实 验 七　程序进程与线程

7.1　实验目的

重点理解程序进程与线程的联系与区别，程序运行中的同步与异步概念。掌握 Windows 平台中的进程对象，进程的属性与参数，编程实现进程启动及其输入输出重定向的实现，程序的同步与异步调用实现。

7.2　程序进程与线程

在 Windows 平台运行程序时首先创建进程描述块（又叫进程控制块），进程控制块包含进程需要的各项资源，进程仅是资源分配的单位。用户编写的程序代码串行存储成磁盘文件，程序运行时系统依据主程序入口创建主线程，工作线程由主线程或其它线程创建，所有线程并发执行，文件中串行的代码经克隆后在机器中并行执行。程序进程与线程的关系由图7-1表示。

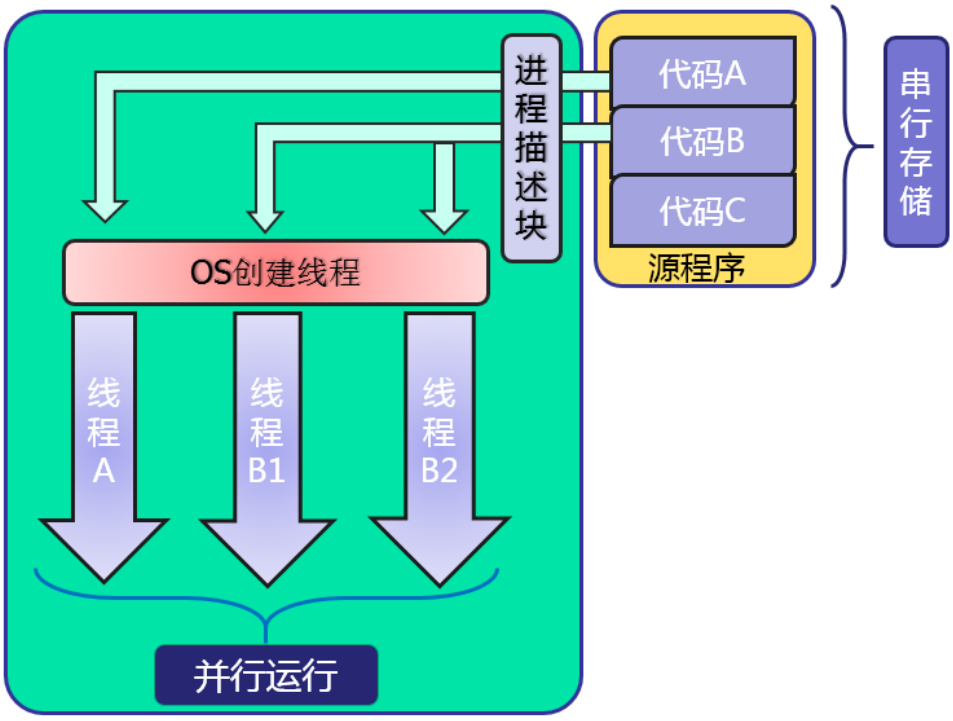

图 7-1 程序进程与线程

Windows 平台 CPU 运行调度的最小单位是线程，所有可运行的线程排成队列，图7-2描绘 CPU 以时间片轮转的方式执行队列中的线程。

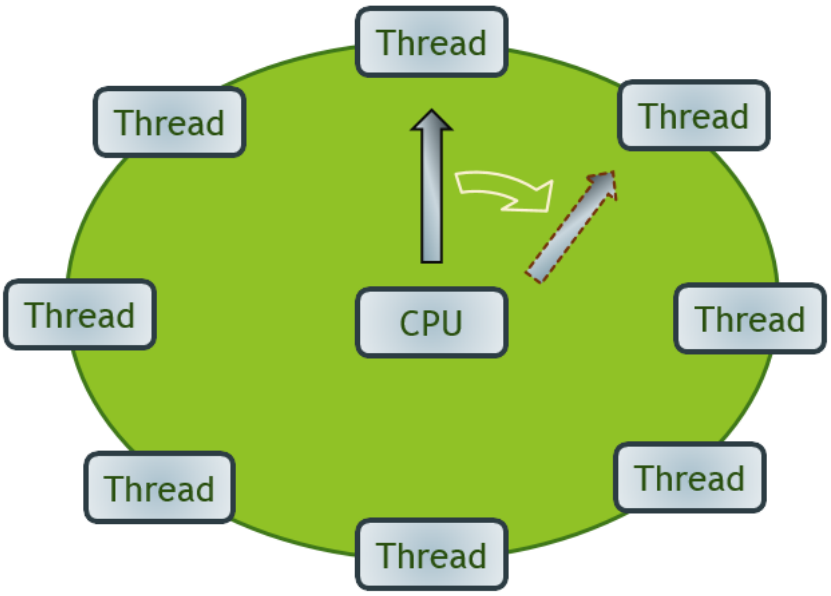

图 7-2 CPU 轮转与线程的并发执行

7.3 程序运行中的同步与异步概念

Windows 平台运行时有大量的系统线程，应用程序生成的线程与系统线程并发执行，当用户线程调用系统函数时系统线程执行任务并生成结果。用户线程对目标结果有两种依赖方式，要求目标结果必须出现的系统调用即是同步调用，仅有函数调用无须目标结果出现的系统调用即是异步调用。函数调用与数据传递在多个线程之间基于数据传递带来的前趋制约

单独线程内的代码或函数是顺次执行前后代码间形成前趋制约，线程间形成同步运行。多个并发运行的线程如果能够前趋制约，线程推进速率不互相牵制即形成异步运行，图7-3描绘线程中的同步调用与异步调用。

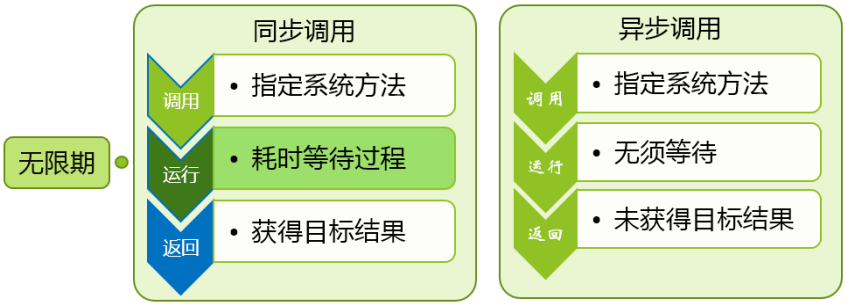

图 7-3 系统函数的同步调用与异步调用

