

CSE445/598 Project 2 (Assignment 2) – (100 Points)

Fall 2013

Team building due: **Saturday, September 14, 2013**, by 11:59pm (Arizona Time)

Project 2 due: **Saturday, September 28, 2013**, by 11:59pm (Arizona Time)

Introduction

The purpose of this project 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 are able to apply these concepts in a programming project.

You can choose to do this project as an **individual project** or as a **team project** of two, three, or four members. In the case of the team project, a declaration must be given, which identifies the parts of individual contributions and team efforts. An overall percentage of contribution of each member (e.g., 50% and 50%, or 30%, 35% and 35%) must be given, which will be used as a reference of assigning (scaling) grades. Only one copy of the project should be submitted by one of the team members. Teams built from 445 and 598 classes are allowed.

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 the 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 project in Section II, you may want go through the tutorial given in the textbook chapter 2, which consists of

- 6.1 Read the case study in text section 2.6.3.
- 6.2 Test the program given in the case study. The program can be used as the starting point for your project in Section II.
- 6.3 Extend the program based on the requirement in Section II.

Section II Project (Submission required)

Warning: This is a long programming project designed for a study load for three weeks of estimated $3 \times 8 = 24$ hours. It is challenging at both conceptual level and implementation level. You must distribute the load in the given three weeks. You would not have enough time to complete it if you start the project only in the last week before the project is due.

Purpose of this project is to exercise the concepts learned in this chapter. It is not the purpose of this project to create realistic services and applications. We will create more realistic services and applications in the subsequent projects. In this project, you can use a console application or a simple GUI application to implement the user interface to your program.

Description: Consider that you are creating a simplified hotel block booking system that involves hotel retailers and wholesale. The system consists of multiple travel agencies (clients), e.g., hotels.com, hotwire.com, and priceline.com, where one can book hotel rooms, and a hotel chain (hotel supplier or server), e.g., Holiday Inn, that supplies block hotel rooms to the agencies. The required architecture and the major components of the system are shown in the diagram below.

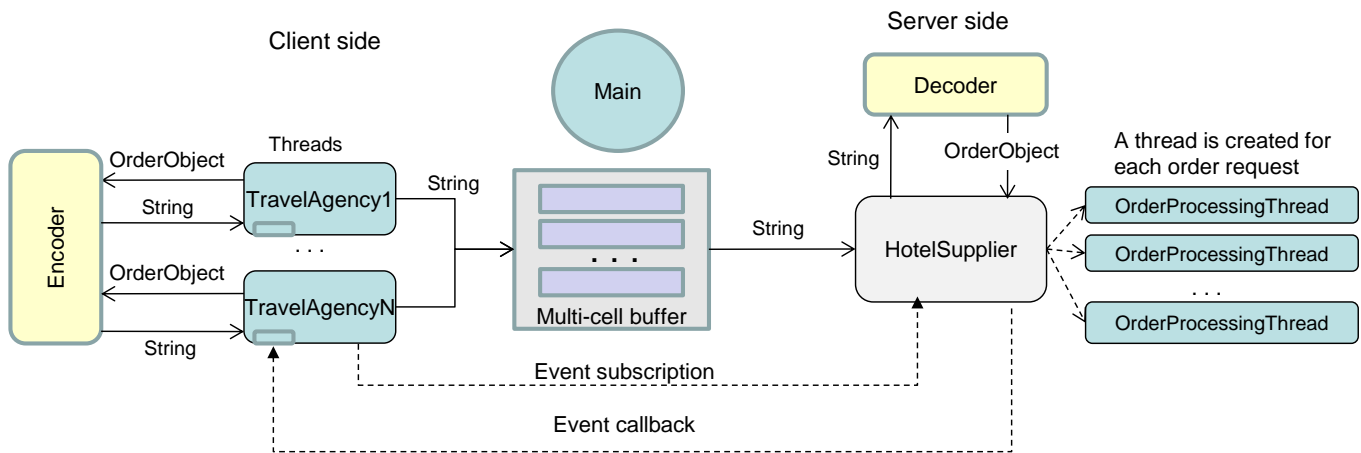


Figure 1 Architecture of a hotel block booking system

An **Operation Scenario** of the hotel block booking system is outlined below:

- (1) The **HotelSupplier** uses a pricing model to calculate the room price. If the new price is lower than the previous price, it emits a (promotional) event and calls the event handlers in the travel agencies that have subscribed to the event.
- (2) A **TravelAgency** evaluates the price, generates an **OrderObject** (consisting of multiple values), and sends the order to the **Encoder** to convert the order object into a plain string.
- (3) The **Encoder** converts the object into a string.
- (4) The **Encoder** sends the encoded string back to the caller.
- (5) The **TravelAgency** sends the encoded string to one of the free cells in the **MultiCellBuffer**.

