Teaching Threads with VB.NET

Dr. Qi Yang
Department of Computer Science and Software Engineering
University of Wisconsin at Platteville
Email: YangQ@uwplatt.edu
Phone: 608-342-1418

Abstract

VB can be used to create GUI programs easily. Since VB.NET is now a fully object-oriented language, providing access to many classes, including Thread, Mutex, ManualResetEvent, ReaderWriterLock and Monitor, it can be used to teach threads and related concepts.

Windows 2000 and other operating systems support multiple threads within a single process. Threading and other related concepts are not easy to understand by the students at the beginning. Examples include process, thread, mutual exclusion, deadlock, starvation, semaphore and monitor. Many classic problems have been created to demonstrate the problems and solutions in Operating Systems, such as Barber Shop, Dining Philosophers and Reader-Writer with or without FIFO queuing. Some programming assignments will help the students to understand the basic Operating Systems concepts. We will show three VB programming assignments in the paper.

Threads in VB.NET

Threads can be created easily in VB.NET. Assume that System.Threading is imported, the following statement declares a reference to a thread:

```
Dim p1 As Thread
```

In order for the thread to do something, a Sub should be defined. Assume Sub ProcessOne has been defined, the following two statements create a thread object and start the thread:

```
p1 = New Thread(AddressOf ProcessOne)
p1.Start()
```

Our first example program has two threads. Both threads generate a value and use the value to update a global variable Total. The following variables are used to implement mutual exclusion and control the threads:

```
Dim theSem As New Mutex()
Dim p1_paused, p1_cancelled As Boolean
Dim p1_Wait As New AutoResetEvent(True)
Dim p2_paused, p2_cancelled As Boolean
Dim p2_Wait As New AutoResetEvent(True)
```

The following is the pseudo code for both threads (i = 1, 2). Method WaitOne() of class Mutex (object theSem) is equivalent to the P operation, and ReleaseMutex() is equivalent to the V operation of a semaphore. Method WaitOne() of object pi_Wait (class `AutoResetEvent`) blocks the current thread until method Set() is called.

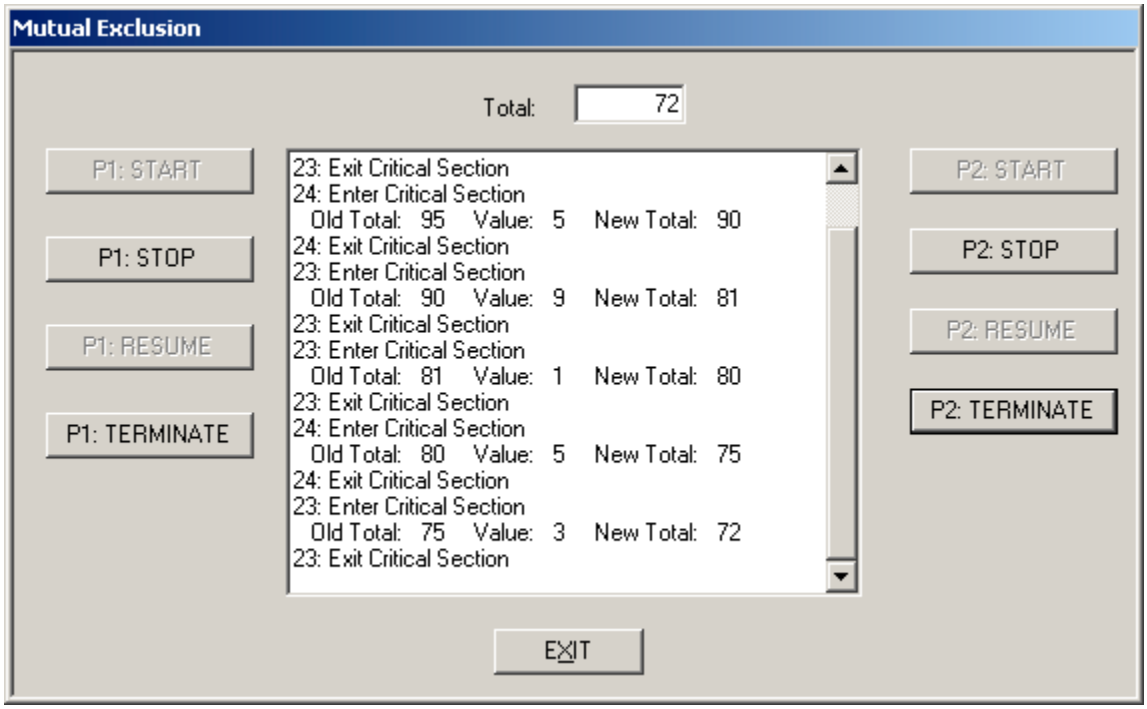
```
While Not pi_cancelled
   If pi_paused
      Wait (pi_Wait.WaitOne())

   If pi_cancelled
      Exit While (terminate the thread)
```

```
Generate a Value
Set Semaphore (theSem.WaitOne())
Update Total
Release Semaphore (theSem.ReleaseMutex())
```