.NET 框架类中很多的方法同时支持同步与异步方式，比如 FileStream 类提供对文件流操作，有同步读写操作，也支持异步读写操作。网络通信程序中 Socket 类即支持同步的 Accept 方法，也支持异步方式的 BeginAccept 方法用于接受一个传入的连接，网络数据接收即具有同步的 Receive 方法又有异步的 BeginReceive 方法等。操作系统提供的多种服务调用以驱动线程或系统线程的形式与用户线程异步运行，同步调用与异步调用对用户界面线程影响较大，例如网页浏览器采用的同步方式果获取网页内容，会产生明显的停滞现象，支持异步方式的 Ajax 技

术会更受用户的欢迎，浏览器在获取网页的内容时，部分区域内容的更新不影响其它区域的显示及响应。

Windows 平台为进程最先创建的线程即是主线程，工作线程由其它线程创建，线程之间并发执行。对于通常的窗体应用程序，主线程即是窗体界面线程，专门负责用户的输入并绘制结果，所有控件事件处理函数都在主窗体线程中。主线程的同步调用可能造成界面停滞，造成极差的用户体验。耗时运算或系统调用宜采用异步调用方式，例如一个聊天进程它即有用户窗体，又通过系统线程接收来自网络的不可确切预计到达时间的数据，采用异步回调方式接收数据是非常必要的。

同步调用与异步调用主要区别是对目标结果的处理方式，同步调用没有获得目标结果的情况下不会返回，程序流程相对简单但线程会进入挂起状态。异步调用会很快返回，线程不会进入挂起状态，目标结果由回调函数处理。同步读写磁盘时由于磁盘调度等不能马上完成，或者网络数据没有到达，线程将一直处于挂起状态。例如在包括网络通信功能的线程中调用 Connect 方法时，同步调用方式线程会一直挂起等待用户连接；通信对方不发送数据情况下线程同步方式调用 Receive 方法，即使经过很长时间都收不到目标数据，也不能认定程序出错；线程在这段长时间不仅处于挂起状态，而且由于线程不能继续执行后续代码而无法终止，这是同步调用最大的缺点。

