Figure 2. The architecture and the components of the program to be developed

Figure 3 illustrates thread creations, events, event handler, event subscription, function calls and event handler callbacks in the architecture.

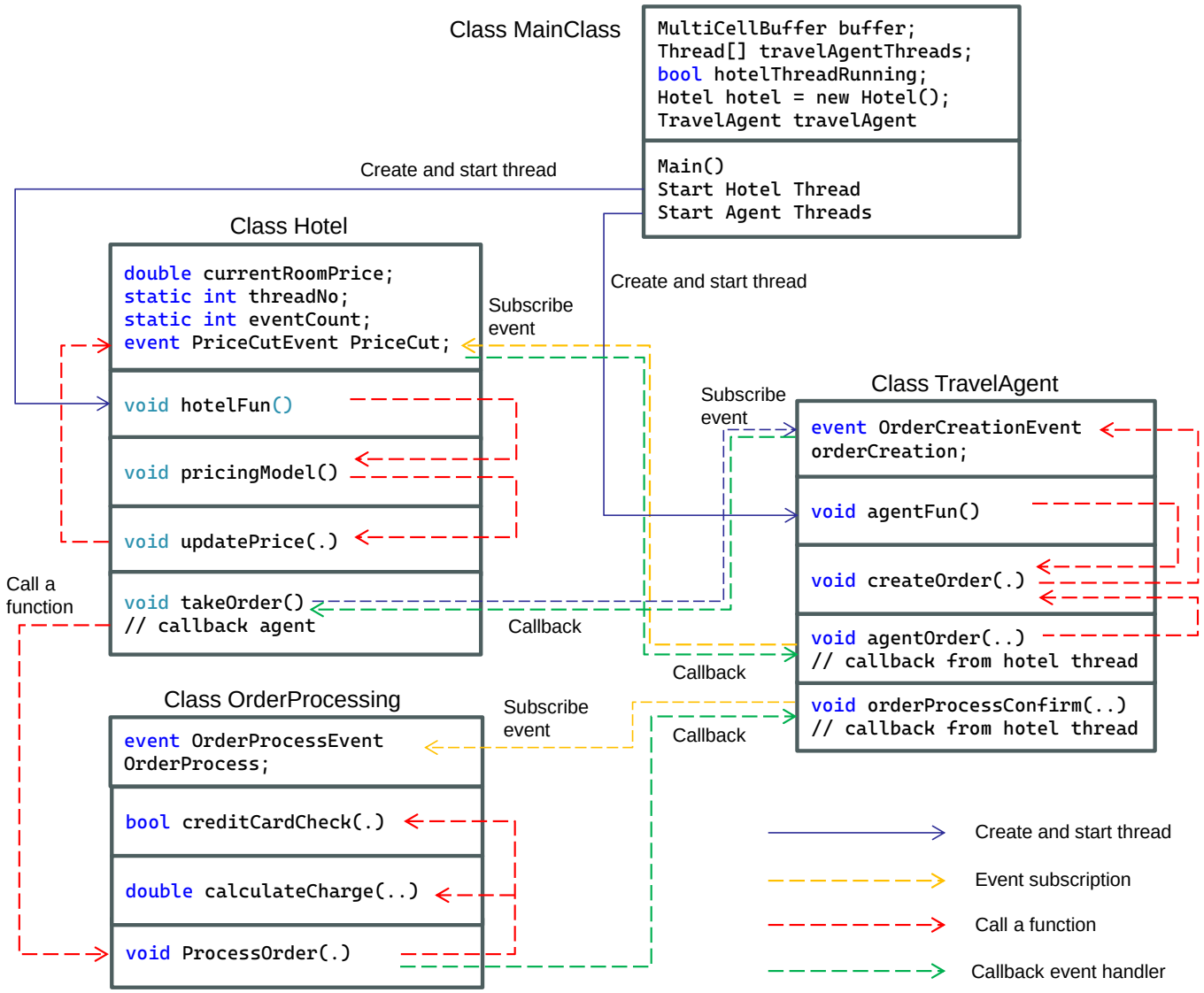

Figure 3. Threads, events, event handler, event subscription, function calls and event handler callbacks

