第二次作业

姓名：杲时雨
学号：202333177

一、开发环境

IDE：Visual Studio 2022
编程语言：C#
操作系统：Windows11

二、需求分解

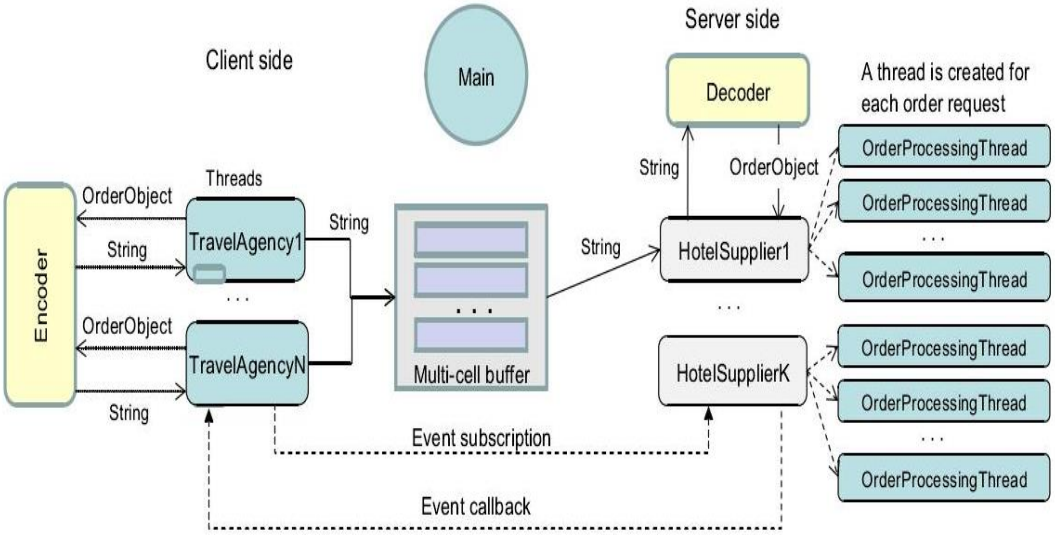


Figure 1 Architecture of a hotel block booking system

酒店整体预订系统的操作场景概述如下：

（1） HotelSupplier 使用定价模型来计算房价。如果新价格低于之前的价格，它会发出（促销）活动，并致电已订阅该活动的旅行社的活动处理人员。

（2） TravelAgency 评估价格，生成 OrderObject（由多个值组成），并将订单发送到编码器，将订单对象转换为纯字符串。

（3） 编码器将对象转换为字符串。

（4） 编码器将编码后的字符串发送回调用者。

（5） TravelAgency 将编码字符串发送到 MultiCellBuffer 中的一个空闲单元格。

（6） HotelSupplier 从 MultiCellBuffer 接收编码字符串，并将字符串发送到解码器进行解码。

（7） 解码器将 OrderObject 发送给 HotelSupplier。解码对象必须包含 TravelAgency 生成的相同值。

（8） HotelSupplier 创建一个新线程来处理订单；

（9） OrderProcessingThread 处理订单，例如检查信用卡号并计算金额。

（10） OrderProcessingThread 向旅行社发送确认并打印订单（在屏幕上）。

三、程序相关代码

1、HotelSupplier（酒店供应商类）

HotelSupplier 使用定价模型来计算房价。如果新价格低于之前的价格，它会发出（促销）活动，旅行社会自己去决定创建订单。

```
1. namespace HotelBookingSystem
2. {
3.     public class HotelSupplier
4.     {
5.         public delegate void PriceCutEventHandler(int newPrice);  // 新建委托，用于传递新价格给旅行社线程
6.         public event PriceCutEventHandler PriceCutEvent;  // 降价事件
7.         private int currentPrice = 500;
8.         private int priceCutCount = 0;
9.         private static readonly object priceLock = new object();
10.        private ArrayList creditCardIDList = new ArrayList();
11.        private int lastCrefitRoomCount = 0;  // 上次所有旅行社订房的数量
12.
13.        public void StartHotelSupplier()
14.        {
15.            while (priceCutCount < 10)
16.            {
17.                int newPrice = PricingModel();
18.                Console.WriteLine($"newPrice: {newPrice}");
19.                if (newPrice < currentPrice)
20.                {
21.                    Console.WriteLine($"开始{priceCutCount+1}次降价处理");
22.                    PriceCutEvent?.Invoke(newPrice);
23.                    priceCutCount++;
24.                    this.ProcessingAllOrder(newPrice);
25.                }
26.                currentPrice = newPrice;
27.
28.                Thread.Sleep(500);
29.            }
30.        }
31.
32.        /**
33.         * 它将检查信用卡号的有效性。您可以定义您的信用卡格式，例如，旅行社的信用卡号必须是向 HotelSupplier 注册的号码
```

```
34.          */
35.         public bool RegCreditCardID(string creditCardID)
36.         {
37.             if (creditCardID.Length != 16)
38.             {
39.                 Console.WriteLine($"卡号:{creditCardID}格式错误，请检查注册信息!");
40.                 return false;
41.             }
42.             if (!creditCardIDList.Contains(creditCardID))
43.             {
44.                 string frontFourNum = creditCardID.Substring(0, 4);
45.                 if (Convert.ToInt32(frontFourNum) > 5000 && Convert.ToInt32(frontFourNum) < 7000)
46.                 {
47.                     creditCardIDList.Add(creditCardID);
48.                     Console.WriteLine($"卡号:{creditCardID}成功注册!");
49.                 }
50.                 else
51.                 {
