

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

Fall 2022 with update

Project 2 (A3 and A4) due: **Saturday, September 24, 2022**, by 11:59pm (Arizona Time), plus a grace period of 24 hours.

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 or three members. In the case of the team project, a declaration must be given at the end of the assignment, 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. For this project, we do not need a formal team building process. For a team project, one copy should be submitted by one team member. For an individual project, each one just submit one's own project. If one submits an individual project, one may not work with anyone else. Any similarity between two individual projects can lead to a cheating case investigation.

Not for this project, but for the following projects 3 and 5, 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 project 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 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://docs.microsoft.com/en-us/visualstudio/debugger/debug-multithreaded-applications-in-visual-studio?view=vs-2019>

Section II Project (Submission required)

Warning: This is a long programming project designed for a study load for three weeks of estimated 3*8 = 24 hours. It is challenging at both the conceptual level and the implementation level. You must distribute the load in the given three weeks. You will not have enough time to complete project 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. You do not need to create Web applications. You do not need to create services. You can simply use classes in object-oriented paradigm.

Description: Consider that you are creating an e-business: a new ticket block booking system that involves ticket agents and cruises (or cruise lines), such as Disney Cruise Lines and Carnival Cruise. The system consists of multiple ticket agents (clients) and multiple cruises (servers). The ticket agents can buy in quantity (block) of tickets from the cruises with lower prices, and then resell the tickets to their customers with competitive prices. The required architecture and the major components of the system are shown in the diagram in Figure 1.

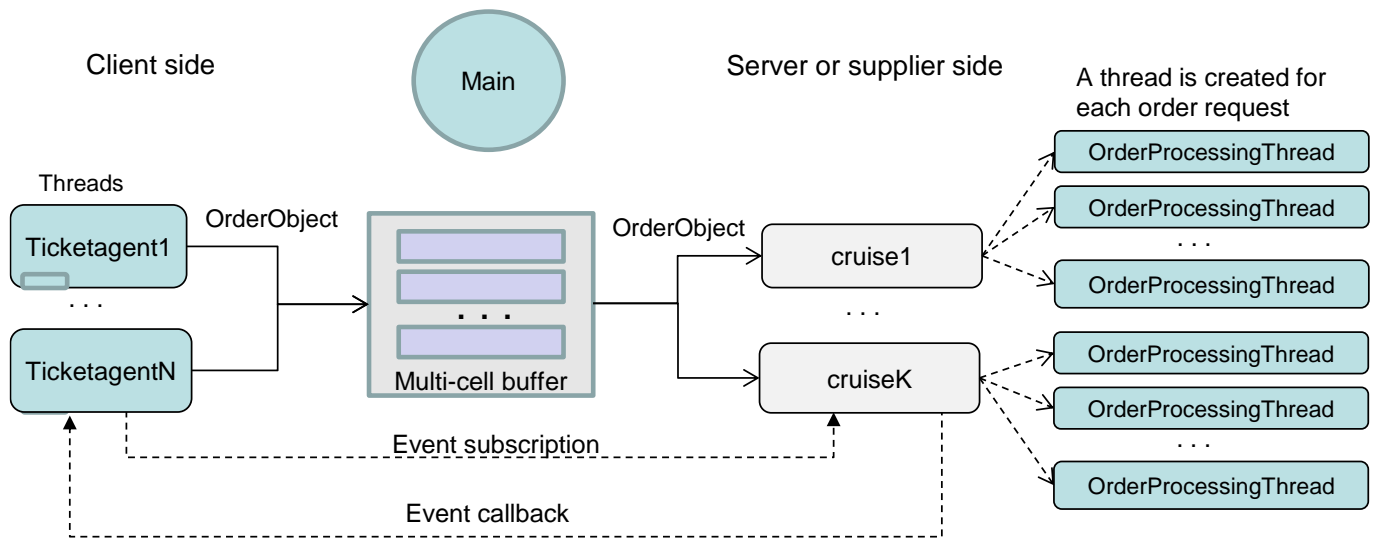


Figure 1 Architecture of a new ticket booking system in multithreading system

In this project, you will simulate both clients and servers in one system using multithreading and event-driven programming. You will **not** implement this project in a web environment. You will further implement such systems in distributed web client and web server systems in the following projects in this course.