回调函数是异步方法调用必须使用的一个参数，它由完成指定任务的系统线程在目标数据就绪后执行，回调函数由用户定义却属系统线程而不属调用线程。调用线程与系统线程推进速度不同，目标数据的产生在时间上不可预计，调用线程通常采用循环结构或同步对象，图片 7-4绘制了基于数据处理的线程同步与异步运行的程序结构。异步执行提供程序最大的并发特性，程序响应更灵活，是成熟软件必备要求。

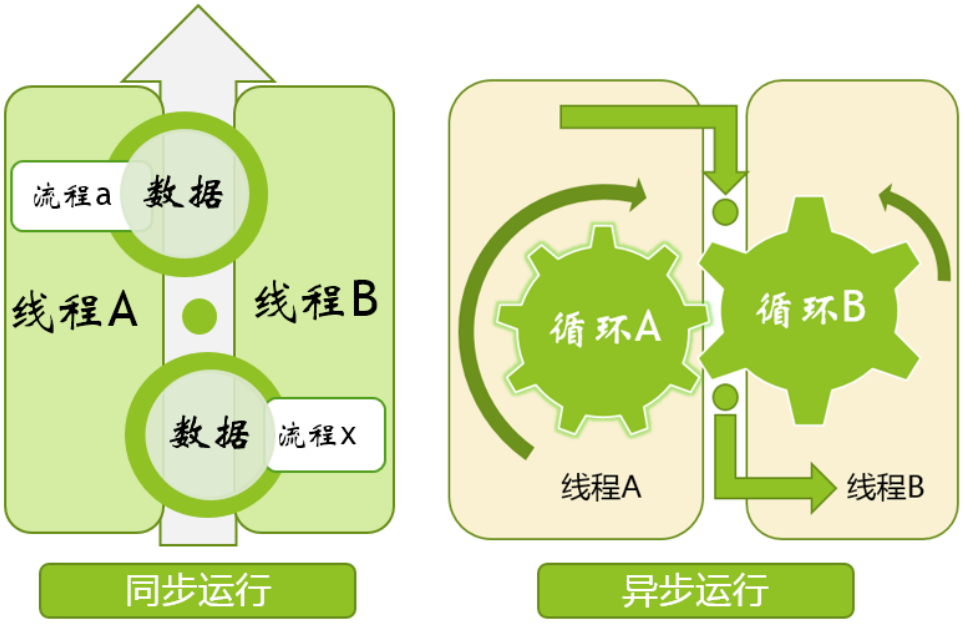

图 7-4 线程的同步与异步运行

7.4 控制台进程的重定向

Windows 中的控制台程序和 Linux/Unix 中的进程类似，操作系统为每个进程创建三个文件描述符，标准输入（stdin）、标准输出（stdout）和标准错误输出（stderr）如图7-5。默认情况下键盘、鼠标是输入文件，屏幕作为输出文件。重定向和管道可更改输入输出描述符，进程从指定文件的内容或另一个应用程序的输出提供文本，或者将输出返回到文件或其他设备。在.NET 平台中，Process 类可以获取系统正在运行的进程的列表，或者可以启动新的进程。Process 类与 ProcessStartInfo 类一起使用可更好地控制启动的进程。ProcessStartInfo 类用于配置进程使用的值，例如程序的命令行参数，要启动的进程名，进程使用的窗口状态。通过设置 ProcessStartInfo 类对象的 RedirectStandardInput、RedirectStandardOutput 和 RedirectStandardError 属性为 true，还要将进程对象的 UseShellExecute 设置为 false，能够在启动控制台程序时重定向进程的输入和输出。