The signatures of the classes, delegates, events, properties, and functions are given as follows. You must use all the classes, events, variables and functions listed. However, you can add more variables and functions.

```
using System;
using System.Collections.Generic;
using System.Threading;
namespace ConsoleApp1
{
   //delegate declaration for creating events
   public delegate void PriceCutEvent(double roomPrice, Thread agentThread);
   public delegate void OrderProcessEvent(Order order, double orderAmount);
   public delegate void OrderCreationEvent();

   public class MainClass
   {
```

```
    public static MultiCellBuffer buffer;
    public static Thread[] travelAgentThreads;
    public static bool hotelThreadRunning = true;
    public static void Main(string[] args)
    {
       Console.WriteLine("Inside Main");
       buffer = new MultiCellBuffer();
       Hotel hotel = new Hotel();
       TravelAgent travelAgent = new TravelAgent();
       Thread hotelThread = new Thread(new ThreadStart(hotel.hotelFun));
       hotelThread.Start();
       Hotel.PriceCut += new PriceCutEvent(travelAgent.agentOrder);
       Console.WriteLine("Price cut event has been subscribed");
       TravelAgent.orderCreation += new OrderCreationEvent(hotel.takeOrder);
       Console.WriteLine("Order creation event has been subscribed");
       OrderProcessing.OrderProcess += new OrderProcessEvent(travelAgent.orderProcessConfirm);
       Console.WriteLine("Order process event has been subscribed");
       travelAgentThreads = new Thread[5];
       for (int i = 0; i < 5; i++)
       {
          Console.WriteLine("Creating  travel agent thread {0}", (i + 1));
          travelAgentThreads[i] = new Thread(travelAgent.agentFun);
          travelAgentThreads[i].Name = (i + 1).ToString();
          travelAgentThreads[i].Start();
       }
    }
}
public class MultiCellBuffer
{
    // Each cell can contain an order object
    private const int bufferSize = 3; //buffer size
    int usedCells;
    private Order[] multiCells;
    public static Semaphore getSemaph;
    public static Semaphore setSemaph;

    public MultiCellBuffer() //constructor
    {
       // add your implementation here
    }
    public void SetOneCell(Order data)
    {
       // add your implementation here
    }
    public Order GetOneCell()
    {
       // add your implementation here
    }
}
```

```
public class Order
{
   //identity of sender of order
   private string senderId;
   //credit card number
   private long cardNo;
   //unit price of room from hotel
   private double unitPrice;
   //quantity of rooms to order
   private int quantity;

   //parametrized constructor
   public Order(string senderId, long cardNo, double unitPrice, int quantity)
   {
      // add your implementation here
   }
   //get methods
   public string getSenderId()
   {
      // add your implementation here
   }
   public long getCardNo()
   {
      // add your implementation here
   }
   public double getUnitPrice()
   {
      // add your implementation here
   }
   public int getQuantity()
   {
      // add your implementation here
   }
}
public class OrderProcessing
{
   public static event OrderProcessEvent OrderProcess;
   //method to check for valid credit card number input
   public static bool creditCardCheck(long creditCardNumber)
   {
      // add your implementation here
   }
   //method to calculate the final charge after adding taxes, location charges, etc
   public static double calculateCharge(double unitPrice, int quantity)
   {
      // add your implementation here
   }
   //method to process the order
   public static void ProcessOrder(Order order)
```

```
    {
        // add your implementation here
    }
}
public class TravelAgent
{
    public static event OrderCreationEvent orderCreation;

    public void agentFun()
    {
        // add your implementation here
    }
    public void orderProcessConfirm(Order order, double orderAmount)
    {
        // add your implementation here
    }
    private void createOrder(string senderId)
    {
        // add your implementation here
    }
    public void agentOrder(double roomPrice, Thread travelAgent) // Callback from hotel thread
    {
        // add your implementation here
    }
}
public class Hotel
{
    static double currentRoomPrice = 100; //random current agent price
    static int threadNo = 0;
    static int eventCount = 0;
    public static event PriceCutEvent PriceCut;
    public void hotelFun()
    {
        // add your implementation here
    }
    //using random method to generate random room prices
    public double pricingModel()
    {
        // add your implementation here
    }
    public void updatePrice(double newRoomPrice)
    {
        // add your implementation here
    }
    public void takeOrder() // callback from travel agent
    {
        // add your implementation here
    }
}
```

```
}
```