- (6) The **HotelSupplier** receives the encoded string from the MultiCellBuffer and sends the string to the Decoder for decoding.
- (7) The **Decoder** sends the OrderObject to the HotelSupplier. The decoded object must contain the same values generated by the TravelAgency.
- (8) The HotelSupplier creates a new thread to process the order;
- (9) The **OrderProcessingThread** processes the order, e.g., checks the credit card number and calculates the amount.
- (10) The OrderProcessingThread sends a confirmation to the travel agency and prints the order (on screen).

Components in the diagram are explained in details as follows, with their grades (points) allocation:

1. **HotelSupplier** is a class on the server side: It will be started as a thread by the Main method and will perform a number of functions. It uses a PricingModel to determine the room prices. It defines a price-cut event that can emit an event and call the event handlers in the TravelAgencys if there is a price-cut according to the PricingModel. It receives the orders (in a string) from the MultiCellBuffer. It calls the Decoder to convert the string into the order object. For each order, you can use the existing thread **or start a new thread (resulting in multiple threads for processing multiple orders [1])** from OrderProcessing class (or method) to process the order based on the current price. There is a counter p in the HotelSupplier.

After p (e.g., p = 10) price cuts have been made, the HotelSupplier thread will terminate. Before generating the first price, a time stamp must be saved [2]. Before the thread terminates, the total time used will be calculated and saved (or printed). [15 points]

[1] Note: If you are doing the project as an individual project, you can choose to use a single thread to process all the orders. If you are doing the project as a group project, you must use multiple threads to process the orders.

[2] Time stamp and performance evaluation are required for group project and not required if you are doing the project as an individual project.

2. **PricingModel**: It can be a class or a method in HotelSupplier class. It decides the price of rooms. It can increase price or decrease the price. You must define a mathematical model [3] (formula) to determine the price based on the order received within a given time period and the number of rooms available in the HotelSupplier in the same time period. You can use a hard-coded table of the price in each week day. However, you must make sure that your model will allow the price goes up some time and goes down some other time. [5 points]

[3] For group project, you **cannot** use the random numbers in the model. You must develop a new price model. For the individual project, you can use random numbers, as being used in the tutorial.

3. **OrderProcessing** is a class or a method in a class on the supplier's side. Whenever an order needs to be processed, a new thread is instantiated from this class (or method) 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 number from the travel agencies must be a number registered to the HotelSupplier, 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., $\text{unitPrice} * \text{NoOfRooms} + \text{Tax} + \text{LocationCharge}$. It will send a confirmation to the travel agency when an order is completed. You can implement the confirmation in different ways. For example, you can use another buffer for the confirmation, where you can use a buffer cell for each thread, so that you do not have to consider the conflict among the threads. However, you still need to coordinate the write and read between the producer and the consumer. [10 points]

4. **TravelAgency1** through **TravelAgencyN**, each travel agency is a thread instantiated from the same class (or the same method) in a class. The travel agency's actions are event-driven. Each travel agency contains a call-back method (event handler) for the HotelSupplier to call when a price-cut event occurs. The travel agency will calculate the number of rooms to order, for example, based on the need and the difference between the previous price and the current price. The thread will terminate after the HotelSupplier thread has terminated. Each order is an OrderClass object. The object is sent to the Encoder for encoding. The encoded string is sent back to the travel agency. Then, the travel agency will send the order in String format to the MultiCellBuffer. Before sending the order to the MultiCellBuffer, a time stamp must be saved. When the confirmation of order completion is received, the time of the order will be calculated and saved (or printed). [10 points]

5. **OrderClass** is a class that contains at least the following private data members and public methods:

- senderId: the identity of the sender, you can use thread name or thread id;
- cardNo: an integer that represents a credit card number;
- amount: an integer that represents the number of rooms to order;
- setID and getID: methods allow the users to write and read senderId member
- setCardNo and getCardNo: methods allow the users to write and read cardNo member
- setAmt and getAmt: methods allow the users to write and read Amount member

You must decide if these methods need to be synchronized. The instances created from this class are of the OrderObject. [10 points]

6. **MultiCellBuffer** class is used for the communication between the travel agencies (clients) and the HotelSupplier (server): This class has n data cells (for individual project, $n = 2$ and for group project, $n = 3$). The number of cells available must be less than ($<$) the max number of travel agencies in your experiment. A setOneCell and getOneCell methods can be defined to write data into and to read data from one of the available cells. You can use a semaphore of value n to manage the cells. You **cannot** use a queue for the buffer, which is a different data structure. Note: The semaphore will allow an agency to gain the right to write into the buffer. But the HotelSupplier can still read at the same time. Synchronization is also required. [10 points]

