CSCI 428/628 - Advanced Web Programming - Assignment: Multithreading

02/20/2013 - draft

Overview

For this assignment you will write a mobile device application that will perform a computationally intensive and lengthy process in the background while leaving the user interface thread free to respond to other user-initiated (and other) events. The computational process will be to find all prime numbers less than a given number. Depending on the platform you may be asked to do this computation in just one or in a user-specified number of threads (from 1 to 4). See Details below for the platform you will develop for. The purpose is not to devise the quickest or most efficient way to come up with the list of primes, but rather to move the computation into the background and also to illustrate and measure the speed of the computation using different numbers of threads, different maximum numbers, and different hardware platforms.

Background

You will recall that a prime number is a number that cannot be divided evenly by any numbers other than itself and 1. So, for example, 5 is prime because it can only be divided by 5 and 1. 6 is not prime since it can be divided by 2 and 3 as well as by 6 and 1. 1 itself is not considered prime. The prime numbers less than 25 are: 2, 3, 5, 7, 11, 13, 17, 19, and 23. Prime numbers are very interesting mathematically. The largest known prime has about 16 *million digits*. Google it and you will find out (too many) more amazing facts. Anyway, for this program we will look for primes in a smaller range - maybe all primes less than a maximum of few thousand. The algorithm you employ must be correct, but does not have to be the smallest or fastest - we do want this to take a while.

You should write the code yourself (why rob yourself of the chance to actually code something and debug it?). If you choose not to and just copy or adapt code from cyberspace, you **must clearly** cite the source of that code in the **program documentation**. Failure to do so is considered plagiarism by NIU and you would be deemed responsible for academic misconduct.

Here are a couple of rough verbal descriptions of algorithms you could implement (*maxInt* is the largest number to be considered for prime-ness):

Method 1 (Sieve of Eratosthenes)

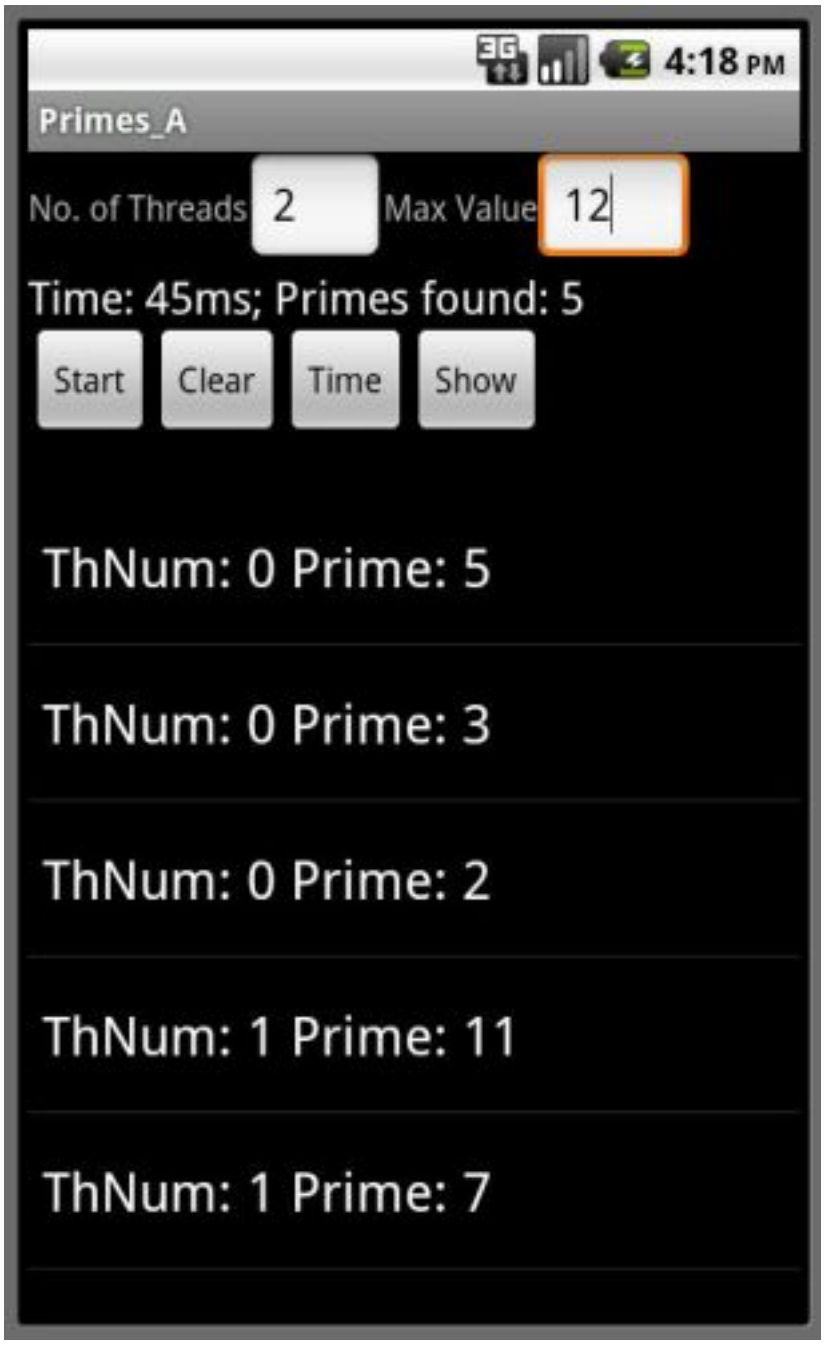
- create an array of maxInt booleans and initialize each to true.
- for each number 2 to sqrt(maxInt), "cross out" all the multiples of that number (i.e. set them to false - they are not prime) in the array. So ar[4] = false, ar[6] = false... and then ar[6] = false, ar[9] = false...
- when you are done, those array elements that are still true are primes (do not report 0 or 1 as prime).