52.                     Console.WriteLine($"卡号:{creditCardID}不在可注册范围内!");
53.                     return false;
54.                 }
55.
56.             }
57.             return true;
58.         }
59.     }
60.
61.}
```

2、PricingModel（定价模型）

定价模型来确定价格，根据在给定时间段内收到的订单和 HotelSupplier 在同一时间段内可用的房间数量来确定价格。

```
1.         /**
2.          * 定价模型：
3.          * 您必须定义一个数学模型（随机函数很好）
4.          * 以根据在给定时间段内收到的订单和 HotelSupplier 在同一时间段内可用的房间数量来确定价格。
5.          */
6.         private int PricingModel()
```

```
        {
            // 随机函数来模拟价格变化
            Random rand = new Random();

            if (lastCrefitRoomCount > 5)
            {
                return rand.Next(500, 700);
            }
            else
            {
                return rand.Next(300, 500);
            }
        }

```

3、OrderProcessing 订单处理

OrderProcessing 是供应商侧的一个类或类中的一个方法。每当需要处理订单时，都会从该类（或方法）实例化一个新线程来处理订单。它将检查信用卡号的有效性。每个 OrderProcessing 线程将计算费用总额，例如单价*房间数量。

（1）处理所有订单+注册信用卡

```
        private void ProcessingAllOrder(int lowPrice)
        {
            string encodedOrder = null;
            do
            {
                encodedOrder = MultiCellBuffer.GetOneCell();  // HotelSupplier 从 MultiCellBuffer 接收编码字符串，并将字符串发送到解码器进行解码。
                if (encodedOrder != null)
                {
                    OrderClass orderClass = Decoder.DecodeOrder(encodedOrder);  // （7） 解码器将 OrderObject 发送给 HotelSupplier。解码对象必须包含 TravelAgency 生成的相同值。
                    lastCrefitRoomCount = orderClass.Amount;
                    OrderProcessing orderProcessing = new OrderProcessing();
                    Thread orderProcessingThread = new Thread(() => orderProcessing.ProcessOrder(orderClass, lowPrice, this.creditCardIDList));
                    orderProcessingThread.Start();
                    // orderProcessing.ProcessOrder(orderClass, lowPrice);
                }
```

```
            } while (encodedOrder != null);
        }

        /**
         * 它将检查信用卡号的有效性。您可以定义您的信用卡格式，例如，旅行社的信用卡号必须是向 HotelSupplier 注册的号码
         */
        public bool RegCreditCardID(string creditCardID)
        {
            if (creditCardID.Length != 16)
            {
                Console.WriteLine($"卡号:{creditCardID}格式错误，请检查注册信息!");
                return false;
            }
            if (!creditCardIDList.Contains(creditCardID))
            {
                string frontFourNum = creditCardID.Substring(0, 4);
                if (Convert.ToInt32(frontFourNum) > 5000 && Convert.ToInt32(frontFourNum) < 7000)
                {
                    creditCardIDList.Add(creditCardID);
                    Console.WriteLine($"卡号:{creditCardID}成功注册!");
                }
                else
                {
                    Console.WriteLine($"卡号:{creditCardID}不在可注册范围内!");
                    return false;
                }

            }
            return true;
        }
```