An **Operation Scenario** of the ticket/cruise booking system is outlined as follows:

- (1) A **Cruise** uses a pricing model to calculate dynamically the ticket price for the ticket agents. The prices can go up and down 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 ticket agents (clients) that have subscribed to the event.
- (2) A **TicketAgent** evaluates the needs based on the new price and other factors, generates an **OrderObject** (consisting of multiple values), and sends the order to the cruise through a **MultiCellBuffer**.
- (3) The **TicketAgent** sends the **OrderObject** to the promoting cruise through one of the free cells in the **MultiCellBuffer**.
- (4) The **Cruise** receives the **OrderObject** from the **MultiCellBuffer**.
- (5) The **Cruise** 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 ticket agent and prints the order (on screen).

Components in the diagram in Figure 1 are explained in details as follows, with their grading scores (points) allocation. Components (Tasks) 1, 2, 3, and 4, belong to Assignment 3, and the rest of the components belong to Assignment 4. You will submit both assignments together as a project. We will enter the scores under assignments 3 and 4, respectively, for the purpose of score management.

Assignment 3 Tasks

1. **Cruise1** through **CruiseK*** are the objects of the same class (or different classes for team project) on the server side: Each object has a method to be started as a thread by the **Main** method and will perform a number of functions. It uses a **PricingModel** to determine the ticket prices (different pricing models for

different cruises in team project). It defines a price-cut event that can emit an event and call the event handlers in the TicketAgent 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 (or method) to process the order based on the current price. There is a counter t in the Cruise. After t (e.g., t = 20) price cuts have been made, a Cruise thread will terminate. The ticket agents do not have to make an order after each price cut. [20 points]

*Note 1: For the individual project, the number of cruises $K = 1$. For a team project, $K = 2$ if you have a 2-member team, and $K = 3$ if you have a 3-member team.

2. **PricingModel**: It can be a class or a method in the Cruise class. It decides the price of tickets, which must be between 40 and 200. It can increase or decrease the price. You must define a mathematical model (formula). The model can be a simple random function for individual projects. However, for the team projects, a more complex model (formula) must be developed, where the price must be a function with multiple parameters, such as the season (weekday), the available number of the tickets, and the number of orders received within a given time period. In other words, the function must take these parameters as inputs. You can use a hard-coded table of the prices, for example, in each season (weekday). For team project, each cruise must have a different price model. You must make sure that your model will allow the price to go up sometimes and go down other times within your iterations of testing. [10 points]
3. **OrderProcessing** is a class or a method in a class on the server'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. If you are doing an individual project or a two-member team project, you can define your credit card format, for example, the credit card number from the ticket agents must be a number registered to the Cruise, or a number between two given numbers (e.g., between 5000 and 7000). For the three-member team project, a bank service must be created. Each OrderProcessing thread will calculate the total amount of charge, e.g., $\text{unitPrice} * \text{NoOfTickets} + \text{Tax} + \text{LocationCharge}$. For the team (two and three members) projects, a confirmation must be sent back to the ticket agent when an order is completed. You can implement the confirmation in different ways. For example, you can use the same buffer or 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. You can also write different TicketAgent classes for different threads, instead of generating them from the same class. [10 points]
4. **TicketAgent1** through **TicketAgentN**. You must use a variable N, but you can set $N = 5$ in your implementation. Each ticket agent is a thread created by a different object instantiated from the same class (or the same method in a class). The ticket agent's actions are event-driven. Each ticket agent contains a callback method (event handler) for the cruises 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 ticket agent thread. The ticket agent will read the new price and calculate the number of tickets 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 Cruise threads have terminated. Each order is an OrderClass object. Then, the ticket agent will send the order to the MultiCellBuffer. For team project with 2 or 3 members, 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). For the three-member team project, the ticket agents must issue priceRequest through the buffer and then order after receiving the response, in addition to order based on the receiving price cut event. The individual project and two-member team project just need to order based on the price cut event. [10 points]

Assignment 4 Tasks

The following components will be counted as assignment 4 tasks.

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.
- receiverID: the identity of the receiver, you can use thread name or a unique name defined for a cruise. If you are doing an individual project, you do not need this field.
- quantity: an integer that represents the number of tickets to order.
- unit price: a double that represents the unit price of the ticket received from the cruise.

