Problem Set 3: Threads, Delegates, Events, and Invoke

The goal of this assignment is to design a Windows application that will produce a list of all prime numbers between 2 and a given n, $2 \leq n \leq 1000000$.

Sieve of Eratosthenes (276-196 BC):

We start with the integers beginning with 2, which is the first prime. To calculate the rest of the primes, we start by filtering the multiples of 2 from the rest of the integers. This leaves a list beginning with 3, which is the next prime. Now we filter the multiples of 3 from the rest of that list. This leaves a list beginning with 5, which is the next prime, and so on.

In other words, this algorithm works like this. First, write down a list of integers

\[ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16 \ 17 \ 18 \ 19 \ 20 \]

Then mark all multiples of 2:

\[ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16 \ 17 \ 18 \ 19 \ 20 \]

Move to the next unmarked number, which in this case is 3, then mark all its multiples:

\[ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16 \ 17 \ 18 \ 19 \ 20 \]

Continue in this fashion, marking all multiples of the next unmarked number until there are no new unmarked numbers. The numbers that survive this marking process (the Sieve of Eratosthenes) are primes.

\[ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16 \ 17 \ 18 \ 19 \ 20 \]

Goal of this assignment:
Problem 1
Implement the Windows application “Eratosthenes.exe” that implements the sieve algorithm. The main form must provide:

1. A TextBox to specify the greatest integer number to be analyzed,

2. A read-only TextBox to print the primes in decreasing order,

3. A Button labeled “Filter” to start the filtering process,

4. A Button labeled “Cancel” to abort the filtering process,

5. A ProgressBar to visually indicate the progress of the filtering process,

6. A StatusBarPanel to display the number of analyzed integers, and
7. A StatusBarPanel to display the number of primes found.

To implement the sieve do not use an array of integers. Find a way to implement the filtering process without constructing a static array of integers. Remember, a declaration

\[ \text{int[]} \text{ numbers} = \text{new int [1000000]}; \]

allocates a 4Mbyte heap object.

The filtering process must be implemented in a separate Thread. That is, you must create a filtering thread that runs concurrently to the Main Thread. To update the Main Form, use delegates and events. Since your application will have at least two Threads, you need to check inside the event handlers of the Main Form, whether to you need to call Invoke or not. Invoke executes a delegate on the Thread that owns the control's underlying window handle!

Use the TextChanged event of the TextBox control (1) to trigger parsing of the string in the TextBox's Text property. The string should denote a valid integer. Use Int32.parse for this purpose. The Filter Button should only be enabled, if the TextBox control contains a valid integer within the range of 2 and 1000000.

The Click event of the Filter Button starts the filtering process. While the filtering process is running the Filter Button must be disabled and the Cancel Button must be enabled. When the filtering process terminates (i.e., the corresponding Thread terminates), enable the Filter button again and disable the Cancel Button.

To be able to observe the progress of the filter process, suspend the filter Thread for 200 milliseconds after each round.

Please note that the above specification does not include all requirements. You need to deduce the missing information from context properties. For example, the Invoke method requires a specific protocol. Moreover, it does not make much sense to be able to resize the Main Form. This should actually not be possible at all.

Warning: Visual Studio .NET associates a binary resource with each Form class. The name of that binary resource is the same as the first type name in a C# compilation unit. Do not add any definitions before Form classes, as this may results in a compile-time error.
Submission deadline: Monday, October 3, 2005, 4:10 p.m.
Submission procedure: on paper in class (.cs files only) and electronically using the turnin-hw3 script, which is located in ~cs430x/public/bin. Please use the printout of the submission confirmation email as cover page and check the problems that you have solved.

In order to submit your homework solutions, go (using your CS UNIX account) into the directory that contains your solution (i.e., C#-source files and all related project files). In that directory run the command “~cs430x/public/bin/turnin-hw3”. After a successful submission, you will receive a confirmation email. Before the due date, you can resubmit your solutions as often as you like.

On the department's Windows XP systems you can use the command csc to compile C#-programs. However, it is recommended to use Visual Studio .NET, because most assignments require some GUI work.