

Advanced Programming – Application Description

Tobias Forve

Student number: S00205070

Prof. Dana Vasiloaica

Trier, 18.5.2020

Foreword

Goal of the lecture was it to write an application that deals with typical multithreading features. To achieve this the application was written in C-Sharp using the System.Threading library.

Table of Contents

2. In general .……………………………………………………………………………….4

2.1 Used programs and libraries ………………………………………………...4

2.2 Windows Forms elements used …………………………..4

3. Umsetzung……………………………………………………………………………….6

3.1 Wang Tile……………………………………………………………………………..6

3.2 Berechnen des Index…………………………………………………………………..6

3.3 Umsetzung im Code…………………………………………………………………..7

4. Fazit……………………………………………………………………………………...9

2. In general

2.1 Used programs and libraries

Programs used are visual studio 19 with windows forms and .NET 4.7.2. Main additional Libraries needed to achieve the goals of the application are System.Threading, System.IO and System.IO.IsolatedStorage, as well as the Windows Forms Libraries (System.Windows.Forms).

2.2 Windows Forms elements used

The Application consists of a Form Form1.cs which holds the other GUI elements of the Windows Form, three labels label1 label2 and label3 just to label parts of the GUI. Two ComboBoxes, one called cb\_TypeOfThread which holds options to choose from how to handle the main task of the application, the other called cb\_ThreadCount that stores an integer which is used as a reference how many threads to create. It also holds a NumericUpDown num\_Timer to set up an integer between 0 and 2000 to set the seconds the threads have to run. The last 3 elements are a button btn\_StartStop to start the threads and a FlowLayoutPanel mainPanel which also includes another label totalTime to display important UI updates and processes.

3. Implementation

First task is to select one option in the ComboBox cb\_TypeOfThread followed by the decision to choose a fix amount between one and four of threads to handle the task. In the NumericUpDown num\_Timer you have to decide how much time the threads have to process. Min time is 0 seconds max 2000 and the time will be split between the threads. After put in all relevant data running the Start button will lead in to one of two possible options as followed.

3.1 “Threads” – Option 1

Choosing the first option “Threads” in the ComboBox will lead to the WaitThread.cs Class, especially to the void Start() method in which one ThreadStart() will be raised calling the DoWork() method. In the DoWork() method an array of WaitHandle will be initialized with the amount of Threads to run to handle the synchronization of the upcoming threads and handle shared resources.

Inside of the for-loop a new EventWaitHandle object called handle will be created with an initialState of false to handle further thread queues.

After that the time to process will be divided by the amount of threads to split the time between them. Between one and four anonymous threads will be raised to count down the time. handle.Set() set the EventWaitHandle state to signalized to give waiting threads the signal to also proceed.

With WaitHandle.WaitAll() it is insured that all WaitHandle objects signalized their status.

At the end the OnThreadEvent will raise an event to set the status of ThreadeventArgs, Status 1 means everything worked fine and the threads have finished, 2 means the threads were interrupted by the user clicking the Button again and calling the startStop\_Click() method in Form1 and status 3 means an error was occurred.

3.2 “Background Worker” – Option 2

Choosing the second option “Background Worker” in the ComboBox and pressing the start button will create WaitBW background workers and pass over the already divided time. The amount of background workers is based on the int set in the cb\_ThreadCount.

In the constructor of WaitBW the WorkerReportsProgress and the WorkerSupportsCancellation properties will be set to true so the progress updates can be reported and asynchronous cancellation will be supported. Also the DoWork, the RunWorkerCompleted and ProgressChanged event handler will be added to the listener.

When running the DoWork() method will calculate the elapsed time and also reports the progress to the ProgressChanged handler. It will also check if the CancellationPending value and if it will be set to true it set e.Cancel to true, break out of the DoWork() and abort the background worker.

Whenever the ProgressChanged() is called it will also call the UpdateGlobalProgress() method. In here a synchronization object syncObj will be locked to call the UpdateGlobal() method from the FormsUpdater.cs just once. The method it self will update the totalTime label with the combined time elapsed in ms.

3.3 Other classes

3.3.1 FormsUpdater.cs

FormsUpdater.cs inherited from the ICallBackBW interface make sure to display the correct texts in the the btn\_StartStop and the label totalTime, activate and deactivate the ComboBoxes and NumericUpDow and calles the WriteToIso() method in the Form1.cs.

3.3.2 IsoStorageHandler.cs

In the IsoStorageHanlder.cs the application creates a path to the isolated storage in the OS to save data in a text file. In this case it will write the time the backgroundWorkers took to complete the task.