CST8221 – JAP
Lab 11
Implementing Producer/Consumer using Blocking Queues

Objectives

The purpose of the lab is to give you a hand-on experience of how to implement the Consumer/Producer relationship using classes from **java.util.concurrent** library (package).

The Nature of Things

You have already seen the low-level building blocks (synchronized primitive, object locks, etc.) that form the foundation of the concurrent programming in Java. You can write your multithreading programs using the low level primitives but it will be much easier and much safer to higher level structures that have been implemented by experienced concurrency expert. Many such high level structures are provided for you in the **java.util.concurrent** library. In this lab you are going to try some of them.

Many threading problems can be formulated elegantly and safely by using the Producer/Consumer relationship or design pattern. In a Producer/Consumer relationship, the producer component of an application generates data and stores it in a shared object. The consumer component of the application reads data from the shared object. The relationship separates the task of identifying the work to be done from the task involved in carrying out the work. One example of common producer/consumer relationship is something you use every day – print spooling. Another example is the "*burn buffer*" used by the CD/DVD recorders.

In multithreaded producer/consumer relationship, a producer thread generates data and places it in a shared object call **buffer**. A consumer thread read the data from the buffer. From what you already know it is clear that the relationship requires synchronization to ensure that the values are produced and consumed without damaging the data. All operation on mutable data that is shared by multiple threads (that is, that data in the buffer) must be carefully guarded with a locking mechanism to prevent data corruption. Operation on the buffer data are also state dependant, which means that the operation should proceed only if the buffer is in proper state. If the buffer is in a not-full state, the producer may produce. If the buffer is in non-empty state, the consumer may consume.

You can write a thread-safe buffer using low-level synchronization and locks. Fortunately, the Java concurrent library provides classes that implement thread- safe buffer. It is called **blocking queue**. A blocking queue causes a thread to block when you try to add an element when the queue is currently full or to remove an element when the queue is empty. Blocking queues are a very useful for coordinating the work of multiple threads. No explicit coordination is required on the part of the application programmer – the blocking queue is synchronized internally. In the Java *java.util.concurrent* package you can find a class called **ArrayBlockingQueue,** which is a fully functional thread-safe buffer class that implements the **BlockingQueue** interface.

In this lab you will try an application that demonstrates the use the ***ArrayBlockingQueue*** class**.**

Tasks

Download the ***CST8221_Lab11_code.zip*** file from Blackboard. Extract the contents. One code example is provided for you: ***BlockingQueueTest.java***.

The application demonstrates how to use a blocking queue (buffer) to control a set of threads. The application searches through all files in a folder (directory) and its subfolders (subdirectories) printing lines that contain a given keyword. A producer thread enumerates all files in all subfolders and places them in a blocking queue. Since the operation is relatively simple, it is very fast and the queue would fill up quickly with all the files in the file system if the operation is not blocked. The application also starts a large number of search threads. Each search thread takes a file from the queue, opens it, prints all lines containing the keyword, and then takes the next file. In order to signal the end of the work, the producer thread places a dummy file object into the queue. When a search thread takes the dummy, it puts it back and terminates.

Note that no explicit thread synchronization is required. In this application, the thread-safe queue data structure is used as a synchronization mechanism.

Run the examples and see how it works providing different base folders and keywords. Try to understand as much as possible how the programs work. Consult the Java documentation to find out what the different classes and methods are implementing.

If you have time try to display the results in a GUI. Modify your midterm test program and display the result in the text area.

Before the lab

Enjoy Java.

During the lab

Ask questions.

Before leaving the lab

Sign the attendance sheet.

After the lab

Remember what you have learned. You will need it soon.

Submission

No submission is required for this lab.

Marks

No marks are allocated for this lab, but do not forget that the joy of learning is priceless.

And do not forget that:

“A stepping-stone can be a stumbling block if we can't see it until after we have tripped over it.”
Cullen Hightower

but

“Americans will put up with anything provided it doesn't block traffic. “ Dan Rather