（2）处理每一个订单

单独的 orderProcessing 类，处理每一个线程的订单任务。

```
namespace HotelBookingSystem
{
    public class OrderProcessing
    {
        public void ProcessOrder(OrderClass order, int hotelPrice, ArrayList creditCardIDList)
        {
```

```
            if (creditCardIDList.Contains(order.CardNumber))
            {
                int totalCost = order.Amount * hotelPrice;
                Console.WriteLine($"订单处理成功，发送线程 ID: {order.SenderId}\n 当前处理线程 ID: {Thread.CurrentThread.ManagedThreadId}\n" +
                    $" 旅行社名: {order.AgencyName}\n 房间数: {order.Amount}\n 总费用: {totalCost}\n 信用卡号: {order.CardNumber}\n 订单创建时间: {order.OrderTime}\n\r");
            }
            else
            {
                Console.WriteLine("信用卡未注册，订单处理失败");
            }
        }
    }

}
```

4、TravelAgency（旅行社类）

TravelAgency1 到 TravelAgencyN，每个旅行社都是从类中的同一类（或同一方法）实例化的线程。旅行社的行动是由事件驱动的。每家旅行社都包含一个回调方法（事件处理程序），供 HotelSuppliers 在发生降价事件时调用。HotelSupplier 线程终止后，线程将终止。每个订单都是一个 OrderClass 对象。对象被发送到编码器进行编码。编码后的字符串被发送回旅行社。然后，旅行社将以字符串格式将订单发送到 MultiCellBuffer。在将订单发送到 MultiCell Buffer 之前，必须保存时间戳。收到订单完成确认后，将计算并保存（或打印）订单时间。您可以在实现中设置 N=5。

```
namespace HotelBookingSystem
{
    public class TravelAgency
    {
        private string agencyName;
        private Random rand = new Random();
        private int threadIndex = 0;
        private int priceCutCount = 0;
        private ArrayList creditCardIDList = new ArrayList() { "5023958311730285", "6385028598820285", "6999620102820285" };
        private int lastPrice = 0;

        public TravelAgency(string name)
        {
            agencyName = name;
        }
```

```
16.
17.         /// 获取时间戳
18.         public static string GetUtcNowTimeStamp()
19.         {
20.             DateTime now = DateTime.Now; // 当前日期和时间
21.             return now.ToString("yyyy-MM-dd HH:mm:ss");
22.         }
23.
24.         public int ComfirmRoomNumByNewPrice(int newPrice)
25.         {
26.             int roomsToOrder;
27.             if (lastPrice > newPrice)
28.             {
29.                 roomsToOrder = rand.Next(5, 10);  // 随机生成订单数量
30.             }
31.             else
32.             {
33.                 roomsToOrder = rand.Next(1, 5);
34.             }
35.             return roomsToOrder;
36.         }
37.
38.         // 降价事件处理函数
39.         public void OnPriceCut(int newPrice)
40.         {
41.             this.priceCutCount = this.priceCutCount + 1;
42.
43.             //根据模型确定订单数量: 旅行社将根据需求以及之前的价格和当前价格之间的差异计算要订购的房间数量。
44.             int roomsToOrder = ComfirmRoomNumByNewPrice(newPrice);
45.
46.             //生成信用卡号
47.             int cardNumberIndex = rand.Next(0, 3);
48.             string cardNumberString = creditCardIDList[cardNumberIndex].ToString();
49.
50.             OrderClass order = new OrderClass
51.             {
52.                 SenderId = this.threadIndex,
53.                 CardNumber = cardNumberString,
54.                 Amount = roomsToOrder,
55.                 OrderTime = GetUtcNowTimeStamp(),  // 在将订单发送到 MultiCell Buffer 之前，必须保存时间戳。
56.                 AgencyName = this.agencyName
```

```
            };
            lastPrice = newPrice;  // 记住本次价格，用于下次定购买数量

            string encodedOrder = Encoder.EncodeOrder(order); // 对象被发送到编码器进行编码。
            // 然后，旅行社将以字符串格式将订单发送到MultiCellBuffer。
            MultiCellBuffer.SetOneCell(encodedOrder);  // TravelAgency 将编码字符串发送到MultiCellBuffer 中的一个空闲单元格。
        }

        public void RunAgency()
        {
            // 旅行社的线程运行逻辑，可以是等待降价事件的发生并处理订单
            while (this.priceCutCount < 10)
            {
                this.threadIndex = Thread.CurrentThread.ManagedThreadId;
                // 这里可以加入具体的业务逻辑，例如等待事件发生，或者处理其他任务
                Thread.Sleep(1000); // 模拟旅行社的运行状态
            }
        }
    }
}
```