7. **Encoder** is a class or a method in a class: The Encoder class will convert an OrderObject into a string. You can choose any way to encode the values into a string, as long as you can decode the string to the original order object. You can use a class or a method to implement the Encoder. [10 points]

[4] For a group project, after data format conversion, the encoder calls encryption service in ASU repository to encrypt the string. If you use the .svc service, you should use "Add Service Reference". If you use the .asmx service, you should use "Add Web Reference". The recommendation is to use the .svc service, as it is the new standard. If you did not choose the encryption/decryption question in assignment 1, you can follow the example in textbook section 6.3.2 to call the encryption/decryption service. The service is at: <http://venus.eas.asu.edu/WSRepository/Services/EncryptionWcf/Service.svc>

8. **Decoder** is a class or a method in a class: The Decoder will convert the encoded string back into the OrderObject.

[4] For a group project, decryption is required. In this case, you will first call ASU decryption service and then convert the string into the OrderObject. [10 points]

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

10. Performance Table and User Manual: You must submit a document in WORD, PDF, or a web page. You must give: [10 points]

(1) Name(s) of developer(s) of the system and the percentage of contribution of each member if the project is done as a group project. The percentage will be used to scale the score.

(2) Class diagram: Each class block should include the class name, attributes and methods, and the relations between the classes. For the definition of class diagram, see

<http://www.objectmentor.com/resources/articles/umlClassDiagrams.pdf>

You can use the Visual Studio to generate the class diagram automatically.

(3) A guide on testing the system: development environment used, loading the program, input, and expected output;

(4) A screenshot of your execution of the program with input and output;

(5) Performance tables (The performance table is required for group projects).

Please measure and list the following performance values. When you are recording the performance data, you must close all other applications before you run the experiment.

The table must contain:

N = 1, 2, 3, ..., 10, where N is the TravelAgency threads' numbers (ID).

ARTOC: Average Response Time between placing Order and receiving Confirmation

TTC: Total Time between the first price release and completion measured by the HotelSupplier

Table 1: Performance data (give the no. of cores and processor clock rate)

N: TravelAgency thread Number	ARTOC: Average Response Time	TTC: Total Time
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

Group project option summary

If you are doing the project as a group of 2, 3, or 4 members, you are required to complete the following additional tasks, which have been explained in the description above:

[1] You must use multiple threads to process the orders.

[2] Time stamp required.

[3] Price Model: You cannot use the random numbers in the model. You must develop a new price model.

[4] Encryption and decryption are required.

[5] Performance Table is required.

Notes:

1. This project can be done in C# (Visual Studio) **OR** in Java (Eclipse). You must indicate the environment (VS 2010, VS 2012, or Eclipse) you use, so that the TA can use the same environment to grade the project. We taught multithreading in both Java and C#. However, we did not teach the event-driven programming in Java. If you choose to do the project in Java, you need to study this part on your own. I suggest that you do the project in Java only if you are really good at Java and already know how to program events. The event handling in Java is not as easy as in C# for this project, and you need to take extra effort to implement the required functions.
2. You must follow what is defined in the assignment/project document. You have flexibility to choose your implementation details if they are not specified in the document. If you are not sure on any issue, ask the instructor or the TA by posting the question in the discussion board.
3. The program and each component of the program must be well commented.
4. You can choose to do this project as an individual project or a group project. I encourage you to do in group. Projects 3 and 5 will be required group projects and thus, you need to build a team anyway.

Grading of Programming Assignment/Project

The TA will grade your program following these steps:

- (1) The TA will read your program and give points based on the points allocated to each component, the readability of your code (organization of the code and comments), logic, inclusion of the required functions, and correctness of the implementations of each function.
- (2) Compile the code. If it does not compile, 40% of the points given in (1) will be deducted. For example, if you are given 20 points in step (1), your points will become 12 if the program fails to compile.
- (3) If the code passes the compilation, the TA will execute and test the code. If, for any reason, the program gives an incorrect output or crashes for any input, 20% of the points given in (1) will be deducted.

Please notice that the TA will not debug your program to figure out how big or how small the error is. You may lose 40% or 20% of your points for a small error such missing a comma or a space!

Late submission deduction policy:

- No penalty for late submissions that are received within 24 hours after the deadline;
- 1% grade deduction for every hour after the 24 hours!

Where to Submission?

All submissions must be electronically submitted to the assignment folder where you downloaded the assignment paper. All files must be zipped into a single file.