CMPE 142 – Spring 2021 - Lab 1 (Individual)

Assignment objectives

1. Understand and use functions to create, read, write, and open files.
2. Understand and use functions related to thread processes.
3. Write a concurrent program that illustrates operating system scheduling issues.

Due Dates: Part A, B are due on February 25th .
Part C is due on March 4th.

Part A - (10 points)

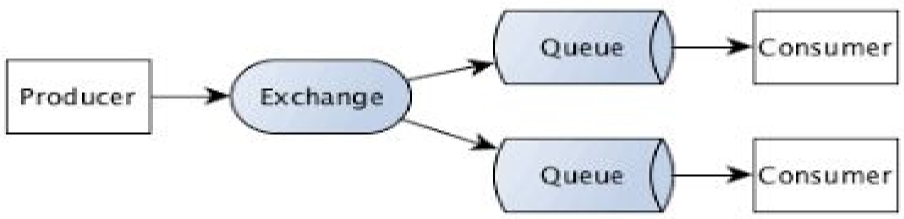
Create three threads that repeat the access of the "counter" (**local variable**) 100 times, then sleep for 10ms each time. Display the time it takes for the main function to exit and the value of the counter. Do this programming exercise in Go and C++.

(1) C/C++: Examine the sample program on thread programming (thread_example.cpp) illustrating thread creations. Run thread_example.cpp.
(2) Go: Implement the equivalent program in Go.
(3) Measure latencies for creation and execution of threads. Compare and contrast two programs for performance.

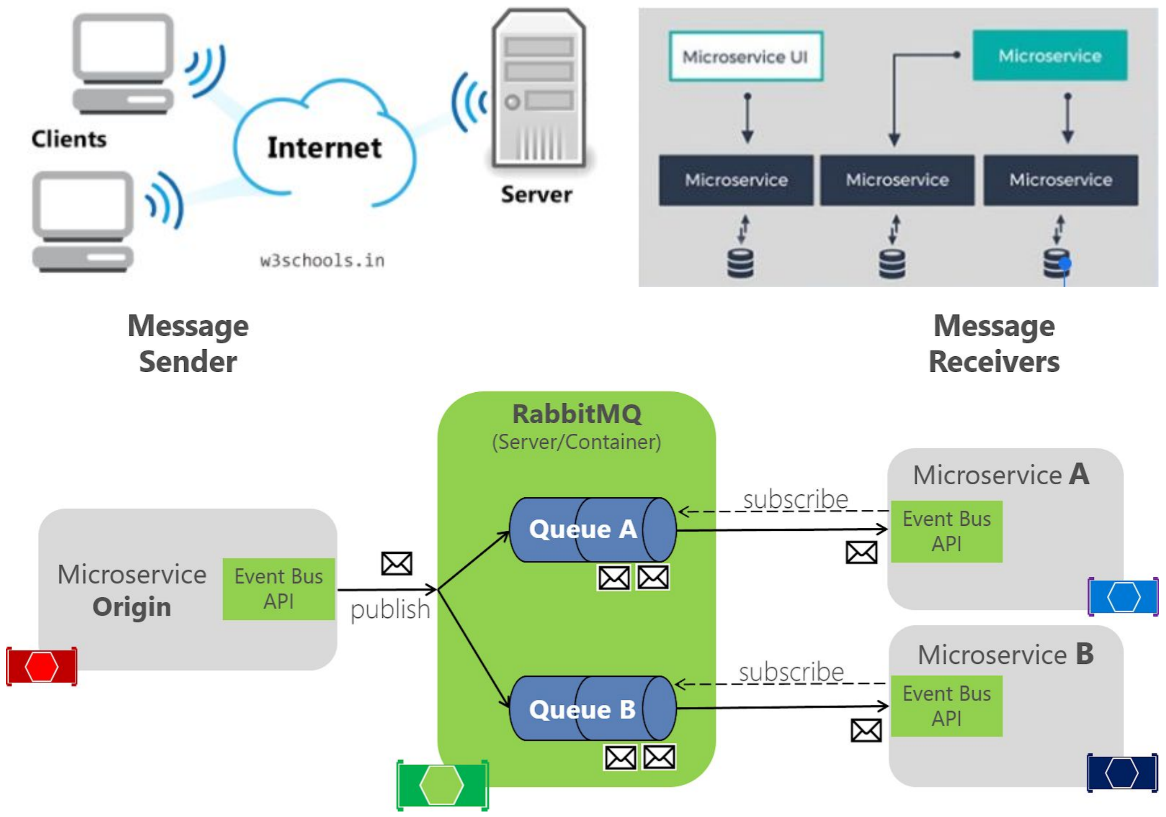
Part B - (20 points)

