

Course MS2957

Additional Lab: Managing Threads

In this lab you will modify single-threaded applications to use threading to improve responsiveness.

You will need to download *ThreadingExercise.zip* and extract the contents.

Exercise 1

Console application

In this exercise you will start with a console application which accesses five web URLs and lists the page size in each case. The web requests are sent sequentially. You will modify the application to use threads to send the requests concurrently.

Examining the starter project

1. Start Visual Studio and open *ThreadingExercise\Starter\ThreadingExercise.sln*.
2. Open *Program.cs* in the *ThreadingExercise* project.
3. Note that the code has a method *GetPageSizes* which initialises an array of strings defining a set of URLs. For each URL a method *GetPageSize* is called. This method sends a web request and returns the page size as an integer. Each URL and the associated page size are printed to the console in *GetPageSizes*.
4. Run the application. Note the order in which the results are listed. Compare this with the order in which the URLs are stored in the array.
5. Set a break point at the return statement in *GetPageSize*. Start debugging the application. When the execution stops at the breakpoint, open the **Debug > Threads** window.

QUESTION: What thread is executing this method?

Adding multi-threading

1. Add references to the *System.Threading* and *System.Runtime.Remoting.Messaging* namespaces.
2. Add a declaration for a delegate *GetPageSizeDelegate* before the declaration of class *Program*. This delegate should take a string parameter and return an int.
3. Add a declaration for a class *Data* within the file *Program.cs*. *Data* should simply contain two public variables, *url* (string) and *pageSize* (int).
4. Modify the foreach loop in *GetPageSizes* so that it does the following:
 - a. Does not increment the variable count or print any output to the console
 - b. Creates a new *GetPageSizeDelegate* to call the *GetPageSize* method.

- c. Creates a *Data* instance and set the value of its *url* field to the current *url* value in the loop.
 - d. Invokes the delegate asynchronously, passing the current *url* value as a parameter, specifying a method *Callback* as the asynchronous callback, and passing the *Data* instance as a state object.
- 5. Add new void method *Callback* with an *IAsyncResult* parameter. Add code to this method to do the following:
 - a. Retrieve the state object and cast it to a *Data* reference.
 - b. Retrieve the delegate and cast it to a *GetPageSizeDelegate* reference.
 - c. Set the *pageSize* field of the *Data* instance to the result of calling *EndInvoke* on the delegate.
 - d. Increments the variable count and prints the *url* and *pagesize* fields of the *Data* instance and the value of count. The value of count may be modified by multiple threads, so place these statements in a block protected by lock.
- 6. Run the application. Note the order in which the results are listed. Compare this with the order in which the URLs are stored in the array.
- 7. Set a break point at the return statement in *GetPageSize*. Set another breakpoint anywhere within *Callback*. Start debugging the application. When the execution stops at the breakpoints, open the **Debug > Threads** window.

QUESTION: What thread is executing each of these methods? What thread is executing *GetPageSizes*?

Exercise 2

Windows Forms application

In this exercise you will start with a Windows Forms application which accesses the same five web URLs as in exercise 1 and lists the page sizes in a test box. Again, in the starter project the web requests are sent sequentially. You will modify the application to use threads to send the requests concurrently.

Examining the starter project

1. Start Visual Studio and open *ThreadingExercise\Starter\ThreadingExercise.sln*.
2. Open *Form1.cs* in the *ThreadingExercise2* project and view the code.
3. Note that the code is very similar to exercise 1, except that the code previously in *GetPageSizes* is now in the event handler for the button, and the output is appended to the *results* text box

4. Run the application. Note the order in which the results are listed. Compare this with the order in which the URLs are stored in the array.

QUESTION: What happens if you click the button before the results are complete?

5. Set a break point at the return statement in *GetPageSize*. Start debugging the application. When the execution stops at the breakpoint, open the **Debug > Threads** window.

QUESTION: What thread is executing this method?

Adding multi-threading

1. Add a declaration for a delegate *GetPageSizeDelegate* before the declaration of class *Program*. This delegate should take a string parameter and return an int.
2. Modify the application to support multi-threading. The modifications will be similar to those in Exercise 1, with the following variations to work within a Forms application:
 - a. Declare a *SynchronizationContext* instance variable in the class *Form1*.
 - b. In the button event handler, set the *SynchronizationContext* field to *SynchronizationContext.Current* and clear the *results* text box.
 - c. In *Callback*, instead of printing the result, create a new *SendOrPostCallback* instance to call a method *UpdateResult*. Call the *Send* method of the *SynchronizationContext* field and pass it the *SendOrPostCallback* instance and the string containing the result string. (Note: you can only update UI controls from the Main thread.)
 - d. Add a new void method *UpdateResult* which takes an *object* as its parameter. Add code to this method to cast the *object* parameter to a string reference and append this to the *results* textbox.
3. Run the application. Note the order in which the results are listed. Compare this with the order in which the URLs are stored in the array.

QUESTION: What happens now if you click the button before the results are complete?

4. Set a break point at the return statement in *GetPageSize*. Set another breakpoint anywhere within *Callback*. Set another breakpoint at the return statement in *UpdateResult*. Start debugging the application. When the execution stops at the breakpoints, open the **Debug > Threads** window.

QUESTION: What thread is executing each of these methods? What is the role of the *SynchronizationContext*?