



MALMÖ HÖGSKOLA

CONCURRENT PROGRAMMING

ASSIGNMENT 2

THREAD SYNCHRONIZATION

A SIMPLE WRITER - READER SCENARIO CHARACTER TRANSFER / SIMPLE MEMORY GAME

MANDATORY

University Lecturer
Faculty of Technology and Society

A SIMPLE READER/WRITER APPLICATION

1 OBJECTIVES

The main goals of this assignment are:

- Handling communication between threads using a simple writer/reader pattern.
- Observing the behavior of non-synchronized threads.
- Learning how multiple threads can be synchronized when accessing a shared resource.

2 DESCRIPTION

This assignment consists of two alternatives. You should implement at least one of them:

1. Character transfer between two threads.
2. A simple memory game with at least two threads.

Implement a Writer/Reader pattern to solve this assignment, as illustrated in the following simple class diagram.



Both alternatives use the same technique with a writer class, a reader class and a third class, the character buffer as the shared object. For Java users the GUIs will be provided.

3 CHARACTER TRANSFER

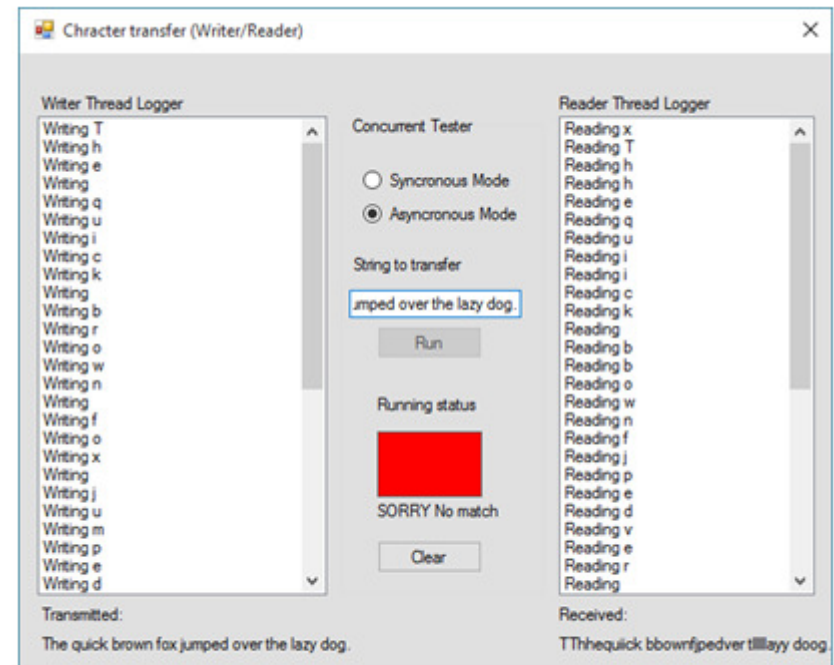
The application should let the user input a string for use as the text to be transferred from the input source (a text-field on the GUI) to the target source (a ListBox on the GUI). The writer thread writes the characters of the string (one by one) to the shared buffer as long as the buffer is not full. The reader thread reads the characters (one by one) from the shared buffer to a target as long as there are characters to read and the buffer is not empty.

The transfer should be done character by character with a random waiting interval between each character writing and character reading, in order to create a timespan long enough to watch each step on the output window; otherwise things will happen too quickly. The states of the process should be displayed to the screen for the user, as demonstrated in the run sample images (Demo 1a: and 1b). The user should be given the option of choosing to run the application in both synchronized and asynchronous modes. You may use any development tools and IDEs and any of the languages, C#, Java or C++.

Note: By **asynchronized**, here in this assignment, it is simply meant **not-synchronized**. Asynchronized programming will be discussed in its technical meaning later in this course.

3.1 The GUI based application

- 3.1.1 Create a GUI-based application and design the user interface with necessary input/out components so the user is able to



