Laboratory Goals / Objectives

Upon completion of the lab, the student should have a good understanding of the concepts involved with queues, a form of distributed message passing system.

Introduction/Background Reading

What is a queue and why is it useful?

A *message queue* is a key component of the message provider, designed to facilitate the passing of messages between different hosts. Let us contrast it with RMI in Java. Given an interface for a service running on another machine, represented by an object, our local machine can call methods on that object remotely. The returned data is received on our local machine. Normally, this happens almost instantly. When the program on our local machine calls the remote service's methods, our programs block and wait for the called method to return. Programs do not continue with execution until this has happened. **This is a synchronous, blocking approach**. It is synchronous because the server executes the method when called. It is blocking because our local program goes into the blocked state until the method call has returned. The client waits for the server execution completion. However, this approach is not always desirable. If a considerable amount of work is to be undertaken by the remote service for us, we may not want to wait for it to complete, especially if this work may take minutes, or even hours to complete, such as a complex mathematical calculation, a complex query on a large database, or resource heavy video or image processing. We may wish for our program to continue with other work, and get the returned results later. With message queues, this is possible.

With a message queue, the local program can create a message that contains data that we would like the remote service to process. We can then simply place this message into a queue, and then continue with other work immediately. The remote service, whenever it is ready, can take this message off the queue and carry out some processing on it that may take, in some cases, a considerable amount of time. Here, the local program placing the message into the queue is known as a *message producer*. The system running the remote service, which removes the message from the queue, is known as a *message consumer*. If the remote service needs to return a result to the local program, it can place another message with the result, into another queue, that can be retrieved by the client whenever it is ready. In this scenario with this other queue for sending data back to the local program, the remote service is the message producer and the local program is the message consumer. **This is an asynchronous, non-blocking approach.** It is asynchronous because the consumer may not receive the message at the time that it was placed into the queue by the producer. It is non-blocking because the producer does not enter the blocked state and wait for any response from the consumer, there is no waiting involved.

Another advantage should be clear here. RMI in Java is based on the concept of remote procedure calls (RPC), but RMI itself is a Java implementation of this concept. A queue is capable of being implementation independent. As long as a common, understood format has been agreed between the local program and the remote service for the messages in the queue, then the local program could be implemented in Java, and the remote service could be implemented in a .NET language such as C# or VB. This idea should also demonstrate another advantage of message queues, they promote loose coupling between different components of the distributed application.

To do:

Task 1
Create in Python (or Java) a FIFO queue that allows its clients to insert and to pop messages. It should also return the queue size when required and a true/false value when it's not or it is empty. Include comments with your code.

Task2
Consider a multi-producer, multi-consumer queue. This queue can be used in multithreading applications, where threads can exchange messages. However, threads cannot access the queue simultaneously. A lock mechanism makes sure that only one thread operates on the queue at a time.
Create in Python (or Java) using multithreading three identical consumer threads that do some processing on the message payload – at minimum they get a message from the queue, print some text, such as "Thread 2 work completed", introduce a delay and become available again. Use the queue to balance the load among all threads. Include comments with your code.

Links:

https://docs.python.org/2/library/queue.html
https://www.tutorialspoint.com/python/python_multithreading.htm

Questions
The following questions are to be filled in individually by each student and returned by the due date.

1. What is a message queue and what attributes can define it?

2. Provide the code for task 1 and screenshots that show how the queue works.

3. Explain your solution for task 2 – you may use a diagram. Provide the code for task 2 and screenshots that show how the system works.

4. Your additional comments on implementing and testing your code.