Method 2:

- for each number maxInt down to 3 (or 3 up to maxInt)
- try dividing it by 2 and then 3 and then 4 and then 5, etc. up to sqrt(maxInt). (Or you can go down rather than up. Same difference.)
- If at any point there is a 0 remainder, quit with this number (it is not prime) and move on to the next number.
- If you never get a 0 remainder in this process (for this number) it is prime - report it (store it in an array, or something).

In either case (or using any other algorithm you may devise or find), check your results with several maxInts you can easily verify - say 10, 25, 50, and 100. Be sure that 2 is always reported, but not 1. Be sure that the largest prime less than or equal to maxInt is reported. Be sure there are no extra numbers (that are not prime) or missing primes. You might want to write a small test program just to test the correctness of your algorithm before you start to worry about threads.

Details



Here is a screen shot of a fairly ugly (Android) app showing the results of running the program with two threads. Your layout could be better, but will show at least the elements included here.

Start: will clear any text left over from a run and will then create and run the calculation thread(s). While the calculation thread(s) are running, it will update that top text area with some or all of the intermediate results: a thread number and a prime found. If you find that this frequent updating of the UI thread is a problem, then show only every 10th or 50th result here.

After all thread(s) have completed, this area will show the elapsed time in milliseconds for the process to complete, and the number of primes found as you can see here.

Clear: will clear the various text fields.

Time: will display the current date and time and should work even while a primes calculation is in progress.

This is proof that the UI thread is still responsive.

Show: will normally be pressed after a calculation is complete. You should disable it while a calculation is running. It will display a list of all the primes found (not necessarily in order, especially if more than one Thread is involved - although you could sort the results). This list will also report the thread number that found the prime.

