CSE445/598 Project 2 (Assignments 3 and 4 – 50+50 Points)
Summer 2020

A3 and A4 due: Saturday, June 6, 2020, by 11:59pm (Arizona Time), plus a one-day grace period

---

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.

This is an **individual project**. No cooperation in doing the assignment is allowed. All questions about the assignment must be posted into the course discussion board. Questions may not be posted to the forums on internet.

Section I Preparation and Practice Exercises (No submission required)

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 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 project in Section II.

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

7. For debugging multithreading programs, please also read:

https://msdn.microsoft.com/en-us/library/ms164746.aspx

Section II Project Tasks (Submission required)

Warning: This is a long programming project and it is challenging at both conceptual level and implementation level.

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 e-commerce system. The system consists of multiple chicken retailers (clients) and a single chicken farm (server). The system consists of the major components shown in the diagram below.

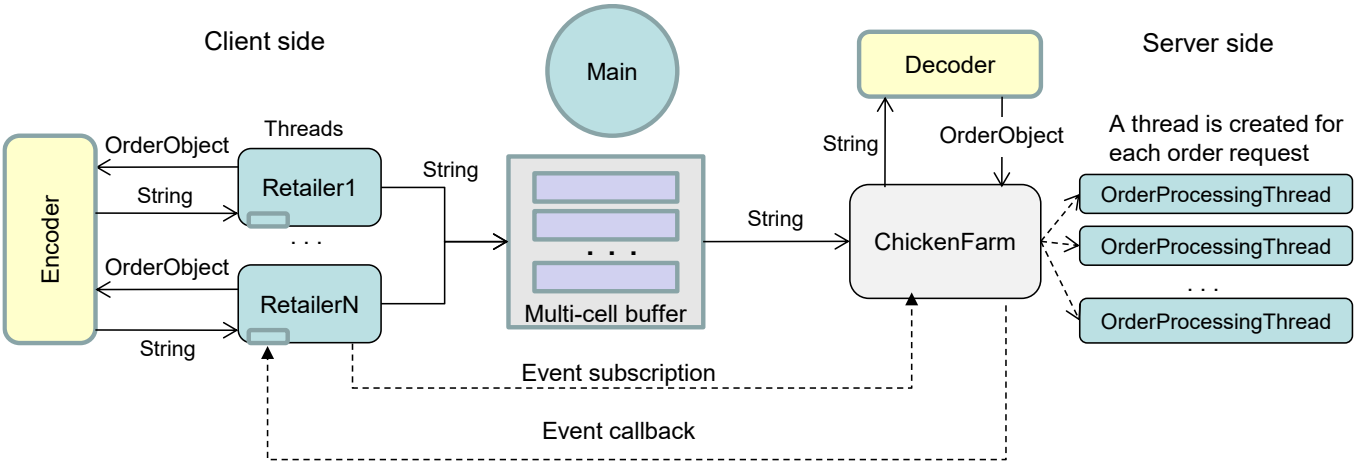

Figure 1 Architecture of an e-commerce system

An **Operation Scenario** of the e-commerce system is outlined below:

(1) The **ChickenFarm** uses a pricing model to calculate the chicken price. If the new price is lower than the previous price, it emits an event and calls the event handlers in the retailers that have subscribed to the event.
(2) A **Retailer** 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 Retailer sends the encoded string to one of the free cells in the **MultiCellBuffer**.
(6) The **ChickenFarm** receives the encoded string from the MultiCellBuffer and sends the string to the Decoder for decoding.

(7) The **Decoder** sends the OrderObject to the ChickenFarm. The decoded object must contain the same values generated by the Retailer.
(8) The ChickenFarm 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 retailer and prints the order.