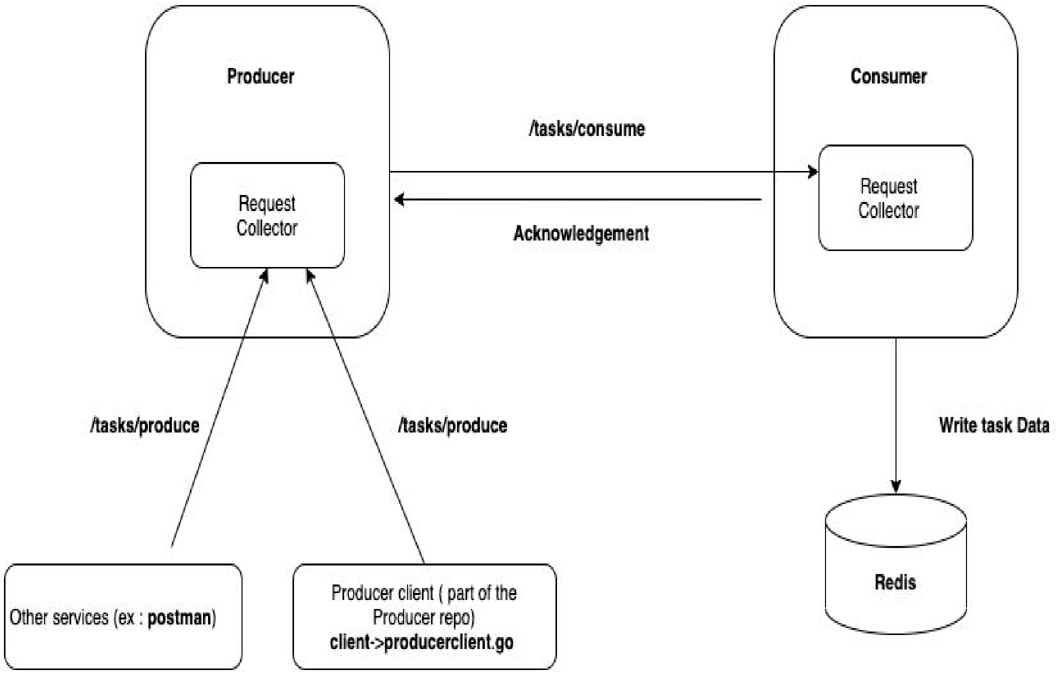
(1) Write a **Go program** that simulates the producer, exchange, and two consumers. You will support two types of queue and they are indicated in the message type as A and B. Indicate the time each message completes, and calculate the mean waiting time for each message from a consumer. The input file that you will use is in the Canvas (input)..

You'll need to create four Go threads. Producer reads the data from the input file and writes it to the exchange thread through a channel. The exchange supports two message queues identified by the message type (A or B). This middleware will receive messages from a producer and pass them to a consumer. You can assume that you know the number of messages in the data file for simplicity.



Part C - (20 points)





Fill in missing parts of code for the given Producer-Consumer program. Only the instructed functions in producer.go and collector.go in "Producer/service" need to be implemented. It will be commented in the given files where you need to fill in the code. Only turn in the completed **collector.go** and **producer.go**.

collector.go:

1) Create a global variable "TaskChan", which is a buffered channel of 10
2) At the end of RequestCollector(), push "taskRequest" into the buffered channel that you created in 1.

producer.go:

1) Implement StartProducer(). This function will loop indefinitely; until StopProducer() is executed. In the for loop, create a select case that will take in the items from two different channels: "TaskChan" and "prd.StopChan"
   a) In the case that a task is received from "TaskChan," call the function "prd.ConsumerClient(task)", passing in the task that you received.
   b) In the case that something is received from "prd.StopChan," break out of the loop.
2) Implement StopProducer(). This function will have an anonymous goroutine function call that simply passes in true to "prd.StopChan."

The provided code is a simple producer-consumer program that utilizes HTTP requests to start and stop a producer/consumer. It is also used to send tasks and acknowledgements between the producer and consumer. The following is a step-by-step look at the flow of the code. A flowchart will be uploaded in the lab files.

1. First, a producer and a consumer needs to be started through an HTTP request at **/producer/start** and **/consumer/start**.
2. These requests will invoke a function that **initializes** and **starts** a producer and a consumer respectively, in separate goroutines.
3. The producer will wait for tasks to be pushed to its buffered channel. This is done through an HTTP request at **/tasks/produce**. This request will invoke a function that creates a task with given variables and push it to a buffered channel.
4. When a task is sent through the buffered channel, it will **create a response** and **post it** on the consumer's HTTP endpoint for task consumption, which is **/tasks/consume**. This request will invoke a function that allows the consumer to get the task and push it to its buffered channel.
5. The task, now at the consumer's end, will be consumed and the consumer will send an acknowledgement back to the producer.
6. Finally, when the task is consumed, it is written to a Redis database.

How to run Producer-Consumer program:

1. Put "go-producer-consumer-v1-main" into your go src folder.
2. Open two terminals in VSCode.
3. Run Consumer/main.go and Producer/main.go in separate terminals.
4. Open Postman.
5. Create a new workspace with any name.
6. Import Producer and Consumer Postman collections.
7. Start a consumer by selecting Start under Consumer and clicking send.
8. Do the same for Producer.
9. Send the Produce Task request. The terminals should show that the task was sent and received successfully.