5、OrderClass（订单类）

OrderClass 是一个至少包含以下私有数据成员的类：

sender Id：发送者的身份，可以使用线程名或线程 Id；

卡号：表示信用卡号的整数；

amount：一个整数，表示要订购的房间数量；

```
namespace HotelBookingSystem
{
    /**
     * OrderClass 是一个至少包含 SenderId CardNumber Amount
     */
    public class OrderClass //每个订单都是一个OrderClass 对象。
    {
        public int SenderId { get; set; }
        public string CardNumber { get; set; }
        public int Amount { get; set; }

        public string OrderTime { get; set; }

```

```
14.         public string AgencyName { get; set; }
15.     }
16.
17.}
```

6、MultiCellBuffer（缓冲区类）

MultiCellBuffer 类用于旅行社（客户）和 HotelSupplier（服务器）之间的通信：该类有 2 个数据单元格。可以定义 setOneCell 和 getOneCell 方法，将数据写入可用单元格之一或从其中读取数据。信号量将允许机构获得写入其中一个缓冲单元的权利。但 HotelSupplier 仍然可以同时阅读。还需要同步。

```
1. namespace HotelBookingSystem
2. {
3.     /**
4.      * MultiCellBuffer 类用于旅行社（客户）和 HotelSupplier（服务器）之间的通信
5.      */
6.     public class MultiCellBuffer
7.     {
8.         private static Semaphore semaphore = new Semaphore(2, 2); // 最多两个单元
9.         private static DataCell[] buffer = new DataCell[2];
10.         private static readonly ReaderWriterLockSlim[] cellLocks = new ReaderWriterLockSlim[2]
11.         {
12.         new ReaderWriterLockSlim(),
13.         new ReaderWriterLockSlim()
14.         };
15.
16.         public static void SetOneCell(string data)
17.         {
18.             semaphore.WaitOne(); // 获取写入权限
19.             try
20.             {
21.                 for (int i = 0; i < buffer.Length; i++)
22.                 {
23.                     cellLocks[i].EnterWriteLock(); // 获取写锁
24.                     try
25.                     {
26.                         if (buffer[i] == null || buffer[i].flag == 0) // 找到空闲单元
27.                         {
28.                             buffer[i] = new DataCell { new_order = data, flag = 1 }; // 写入数据并标记为已使用
```

```
                            break;
                        }
                    }
                    finally
                    {
                        cellLocks[i].ExitWriteLock(); // 释放写锁
                    }
                }
            }
            finally
            {
                semaphore.Release(); // 释放写入权限
            }
        }

        public static string GetOneCell()
        {
            string data = null;
            for (int i = 0; i < buffer.Length; i++)
            {
                cellLocks[i].EnterReadLock(); // 获取读锁
                try
                {
                    if (buffer[i] != null && buffer[i].flag == 1) // 找到已使用的单元
                    {
                        data = buffer[i].new_order; // 读取数据
                        buffer[i].flag = 0; // 标记为空闲
                        break;
                    }
                }
                finally
                {
                    cellLocks[i].ExitReadLock(); // 释放读锁
                }
            }
            return data;
        }
    }

    // 数据单元类，用于缓冲区中的每个单元
    public class DataCell
    {
        public string new_order; // 存储订单
```

```
        public int flag = 0; // 标记单元状态：0 表示空闲，1 表示已使用
    }

}
```