The GUI interface of the program is shown below. The event procedures of the buttons are straightforward by using the methods of the thread object and variables defined above:

START : create a thread object and call the Start method of the object
TERMINATE : set pi_cancelled to True and the thread will be terminated later
STOP : set pi_paused to True
RESUME : set pi_paused to False and call pi_Wait.Set() to wakeup the thread



Reader-Writer without FIFO

The second VB program is a simulation of the Reader-Writer problem without FIFO queuing. Two classes are created: Reader and Writer. Each class has a thread object as a private member, some properties to set parameters for the thread, one private Sub that will be executed by any object of the class, and some public methods to start and terminate the thread. A public variable of class ReaderWriterLock is defined in a module and is accessible from both classes:

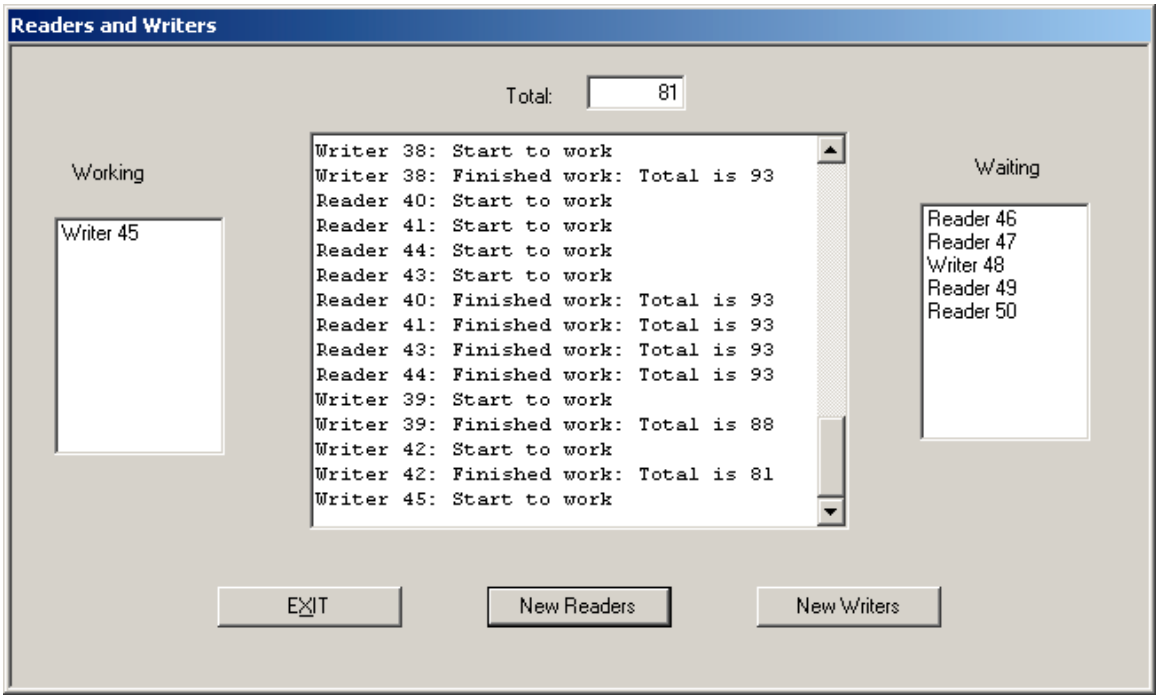
```
Public theLock As New ReaderWriterLock()
```