A part of a sample output is given as follows.

```
Inside Main
Price cut event has been subscribed
Order creation event has been subscribed
Order process event has been subscribed
Creating  travel agent thread 1
Creating  travel agent thread 2
Starting travel agent now
Creating  travel agent thread 3
Starting travel agent now
Creating  travel agent thread 4
Starting travel agent now
Creating  travel agent thread 5
Starting travel agent now
Starting travel agent now
New price is 132
New price is 83
Updating the price and calling price cut event
Incoming order for room with price 83
Inside create order
Setting in buffer cell
Exit setting in buffer
Exit reading buffer
Travel Agent 1's order is confirmed. The amount to be charged is $7031
New price is 119
New price is 76
Updating the price and calling price cut event
Incoming order for room with price 76
Inside create order
Setting in buffer cell
Exit setting in buffer
Exit reading buffer
Travel Agent 2's order is confirmed. The amount to be charged is $7320
Inside create order
Setting in buffer cell
Exit setting in buffer
Exit reading buffer
Travel Agent 2's order is confirmed. The amount to be charged is $17434
Inside create order
Setting in buffer cell
Exit setting in buffer
Exit reading buffer
Travel Agent 5's order is confirmed. The amount to be charged is $23866
Inside create order
…
```

In this assignment, you will simulate both clients and servers in one system using multithreading and event-driven programming.

An **Operation Scenario** of the hotel room block booking system is outlined as follows:

(1) A **Hotel** uses a pricing model to calculate dynamically the hotel room price for the travel agents. The prices can go up (when room occupancy rate is high) and down (when room occupancy rate is low) from time to time. If the new price is lower than the previous price, it emits a (promotional) event and calls the event handlers in the travel agents (clients) that have subscribed to the event.

(2) A **TravelAgent** evaluates the needs based on the new price and other factors, generates an Order object (consisting of multiple values), and sends the order to the hotel through a MultiCellBuffer.
(3) The **TravelAgent** sends the Order object to the promoting hotel through one of the free cells in the **MultiCellBuffer**.
(4) The **Hotel** receives the Order object from the MultiCellBuffer.
(5) The Hotel creates a new thread, an OrderProcessingThread, to process the order;
(6) The **OrderProcessingThread** processes the order, e.g., checks the credit card number and the maximum number allowed to purchase, etc., and calculates the total amount.
(7) The OrderProcessingThread sends a confirmation to the travel agent through a callback and prints the order information (on screen). You will submit the output screenshots in a PDF or Word file. A part of a sample output is given in in this document.

Components and **tasks** implementing the components in the diagram in Figures 1, 2 and 3 are explained in details as follows, with their grading scores (points) allocation.

Assignment Tasks

1. **Hotel*1*** through **Hotel*K**** are the objects of the Hotel class on the server side. The Hotel class has a method hotelFun, which will be started as a thread by the Main method, and it will perform a number of functions. It uses a PricingModel to determine the block room prices. It defines a price-cut event that can emit an event and call the event handlers in the Hotel objects if there is a price-cut according to the PricingModel. It receives the orders from the MultiCellBuffer. For each order, you must start a new thread (resulting in multiple threads for processing multiple orders) from OrderProcessing class to process the order based on the current price. There is a counter in the Hotel. After t (e.g., counter = 10) price cuts have been made, the Hotel thread will terminate. The travel agents do not have to make an order after each price cut, but must make at least one order within the entire iterations of each test. [5 points]

   In this assignment, you set the number of hotels K = 1.

2. **PricingModel**: It is a method. It decides the price of rooms, with a base price of $100. It can increase or decrease the price. You must define a mathematical model (formula) or use a random function that generates the price in the range [80, 160]. You must make sure that your model will allow the price to go up sometimes and go down other times within your iterations of each test. [5 points]

3. **OrderProcessing** is a class on the server's side. Whenever an order needs to be processed, a new thread is instantiated from this class to process the order. It will check the validity of the credit card number. You can define your credit card format, for example, the credit card numbers from the travel agents must be a number registered to the Hotel, or a number between two given numbers (e.g., between 5000 and 7000). Each OrderProcessing thread will calculate the total amount of charge, e.g., unitPrice*NoOfTicket + Tax + LocationCharge. The tax and location charges must be represented as random values within predetermined ranges. For example, tax should be between 8% and 12%, and location charge should be $20 and $80. [5 points]