7、Encoder（编码器）

Encoder 是类或类中的方法：Encoder 类将 OrderObject 转换为字符串。您可以选择任何方式将值编码为字符串，只要您可以将字符串解码为原始订单对象即可。您可以使用类或方法来实现 Encoder。

```
namespace HotelBookingSystem
{
    public class Encoder
    {
        /***
         * Encoder 类将 OrderObject 转换为字符串。
         */
        public static string EncodeOrder(OrderClass order)
        {
            return $"{order.SenderId},{order.CardNumber},{order.Amount}, {order.OrderTime}, {order.AgencyName}";
        }
    }

}
```

8、Decoder（解码器）

解码器是类或类中的方法：解码器将编码字符串转换回 OrderObject。

```
namespace HotelBookingSystem
{
    /**
     * 解码器将编码字符串转换回 OrderObject。
     */
    public class Decoder
    {
        public static OrderClass DecodeOrder(string encodedOrder)
        {
            string[] parts = encodedOrder.Split(',');
            return new OrderClass
            {
                SenderId = int.Parse(parts[0]),
                CardNumber = parts[1],
                Amount = int.Parse(parts[2]),
```

```
                OrderTime = parts[3],
                AgencyName = parts[4],
            };
        }
    }
}
```

9、Main 类

主线程将执行必要的准备、创建缓冲区类、实例化对象、创建线程和启动线程。

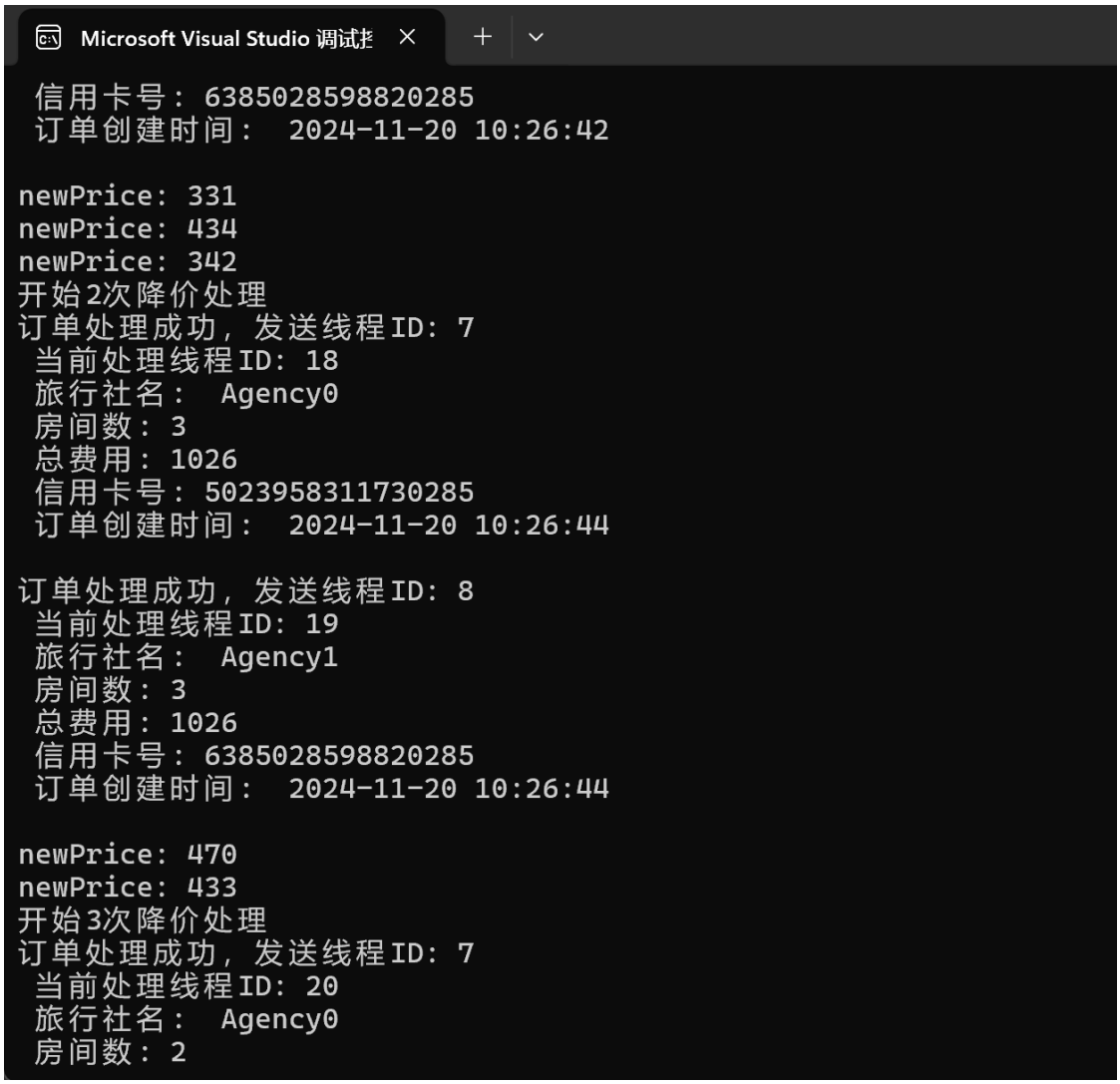
```
public class Program
{
    public static int MAX_TRAVEL_NUM = 5;
    public static int MAX_SUPPLIERS_NUM = 1;
    public static void Main(string[] args)
    {
        HotelSupplier hotelSupplier = new HotelSupplier();
        // 创建每个旅行社对象 并 启动每个旅行社线程
        TravelAgency[] travelAgency = new TravelAgency[MAX_TRAVEL_NUM];
        Thread[] agenciesThreads = new Thread[MAX_TRAVEL_NUM];
        for (int i = 0; i < MAX_TRAVEL_NUM; i++)
        {
            travelAgency[i] = new TravelAgency("Agency" + i); // TravelAgency1 到 TravelAgencyN，每个旅行社都是从类中的同一类（或同一方法）实例化的线程。