The pseudo code for Reader and Writer processes is

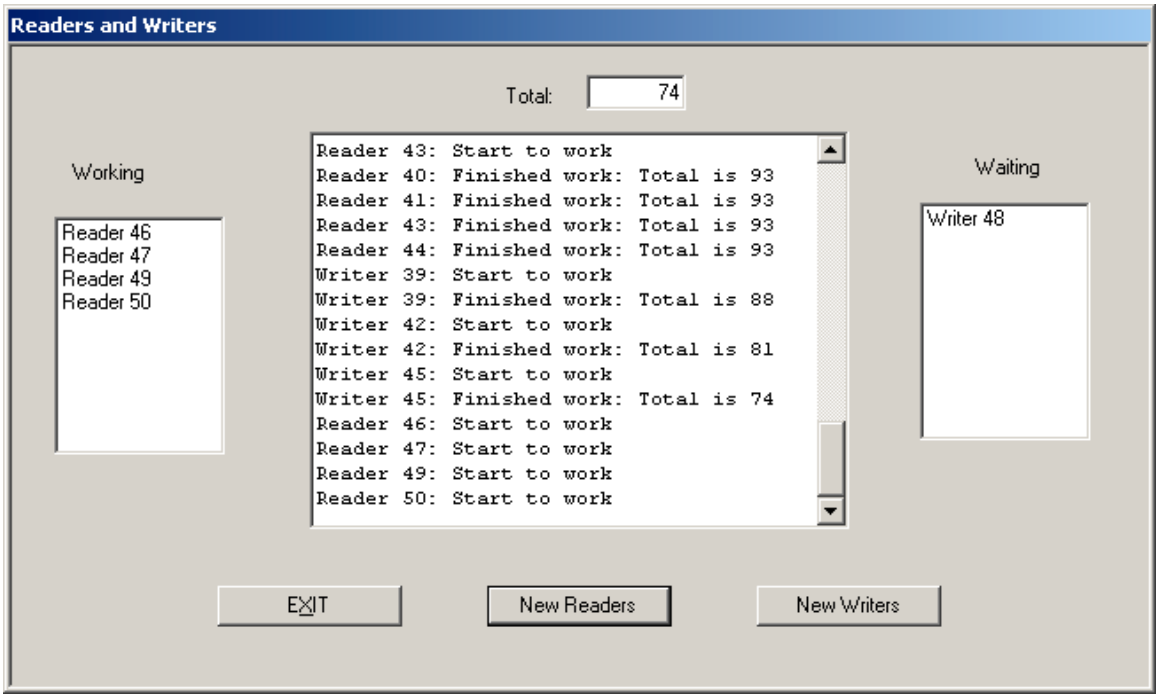
```
Reader
  Ask for read permission (theLock.AcquireReaderLock())
  Read and do work
  Release the lock (theLock.ReleaseReaderLock())

Writer
  Ask for write permission (theLock.AcquireWriterLock())
  Generate value and update Total
  Release the lock (theLock.ReleaseWriterLock())
```

The GUI interface of the program is shown below. Clicking on New Readers or New Writers will generate a new Reader object or Writer object, and the thread of the new object will be started. We can see that writer 48 arrives before readers 49 and 50 but after readers 46 and 47 while Writer 45 is working.



After Writer 45 is done, Reader 46 and 47 can read data, but Writer 48 has to wait. Because FIFO is not enforced, Reader 49 and 50 can also read data while Writer 48 is waiting.



Reader-Writer with FIFO

To enforce the FIFO rule, the following variables are added to the module to replace the ReaderWriterLock:

```
Public MeFIFO As New Mutex()    'Control FIFO Queue
Public MeData As New Mutex()    'Control Data access
Public MeRC As New Mutex()      'Control Reader Count
Public RC As Integer            'Reader Count
```

The Reader and Writer process Subs are modified accordingly:

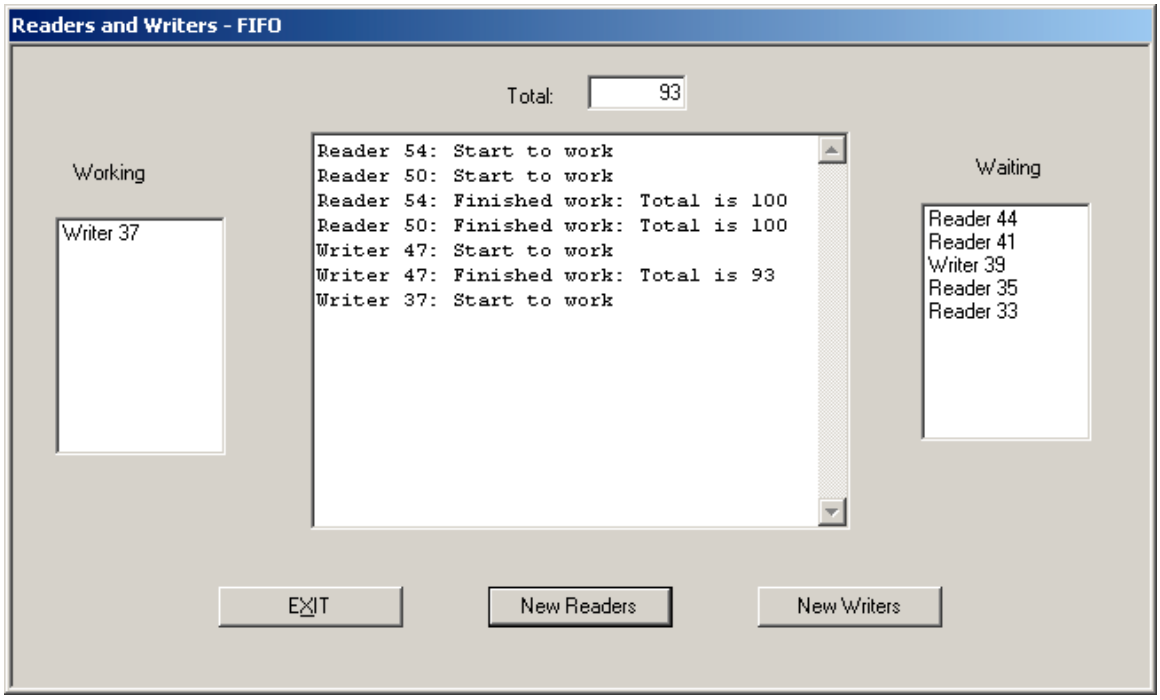
```
Reader
   MeFIFO.WaitOne()
   MeRC.WaitOne()
   RC += 1
   If RC = 1
      MeData.WaitOne()
   End If
   MeRC.ReleaseMutex()
   MeFIFO.ReleaseMutex()

   Read data
```