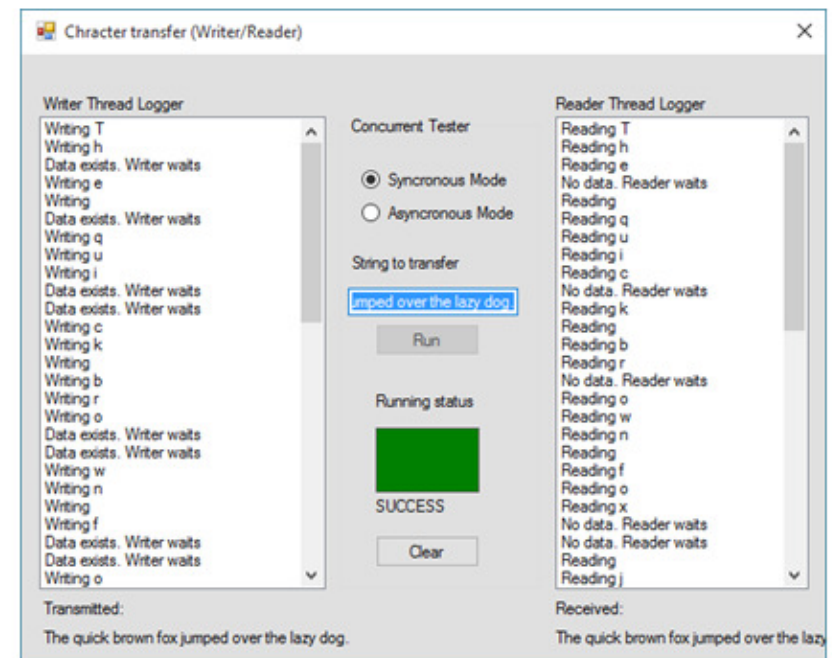
Demo 1a: Asynchronized

input a text string and run the application, with either synchronized or asynchronous option. The user should also see a log of the readings and writings as they happen.

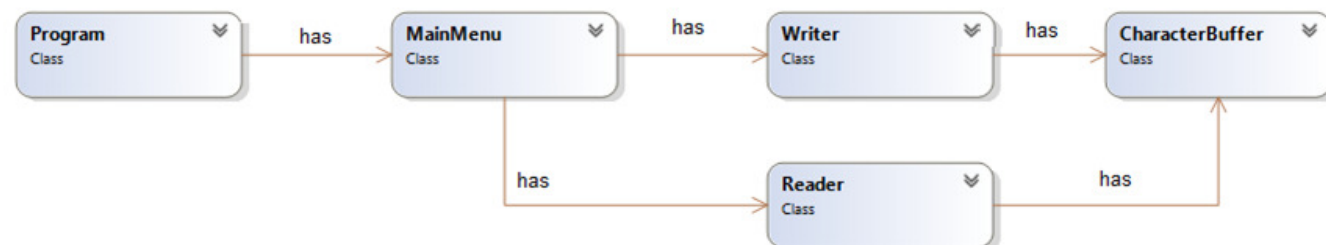
- 3.1.2 Create and initialize an instance of the above-mentioned classes in the user-interface class (MainForm in C#) or another start class.
- 3.1.3 Do the necessary programming to accomplish the task (see the run examples images to get an idea of how the application is expected to work).

3.2 Console based application

- 3.2.1 If you find GUI-based programming (in particular in C++) difficult, you may implement a console application. In this case you use a program structure as below can be applied.



Demo 1b: Synchronized



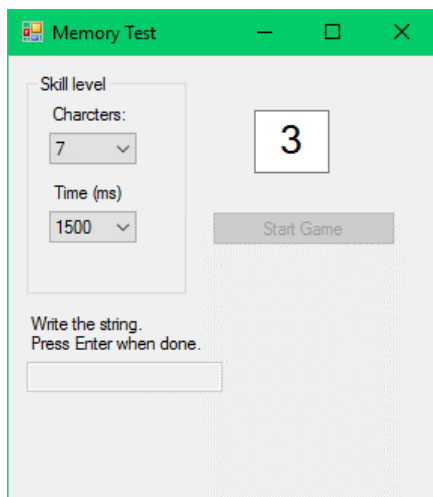
You can start synchronized/asynchronous processing.

4 SIMPLE MEMORY GAME

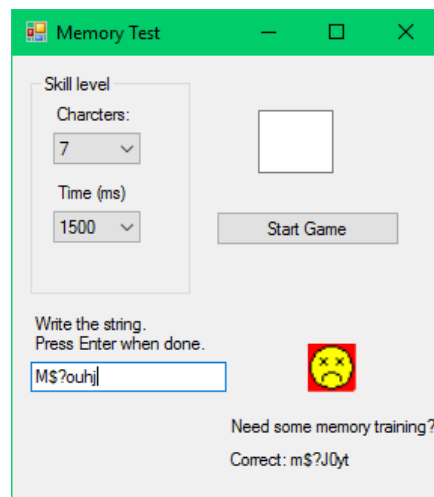
This assignment can only be done as a GUI based application. Same writer/reader structure sharing a common as in the above case is applicable for this alternative too. In this game, the application randomly selects a number of characters and displays them for a specific time span (ms). During this period, the user tries to memorize the characters and then rewrites the chars in correct order in a textbox.