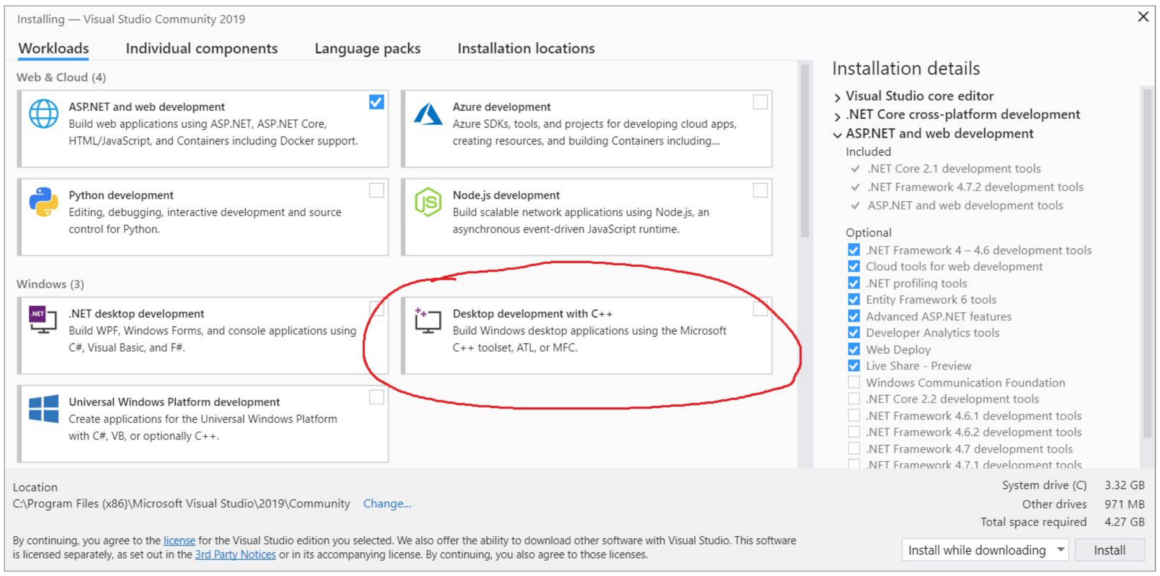
Instructions and Requirements

You must work on this assignment **alone**. (Start early)
Please "hand in" the following before class on the date due.

1. Screenshot of the output of program execution from both thread_example.cpp and your implementation in Go. The output should show the execution times in both C++ and Golang.
2. Source codes for part A and B. Here are the names to use: **partA.go, partB.go**.

If you do not have a C++ compiler installed, follow the instructions. **Note:** This is for Windows only, if you have MacOSX or Linux, you will have to find out how to get C++ on your system yourself. Because we are only looking for execution times, you will only have to run thread_example.cpp and take note of the execution time.

1. Go to this link: https://visualstudio.microsoft.com/downloads/?utm_medium=microsoft&utm_source=docs.microsoft.com&utm_campaign=button+cta&utm_content=download+vs2019+rc
2. Download the Community Version for Windows
3. Open the installer and check "Desktop development with C++"



Installing — Visual Studio Community 2019

Workloads | Individual components | Language packs | Installation locations

Web & Cloud (4)

- [✓] ASP.NET and web development — Build web applications using ASP.NET, ASP.NET Core, HTML/JavaScript, and Containers including Docker support.
- [ ] Azure development — Azure SDKs, tools, and projects for developing cloud apps, creating resources, and building Containers including...
- [ ] Python development — Editing, debugging, interactive development and source control for Python.
- [ ] Node.js development — Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Windows (3)

- [ ] .NET desktop development — Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F#.
- [ ] Desktop development with C++ — Build Windows desktop applications using the Microsoft C++ toolset, ATL, or MFC.
- [ ] Universal Windows Platform development — Create applications for the Universal Windows Platform with C#, VB, or optionally C++.

Installation details

- Visual Studio core editor
- .NET Core cross-platform development
- ASP.NET and web development
  - Included
    - ✓ .NET Core 2.1 development tools
    - ✓ .NET Framework 4.7.2 development tools
    - ✓ ASP.NET and web development tools
  - Optional
    - [✓] .NET Framework 4 – 4.6 development tools
    - [✓] Cloud tools for web development
    - [✓] .NET profiling tools
    - [✓] Entity Framework 6 tools
    - [✓] Advanced ASP.NET features
    - [✓] Developer Analytics tools
    - [✓] Web Deploy
    - [✓] Live Share - Preview
    - [ ] Windows Communication Foundation
    - [ ] .NET Core 2.2 development tools
    - [ ] .NET Framework 4.6.1 development tools
    - [ ] .NET Framework 4.6.2 development tools
    - [ ] .NET Framework 4.7 development tools
    - [ ] .NET Framework 4.7.1 development tools

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community Change...

System drive (C)	3.32 GB
Other drives	971 MB
Total space required	4.27 GB

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Install while downloading | Install

4. Keep the selected defaults for any further installation queries and install.

To import thread_example.cpp to your workspace in Visual Studio, go to File > New > Project. In the "New Project" menu, Installed > Visual C++ > Empty Project. Enter a name for the project and select OK. When created, go to Project > Add Existing Item, and add thread_example.cpp. You should be able to compile and run the program now.