Components and assignment tasks in the diagram are explained in details as follows, with their grades allocation:

Assignment 3 tasks [50 points]:

1. **ChickenFarm** is a class on the server side: It will be started as a thread by the Main method and will perform a number of service functions. It uses a PricingModel to determine the chicken prices. It defines a price-cut event that can emit an event and call the event handlers in the Retailers if there is a price-cut according to the PricingModel. It receives the orders (in string) from the MultiCellBuffer. It calls the Decoder to convert the string into the order object. For each order, it starts a new thread (resulting in multiple threads for processing multiple orders*) from OrderProcessing class (or method) to process the order based on the current price. There is a counter p in the ChickenFarm. After p (e.g., p = 10) price cuts have been made, the ChickenFarm thread will terminate. [20 points]

2. **PricingModel**: It can be a class or a method in ChickenFarm class. It decides the price of chickens. It can increase price or cut the price. You can define a mathematical model (formula) to determine the price based on the order received within a given time period and the number of chickens the farm can produce in the same time period. You can use a random model to generate the prices. 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. **OrderProcessing** is a class or a method in a class on server 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 own credit card format, for example, the credit card number from the retailers must be a number registered to the ChickenFarm, 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*NoOfChickens + Tax + shippingHandling. It will send a confirmation to the retailer when an order is completed. The confirmation must be implemented using a callback method. [10 points]

4. **Retailer1** through **RetailerN**, where N = 5, each retailer is a thread instantiated from the same class (or the same method) in a class. The retailers' actions are event-driven. Each retailer contains a callback method (event handler) for the ChickenFarm to call when a price-cut event occurs. The retailer will calculate the number of chickens 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 ChickenFarm 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 retailer. Then, the retailer 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). [15 points]

Assignment 4 tasks [50 points]:

5. **OrderClass** is a class that contains at least 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;
- amount: an integer that represents the number of chickens to order;

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 created from this class are of the OrderObject. [10 points]

6. **MultiCellBuffer** class is used for sending the order from the retailers (clients) to the chickenFarm (server). This class has n data cells (you can simply set n = 2). The number of cells is less than the max number N (you can set N = 5 or enter N from keyboard) of retailers 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 must use a semaphore of value n to manage the cells and use a lock for each cell to ensure the synchronization.

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]

8. **Decoder** is a class or a method in a class: The Decoder will convert the string back 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. Submit the test input cases and test output results. [10 points]

Notes:

1. This project can be done in C# (Visual Studio) **OR** in Java (NetBeans). You must indicate the environment (VS 2013, VS 2015, or NetBeans) 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. 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 retailer must have completed at least one ordering process before the chickenFarmer terminate.

Submission Requirement

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

If you have saved a project/Website in a different folder, you can copy the folder containing the project/Website to the directory where the other projects are saved. Then go into Visual Studio and delete the project/website that was in a different place. Then right click the solution in Visual Studio and add existing project/website, browse to the new location and select the project/website to link the moved project/website into the solution.

Submission preparation notice: The assignment consists of multiple **distributed** projects and components. They may be stored in different locations on your computer when you create them. You must copy these projects into a single folder for blackboard submission. To make sure that you have all the files included in the zip file and they work together, you must test them before submission. You must also download your own submission from the blackboard. Unzip the file on a different location or machine, and test your assignment and see if you can run the solution in a different location, because the TA will test your application on a different machine.

Grading and Rubrics

Each sub-question (component) has been assigned certain points. We will grade your programs following these steps:

(1) Compile the code. If it does not compile, 50% of the points given for the code under compilation will be deducted. Then, we will read the code and give points between 50% and 0, as shown in right part of the rubric table.

(2) If the code passes the compilation, we will execute and test the code using test cases. We will assign points based on the left part of the rubric table.

In both cases (passing compilation and failed compilation), we will read your program and give points based on the points allocated to each component (sub-question), 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.

Please notice that we will not debug your program to figure out how big or how small the error is. You may lose 50% of your points for a small error such missing a comma or a space!

We will apply the following rubrics to **each sub-question** listed in the assignment. Assume that points assigned to a sub-question is *pts*:

Rubric Table

Major	Code passed compilation				Code failed compilation		
Points	pts * 100%	pts * 90%	pts * 80%	pts * 70% -60%	pts * 50% - 40%	pts * 30% - 10%	0
Each sub-question	Meeting all requirements, well commented, and working correctly in all test cases	Working correctly in all test cases. Comments not provided to explain what each	Working with minor problem, such as not writing comments, code not working in certain	Working in most test cases, but with major problem, such as the code fail a	Failed compilation or not working correctly, but showing serious effort in	Failed compilation, showing some effort, but the code does not implement the required work.	No attempt

		part of code does.	uncommon boundary conditions.	common test case	addressing the problem.		

Late submission deduction policy:

- No penalty for late submissions that are received within 24 hours of the given deadline;
- 2% grade deduction for every hour after the first 24 hours! No submission after Monday.