The user select a hardship level by means of two variables, the number of characters and time span for each character to remain visible. The writer thread loops through the number of characters at the desired time and stores each character in the common buffer. The reader thread reads the characters one by one from the buffer and displays them on the GUI. The character can be all ASCII (alpha numeric) characters or only letters and digits, but they must be created randomly.

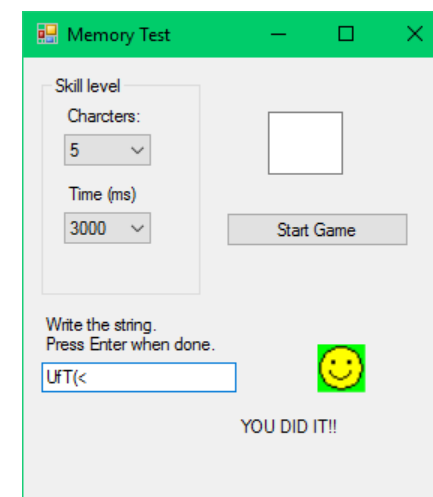
Remember to disable the button when the game is in progress and also erase the last character after the time is over. When all characters have been displayed, the threads stop, and the result text box gets enabled. The user should then rewrite the character sequence press enter at the end. The figures below shows a running session with two results.



Running session



Bad luck?



Nice work!

5 SPECIFICATIONS AND REQUIREMENTS FOR A PASS GRADE (G)

- 5.1 To qualify for a pass-grade, you should implement at least one of the above alternatives with good code quality.
- 5.2 Do your programming work well-structured, well organized and always have OOP in mind. Use proper variable and method names, document your code by writing comment in your code.
- 5.3 You may certainly bring changes in the application to make it more it more fun full-featured.
- 5.4 Test your application carefully before submitting.

6 GRADING AND SUBMISSION

Show your assignment to your lab leader during the scheduled hours in the labs, but before doing so, you must upload your work to Its' L. Make sure that you submit the correct version of your project and that you have compiled and tested your project before handing in. Be careful not to use any hard-coded file paths (for example path to an image file on your C-drive) in your source code. It will not work on other computers. Projects that do not compile and run correctly, or is done with poor code quality, will be returned for completion and resubmission.

Compress all the files, folders and subfolders into a Zip, Rar or 7z file, and then upload it via the Assignment page on It's L. Click the button "Submit Answer" and attach your file. Do not send your project via mail!

7 TIPS AND LINKS

C#: In general, avoid locking on a public type, e.g. `lock(this)`, or instances beyond your code's control. Best practice is to define a private object to lock on, or a private static object variable to protect data common to all instances.

```
private Object lockObj = new Object(); //instance variable

//lock block
lock (lockObj)
{
    //code
}
```

As we have not yet covered some features that make the work easier for this assignment, here is some guidelines:

To notify a thread in the waiting queue of a change in the locked object's state, the **Monitor.Pulse** in C# and **threadName.notify** in Java can be used. In addition, the following is copied from the Internet:

C#: The equivalent functionality (including the normal locking) is in the [Monitor](#) class (foo is the thread object):

```
foo.Notify() => Monitor.Pulse(foo)
foo.NotifyAll() => Monitor.PulseAll(foo)
foo.Wait() => Monitor.Wait(foo)
```

The lock statement in C# is equivalent to calling **Monitor.Enter** and **Monitor.Exit** with an appropriate try/finally block. The lock statement is recommended.

C++

Where you would have called `java.lang.Object.wait`, call `pthread_cond_wait` or `pthread_cond_timedwait`.

Where you would have called `java.lang.Object.notify`, call `pthread_cond_signal`.

Where you would have called `java.lang.Object.notifyAll`, call `pthread_cond_broadcast`

Useful Links:

C#:
Reader/Writer: <http://www.yoda.arachsys.com/csharp/threads/>
Reader/Writer: [http://msdn.microsoft.com/en-us/library/system.threading.readerwriterlock\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.threading.readerwriterlock(v=vs.110).aspx)
Lock: <http://msdn.microsoft.com/en-us/library/c5kehkc2.aspx>

Java: <http://www.journaldev.com/1037/java-thread-wait-notify-and-notifyall-example>
Reader/Writer: <https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReadWriteLock.html>

C++: <http://stackoverflow.com/questions/2085511/wait-and-notify-in-c-c-shared-memory>
Reader/Writer: <http://www.codeproject.com/Articles/598695/Cplusplus-threads-locks-and-condition-variables>

Good Luck!

Farid Naisan,
Course Responsible and Instructor

© Farid Naisan, farid.naisan@mah.se