            for (int j = 0; j < MAX_SUPPLIERS_NUM; j++)
            {
                hotelSupplier.PriceCutEvent += travelAgency[i].OnPriceCut; // 旅行社的行动是由事件驱动的。每家旅行社都包含一个回调方法（事件处理程序），供 HotelSuppliers 在发生降价事件时调用。
            }
            agenciesThreads[i] = new Thread(new ThreadStart(travelAgency[i].RunAgency));
            agenciesThreads[i].Start();
        }
        //TravelAgency travelAgency1 = new TravelAgency("Agency1");
        //TravelAgency travelAgency2 = new TravelAgency("Agency2");

        //hotelSupplier.PriceCutEvent += travelAgency1.OnPriceCut;
        //hotelSupplier.PriceCutEvent += travelAgency2.OnPriceCut;
        hotelSupplier.RegCreditCardID("5023958311730285");
        hotelSupplier.RegCreditCardID("6385028598820285");
        hotelSupplier.RegCreditCardID("6999620102820285");
```

```
29.
30.         // 启动酒店供应商线程
31.         Thread hotelThread = new Thread(new ThreadStart(hotelSupplie
    r.StartHotelSupplier));
32.         hotelThread.Start();
33.
34.         // 主线程等待 hotelThread 执行完成
35.         hotelThread.Join(); // 确保主线程等待酒店线程结束,HotelSupplier
    线程终止后，线程将终止。
36.         foreach (Thread agencyThread in agenciesThreads)
37.         {
38.             agencyThread.Join();
39.         }
40.     }
41. }
```

四、运行截图

1、初始向服务器注册信用卡号

2、显示每一次价格波动，将其打印在控制台中。每个 newPrice 就是一次价格波动，保证有增有降。



3、打印订单的详情

```
newPrice: 451
开始10次降价处理
订单处理成功，发送线程ID: 7
 当前处理线程ID: 18
 旅行社名：  Agency0
 房间数：8
 总费用：3608
 信用卡号：6999620102820285
 订单创建时间：  2024-11-20 10:26:51

订单处理成功，发送线程ID: 8
 当前处理线程ID: 19
 旅行社名：  Agency1
 房间数：6
 总费用：2706
 信用卡号：5023958311730285
 订单创建时间：  2024-11-20 10:26:51
```