Threading
Cpt S 321 Homework Assignment
Washington State University

Submission Instructions: (see syllabus)

Assignment Instructions:

Read all instructions *carefully* before you write any code.

In this assignment, you will build a new WinForms application that utilizes different aspects of threading and asynchronous methods. The two parts are two unrelated tasks, but put them into a single application. Design an interface with everything visible at once, such that part one is on the left half of the form and part two is on the right half. Whatever the interface layout is, it must allow the user to run both parts 1 and 2 at the same time (see the very end of the assignment describing the last of the 10 points).

Part 1: Asynchronous download from web (4.5 points)

Provide a way for the user to type a URL into a text box and you will download that file as a string from the web. When testing your app, the TA(s) will use a http:// web link (you don't have to worry about https://). Provide a single-line text box for the URL and a multi-line text box that stores the result.

Use the WebClient (https://msdn.microsoft.com/en-us/library/system.net.webclient.aspx) class. You will have to read a bit on MSDN to figure out how it works. Fetch the data from the web on a new thread so that the UI does not freeze while it is downloading. The WebClient class has asynchronous methods, but do not use these. Manually create a new thread and then call the synchronous download function. Remember that anything done on the non-UI thread will require you to Invoke (https://msdn.microsoft.com/en-us/library/system.windows.forms.form.invoke.aspx) (or BeginInvoke may be a better option) a method if you want an interface update at the end. Make sure of the following:

- Disable the URL text box, result-data text box, and start-download-button while the download is taking place. The text box and button controls have a Boolean "Enabled" property.
- Your interface must not freeze when large downloads are taking place. You *are* disabling interface components but your UI must still be responding. We went over the difference between an unresponsive interface and an interface with disabled components in class. You must be able to move and resize the window during the download.
- When the download completes, re-enable the UI components.
- Do NOT use busy-waiting loops or timers to check for download completion. The thread downloading the data from the web should notify the UI thread when it's done. The UI thread shouldn't have to be repeatedly checking if the download is done.

- For the text box that will display the downloaded data (as a string) set the Multiline property to true and WordWrap to true as well. Size the text box to take up a fairly large portion of the interface so it's easy to look at the downloaded string (which will be HTML data when the TA does their tests). The process of setting a large string as the text for a TextBox control may take a second or two. Since you cannot access the UI controls from a non-UI thread, setting the text must be done on the UI thread. A small pause in interface responsiveness is acceptable for this reason, but it must be only because of setting the text, not downloading the data from the web.

Part 2: Sorting a collection of lists in 2 ways (4.5 points)

Implement code that generates 8 lists of random numerical values (double, float, or int). Make sure each list has a decent number of items so the computation actually takes some time. 1,000,000 items for each is probably good. Have a button or menu item in your interface for kicking off the following actions:

1. Sort all 8 lists on a single thread, one after the other

2. Create a thread for each of the lists, so 8 threads if you have 8 lists, and sort each list on its own thread.

Measure the time taken for each of the two actions and report it somewhere in the interface. A multi-line text box should work fine for the report. Keep the following points in mind:

- To make it clear, you will do part 1 in its entirety (and measure the time) and after it completes do part 2 in its entirety (and measure that time as well). The idea is that your doing an (almost) identical task in two ways. One way uses a single thread and the other uses 8 threads. Based on the number of processors you have on your system, you may see a significant increase in speed if it can sort multiple lists in parallel.
- Make sure you use random lists at the beginning of each test. Don't sort all lists on one thread and then have the multi-threaded method just sorting already sorted lists.
- You can use the Stopwatch (https://msdn.microsoft.com/en-us/library/system.diagnostics.stopwatch.aspx) class for measuring the elapsed time of each method. Another option is to use DateTime:
    - DateTime start = DateTime.Now;
    - // Do computation
    - DateTime end = DateTime.Now;
    - // DateTime overloads – operator so elapsed time is: end – start
- Display the elapsed time in milliseconds. There should be two time values reported, one for the single-threaded method and another for the multi-threaded method.
- Use built-in sorting for each list. You don't need to implement quicksort or anything like that. The List class has a Sort (https://msdn.microsoft.com/en-us/library/3da4abas(v=vs.110).aspx) method.

- Disable interface components while the sorts occur, make sure the interface is still responding during the sorting, and re-enable interface components after the last of the 8 threads completes. You'll have to figure out a way to reliably determine when the last of the 8 threads completes so that you can invoke a call on the UI thread.
- You don't have to display the contents of the lists in the interface, just the computation times.

Remaining 1 point:

This comes pretty much for free if you do the two above parts correctly, but your app must support downloading data from the web while also running the sorting tests concurrently. So make sure that when you start the download from web that you only disable interface components relating to part 1 and the stuff for part 2 stays enabled so that you can start the sorting while the download happens. The same should be true the other way around so if you start the sorting tests first you can then start a download that will run concurrently.

Notes:

- There are many different things that can be used to achieve parallel execution in .NET. You're free to experiment with these all you want, but you must use the Thread class for this assignment and know how to use it for the exams.
- The image below shows what your interface might look like. You should be able to click both buttons and move around the window while they're processing their tasks.