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** is a class that is used for the communication between the ticket agents (clients) and the cruises (servers): This class has n data cells (for individual project, n = 2, and for team project, n = 3). Each cell is a **reference** to an **OrderClass** object. The number of cells available must be less than (<) the max number of ticket agents in your experiment. To write data into and to read data from one of the available cells, setOneCell and getOneCell methods can be defined. You **must** use a semaphore of value n to manage the availability of the cells. You **must** use an additional lock mechanism to provide read or write permissions for a cell. This lock mechanism can be one of the mechanisms offered through a library function, or a mechanism that you define by yourself. A cell can be used as long as it is available (unlocked). You **cannot** use a queue for the buffer, which is a different data structure. You can use an array of objects. The semaphore allows a ticket agent to see the availability of the cells, while the lock mechanism allows a ticket agent to gain the right to write into one of the buffer cells. The Cruise can read buffer cells at the same time. To ensure the thread safety, synchronization through lock or monitor is required for read/write and write/write overlap. If your message generation is slow, e.g., you use sleep function in the cruise threads and in the agent threads, resulting in 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 can increase N and/or reduce the sleep time in the cruise 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. [30 points]

Second, if your message generation is slow, e.g., you use sleep function in the cruise thread and in the agency threads, resulting that the buffer is idle most of the time. In this case, you can reduce the sleep time, or to add a sleep time on the buffer operation, so that the cell will be locked for a while and second and third cell have a chance for being used and being reflected in the printouts.

7. **Bank service** is a class. This question is for the three-member team projects only. It allows the ticket agent to apply for a credit card number to a bank. The ticket agent then will use this number to purchase tickets from the cruises. The cruises will send the credit card number and the amount to the bank. The bank charges the account and returns the message “valid” if the account exists and the funds are sufficient for the purchase, otherwise, it returns “not valid”. When a cruise sends the credit card number and the amount to the bank, it will call the encryption service in the ASU repository to encrypt the credit card numbers in a string. If you use the .svc service, you must use "Add Service Reference". On the other hand, the bank will call the ASU decryption service to decrypt the number before

comparing with the valid credit card numbers. The Bank class must also allow the account holder to deposit funds into the account. [20 points]

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

Additional Tasks for team projects

Two-member Teams: If you are doing the project as a team project with two members, you are required to complete the following additional tasks:

- [1] You must implement multiple suppliers (cruises) with $K = 2$. You must deal with all the issues that are incurred because of the increase in the number of suppliers.
- [2] You must develop a more complex price model and different price models for different cruises.
- [3] The supplier must send a confirmation back to the ticket agent when an order is completed.
- [4] The buffer size (number of cells in the buffer) is 3, instead of 2.

In this option, one member must write the code of the ticket agents and the main program. The other member must write the code of the supplier. The two members must jointly write the code for the buffer class.

Three-member Teams: If you are doing the project as a team project with three members, you are required to complete all the additional tasks for the two-member team projects and the following additional tasks:

- [1] You must implement multiple suppliers (cruises) with $K = 3$. You must deal with all the issues that are incurred because of the increase in the number of suppliers.
- [5] If you have a three-member team, you must also define the bank service. One member must implement the Bank class while sharing some other tasks in the assignment.

For three-member team, there are totally 70 points in Assignment 4 of the project. The 70 points will be scaled to 50 points proportionally. For example, $70 \rightarrow 50$, $60 \rightarrow 43$, and $50 \rightarrow 36$.

Notes:

1. It is the purpose of this course to enforce the knowledge of C# and Visual Studio, as we have multiple courses in our program that use Java. We encourage you to use C#. However, we allow the use of Java (on NetBeans) in this assignment. You must indicate the environment (e.g., 2019 or 2022 or NetBeans) that 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 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 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 explicitly specified in the document. If you are not sure on any issue, ask the instructor or the TA by posting the question in the course discussion board.
3. The program and each component of the program must be well commented.

4. 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.
5. You can choose to do this project as an individual project or a team project. I encourage you to do as a team project. Projects 3 and 5 will be required team projects and thus, you need to build a team anyway. The mandatory team building process is for projects 3 and 5. The team used in project 2 does not have to be the same team for projects 3 and 5. However, the team for projects 3 and 5 must be the same, as the project in project 5 will be based on project 3.
6. Projects 3 and 5 are team projects. If you have formed a team in this project, you can use the same team for projects 3 and 5. Of course, you can also use a new team for the projects 3 and 5.

Submission Requirement

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.

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 choose your own location to store the project when you create the project. Then, you can copy these projects into a single folder for the 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 Canvas. Unzip the file on a different machine, and test your assignment and see if you can run the solution in a different machine, because the TA will test your application on a different machine.

If you submitted an empty project folder, or an incomplete project folder, we cannot grade your resubmission after the due date! We grade only what you submitted before the submission due date. Please read FAQ document in the course Web page for more details.

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!

Grading and Rubrics

Each sub-question (programming tasks) 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 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 as 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 part of code does.	Working with minor problem, such as not writing comments, code not working in certain uncommon boundary conditions.	Working in most test cases, but with major problem, such as the code fail a common test case	Failed compilation or not working correctly, but showing serious effort in addressing the problem.	Failed compilation, showing some effort, but the code does not implement the required work.	No attempt