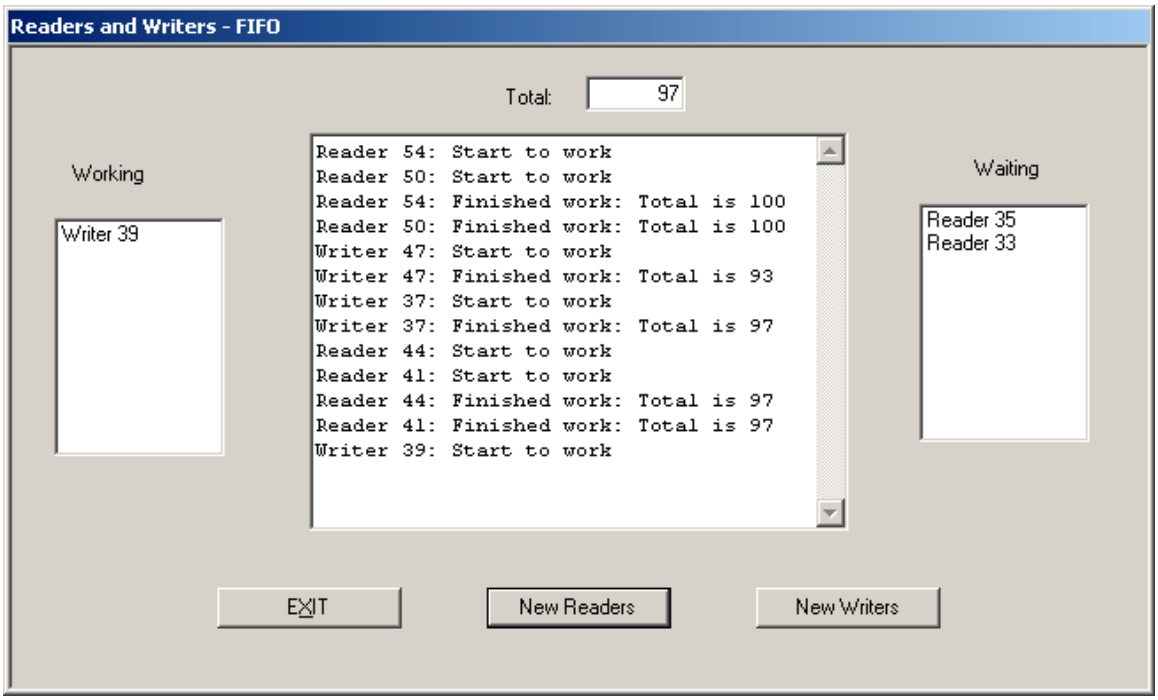
```
    MeRC.WaitOne()
    RC -= 1
    If RC = 0
       MeData.ReleaseMutex()
    End If
    MeRC.ReleaseMutex()

Writer
    MeFIFO.WaitOne()
    MeData.WaitOne()
    MeFIFO.ReleaseMutex()
    Generate a Value and update Total
    MeData.ReleaseMutex()
```

In the following picture, while Writer 37 is working, Writer 39 arrives after Reader 44 and 41, but before Reader 35 and 33.



After Writer 37 is done, Reader 44 and 41 start to work, and Writer 39 and Reader 35 and 33 are all waiting. After Reader 44 and 41 are done, Writer 39 start to work, and Reader 35 and 33 are still waiting.



Summary

Programming threads in VB.NET is straightforward, and the GUI interface can show the result clearly. VB.NET becomes an excellent choice as the language for programming assignments in an Operating Systems course, and such assignments will definitely help students to understand the concepts.