CHAPTER 20

Introduction to Asynchronous Programming

What Is Asynchrony?
The Structure of the async/await Feature
What Is An async Method?
Async Operations in GUI Programs
Using an async Lambda Expression
A Full GUI Example
The BackgroundWorker Class
Parallel Loops
Other Asynchronous Programming Patterns
BeginInvoke and EndInvoke
Timers
What Is Asynchrony?

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 the thread executing 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 is 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. There are many situations, however, where this simple model produces unacceptable behavior, in either performance or end-user experience.

For example, a server program might be constantly initiating connections with other servers and requesting data from them, while at the same time processing the requests from many client programs. These communications tasks usually require a fair amount of time where the program is just waiting for a response from another computer on the network or on the Internet. This significantly decreases performance. Instead of just wasting this time waiting for a response, it would be more efficient to work on other tasks in the meantime, and then resume working on the first task when the reply arrives.

Another example would be an interactive GUI program. If the user initiates an operation that takes a significant amount of time, it’s unacceptable for the program to freeze on the screen until the action completes. The user should still be able to move the window around on the screen, and maybe even cancel the operation.

In this chapter we’re going to look at asynchronous programming, which is a type of programming where portions of a program’s code aren’t necessarily executed in the strict order in which the code is written. Sometimes this involves running a section of code on another thread. Other times, however, no new thread is created, but instead, the execution of the code is reordered to make better use of the single thread’s capacity.

We’ll start by looking at a new feature introduced in C# 5.0 that allows you to build asynchronous methods. It’s called the async/await feature. After that, we’ll look at several features that are part of the .NET Framework, but not built into the C# language, that allow additional forms of asynchrony. These topics include the BackgroundWorker class and an introduction to the .NET Task Parallel Library. Both these topics implement asynchrony by creating new threads. We’ll finish the chapter by looking at other ways of producing asynchrony.