CHAPTER 22

Introduction to Asynchronous Programming

- Processes, Threads, and Asynchronous Programming
- Parallel Loops
- The BackgroundWorker Class
- Asynchronous Programming Patterns
- BeginInvoke and EndInvoke
- Timers
Processes, Threads, and Asynchronous Programming

In this chapter, we’re going to introduce four methods you can use to add multithreading to your programs. This chapter is a bit different from the previous chapters in that it goes beyond just the language features. Instead, we’ll also include classes from the BCL and include some programming techniques. In spite of the fact that these things are a bit beyond just the language features, I want to do this because it’s imperative that we as programmers increase our use of multiprocessing in our code—and I think a first book on C# is a good place to start.

When you start a program, the system creates a new process in memory. A process is the set of resources that comprise a running program. These include the virtual address space, file handles, and a host of other things required for the program to run.

Inside the process, the system creates a kernel object, called a thread, which represents the actual executing program. (Thread is short for “thread of execution.”) Once the process is set up, the system starts execution of the thread at the first statement in method Main.

Some important things to know about threads are the following:

- By default, a process contains only a single thread, which executes from the beginning of the program to the end.
- A thread can spawn other threads so that at any time, a process might have multiple threads in various states, executing different parts of the program.
- If there are multiple threads in a process, they all share the process’s resources.
- It’s threads, not processes, that are the units scheduled by the system for execution on the processor.

All the sample programs shown so far in this book have used only a single thread and have executed sequentially from the first statement in the program to the last. This is called synchronous programming. Asynchronous programming refers to programs that spawn multiple threads, which are, at least conceptually, executed at the same time. (They might not actually be executed at the same time.)

If the program is running on a multiprocessor system, the different threads might actually be executing at the same time on different processors. This can considerably improve performance, and as multicore processors become the norm, we need to write our programs to take advantage of this opportunity.

On a single-processor system, though, clearly only one instruction can be executed by the processor at a time. In this case, the operating system coordinates the threads so that the processor is shared among them. Each thread gets the processor for a short time, called a time slice, before being kicked off the processor and sent to the back of the line. This round-robin sharing of the processor lets all the threads work their ways through the code.