4. **Agent*1*** through **Agent*N***. You must use a variable N, but you can set N = 5 in your implementation. Each travel agent is a thread created by a different object instantiated from the same class. The travel agent's actions are event-driven. Each travel agent contains a callback method (event handler) for the hotels to call when a price-cut event occurs. The event handler will write the new price (reduced price) into a class variable that can be read by the travel agent thread. The travel agent will read the new price and calculate the number of rooms to order, for example, based on the need and the difference between the previous price and the current price. Each order is an Order class object. Then, the travel agent will

send the order to the MultiCellBuffer. The thread will terminate after the Hotel threads have terminated. [5 points]

5. **Order** is a class that contains the following private data members:
   - senderId: the identity of the sender, you can use thread name or thread id.
   - cardNo: an integer that represents a credit card number.
   - receiverID: the identity of the receiver, you can use thread name or a unique name defined for an hotel. This field is optional.
   - quantity: an integer that represents the number of rooms to order.
   - unit price: a double that represents the unit price of the room received from the hotel.

   You must use public methods to set and get the private data members. You must decide if these methods need to be synchronized. The instances (objects) created from this class are of the Order object. [10 points]

6. **MultiCellBuffer** is a class that is used for the communication between the travel agents (clients) and the hotels (servers): MultiCellBuffer is an array of objects, and each array element (cell) consists of a **reference** to an **Order** object. To write data into and to read data from one of the available cells, setOneCell and getOneCell methods must be defined. You must use a semaphore with an initial value n (n = 3) to manage the available cells for write and a semaphore with an initial value 0 to manage the available cells for read.

6.1 Using setOneCell, the travel agent can write the MultiCellBuffer. If the semaphore value is 0, no attempt should be made to write the buffer at all. If the semaphore > 0, setOneCell must use lock method to obtain write permission. A cell can be used as long as it is available (unlocked) and the cell is empty (null). The semaphores allow a travel agent and hotel to see the availability of the cells, while the lock mechanism allows a travel agent to gain the right to write into one of the buffer cells.

6.2 For getOneCell, the hotels can read buffer cells. If the semaphore value is 3, no attempt should be made to read the buffer at all. If the semaphore < 3, getOneCell must use Monitor.Enter method to obtain read permission. A cell can be used as long as it is available (unlocked) and is not empty (null). The semaphore allows a hotel to see the availability of the cells, while the lock mechanism allows a travel agent to gain the right to read a cell.

If your message generation is slow, e.g., you can use sleep function with a random sleeping time in the hotel threads and in the agent threads. You need to choose proper sleeping time to avoid that the buffer cells are idle most of the time and the first cell is always used and the rest of the cells are never used. In this case, you need to reduce the sleep time in the hotel and agent threads, and/or to add a sleep time in the buffer operation to make the cell to be locked longer, so that all cells have a chance for being used and the usage being reflected in the outputs/printouts. [10 points]

8. **Main**: The Main thread will perform necessary preparation, create the buffer classes, instantiate the objects, create threads, and start threads. [10 points]

Notes:

1. You must follow what is defined in the assignment document. You have flexibility to choose your implementation details if they are not explicitly specified in the document. If you are not sure on any issue, ask the instructor or the TA by posting the question on the course discussion board.

2. The program and each component of the program must be well commented.

3. In this assignment, you may not need to have external input, but you must set your internal inputs in such a way that the program is properly tested, for example, each client must have completed at least one ordering process before the server terminates.

4. The submitted code **must** be named "submission.cs"

5. The submitted code **must** belong to the namespace: "ConsoleApp1"

Submission Requirement and Grading Rubric

The assignment will be graded using Gradescope. Please check the grading rubric in a separate document and follow the Gradescope submission instruction.

Late submission deduction policy:

- Grace period (Sunday): No penalty for late submissions that are received within 24 hours of the given due date.
- 1% grade deduction for every hour after the first 24 hours of the grace period (from Monday through Tuesday!
- No submission will be allowed after Tuesday midnight. The submission link will be disabled at 11:59pm on Tuesday. You must make sure that you complete the submission before 11:59pm. If your file is big, it may take more than an hour to complete the submission!