A small mobile device is not the best place to try to show hundreds or thousands of items. The Show command, depending on your platform, the number of items to display, and the method used to display them may take many seconds to complete and may cause a warning message or a shutdown of the app. We will not worry too much about this - the point of the program is to keep the UI system responsive *while the computation is in progress*. What happens after that is less of a concern.

Other Notes

1. If you are asked to support a variable number of threads to perform the computation, you could divide up the range into 1, 2, 3, or 4 equal sub-ranges from 1 to maxInt. Or you could devise another scheme to divide up the computational task. Your program will need to know when all the threads have completed so that you can report the results and enable the *Show* button and the data input fields (described earlier but not shown in the picture above).

2. If you decide to store the found primes in an array, you will need to dimension that array. There is an approximate formula for the number of primes less than a given number, n. It is (in Java):

```
numPrimes = (int)(n/Math.log(n));  // natural (base e) logarithm
```

This is not exact, and you will find - sooner or later - if you use this number you will get a program termination or other ugly problem due to array overflow (in Android, ArrayOutOfBoundsException). To avoid this, take the value calculated and increase it by 25%. If you prefer, you could store results in a data structure that can grow dynamically.

Platform Implementation Notes

Android

Implement the primes calculation by extending the AsynchTask class. Periodically update the UI with the progress of the computation in some reasonable way so the user can see that something is happening in the background - but don't do this for every number checked! When the computation is finished, communicate the set of primes found to the UI *Activity* and display the results, including the elapsed time.

In order to display the results quickly, you could have your activity extend ListActivity and use an ArrayAdapter to help manage the array display. The list itself then would be an array of Strings. However, you may use other methods or classes if you prefer - just ensure that the end display for a reasonably large set of primes is fairly quick (a second or two).

iOS

Implement the primes calculation using Grand Central Dispatch. Get references to the default priority global concurrent queue using `dispatch_get_global_queue()` and the main queue using `dispatch_get_main_queue()`. For example

```
dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
```

You'll also need a way to store your list of primes (an `NSMutableArray`, for example).

The code to compute the primes for a given subset of the range of numbers to check can be passed as a block to the concurrent queue using `dispatch_async()`. For example

```
dispatch_async(queue, ^{
    // code for block
    ...
});
```

When you need to update the UI with the progress of the computation or you want to add a prime number you've found to the your list of prime numbers, you can pass the code to do so as a block to the main queue (once again using `dispatch_async()`). Since blocks placed in the main queue will execute in a serial fashion, you don't need to worry about synchronizing the list of primes. When a thread finishes, you can can update a counter variable using a block passed to the main queue. Once the counter becomes equal to the number of threads, all threads have completed and you can enable the "Show" button to allow the list of primes to be displayed.

The list of prime numbers found may simply be displayed in a `UITextView`.

You can get the current date and time as an `NSString` using the message expression

```
[[NSDate date] description]
```

The message expression

```
[NSDate timeIntervalSinceReferenceDate]
```

can be used to get start and stop times for the primes calculation. The return value for this message is `NSTimeInterval`, which is effectively a `double` number. This means you can subtract the start time from the stop time to find the elapsed time in seconds and display it.

Windows Phone

You will implement two versions of this program. In the first version, only use one BackgroundWorker thread to perform the computation. Attach event handlers for the BackgroundWorker's events: WorkerSupportsCancellation, WorkerReportsProgress, ProgressChanged, and RunWorkerCompleted.

In your prime calculation code, check to see if there is a CancellationPending for the thread. If so, cancel the thread.
In the thread's ProgressChanged event handler advance a progress bar according to the ProgressChangedEventArgs percentage value.
In the Completed event handler, report the number of primes found and display the found primes.

In the second version, provide a textbox into which one can enter the number of threads (1-4) to use in finding the primes. Do not worry about the events you worked with in version 1. Simply start the threads and let them do their work. You will need 1 - 4 AutoResetEvent objects that will allow you to signal when each thread has completed its work and thus you know it is time to display the number of primes found and the list of primes on the UI.