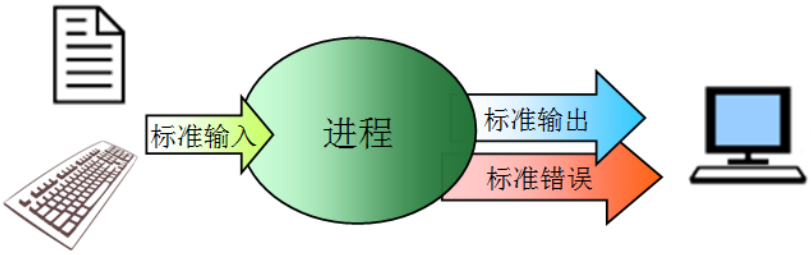

图 7-5 进程的输入输出文件

本实验通过 ping 命令来演示对控制台进程的输入输出重定向功能，参考界面设计如图7-6：

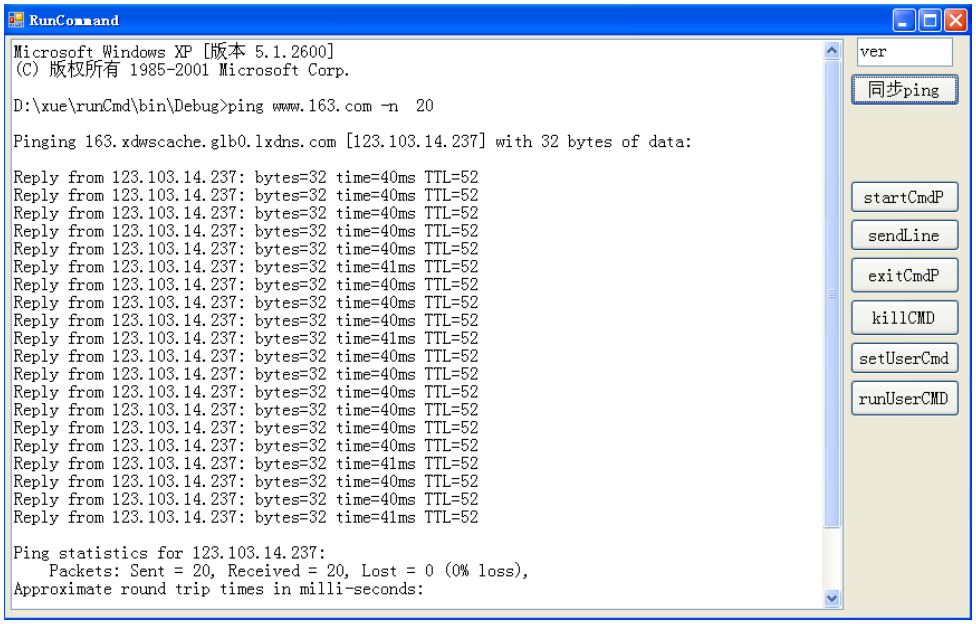

图 7-6 控制台进程 ping 输入输出重定向界面设计

7.4.1 同步读取方式

同步读取在程序逻辑上比较简单，设置新创建进程重定向属性，如进程文件名称，重定向流等，启动进程后，向重定向后的输入流中写入数据，并从进程的输出流读出数据。结合界面，在" 同步 ping" 按钮中编写代码，实现 ping 功能参考代码如下：

```
Process process = new Process();
process.StartInfo.FileName = "cmd.exe";
// 是否使用外壳程序
process.StartInfo.UseShellExecute = false;
// 是否在新窗口中启动该进程的值
process.StartInfo.CreateNoWindow = true;
// 重定向输入流
process.StartInfo.RedirectStandardInput = true;
// 重定向输出流
process.StartInfo.RedirectStandardOutput = true;
//使 ping 命令执行二十次
string strCmd = "ping www.163.com -n 20";
process.Start();
process.StandardInput.WriteLine(strCmd);
process.StandardInput.WriteLine("exit");
// 获取输出信息
textBox2.Text = process.StandardOutput.ReadToEnd();
process.WaitForExit();
process.Close();
```

这种方式动行的控制台进程与窗体进程会以同步的方式执行，窗体进程在启动控制台进程后将等待进程结束，这段时间窗体无法响应，用户点击窗体出现没有响应提示，程序应避免这种实现方案。

7.4.2 异步读取方式

窗体程序适合异步响应方式，工作线程或创建的子进程的运行不影响窗体响应；需定义进程对象，以及被重定向后的输入输出流对象：

```
public static Process cmdP;
public static StreamWriter cmdStreamInput;
private static StringBuilder cmdOutput = null;
```

添加一个成员函数 StartCmd，此函数设置要启动进程的输入输出重定向，为了实现对进程输出流的异步接收，需要向进程对象的 OutputDataReceived 成员添加 DataReceivedEventHandler 类型的回调函数，StartCmd 函数参考代码如下：

```
private void StartCmd(string cmdFile)
{
    if (cmdP!=null)
```

```
        {
            if (!cmdP.HasExited)
            {
                //停止未结束进程，
                cmdP.Close();
            }
        }
        //根据进程文件名创建进程对象
        cmdP = new Process();
        //设定进程名
        cmdP.StartInfo.FileName = cmdFile;
        cmdP.StartInfo.CreateNoWindow = true;
        cmdP.StartInfo.UseShellExecute = false;
        //重定向输入输出流
        cmdP.StartInfo.RedirectStandardOutput = true;
        cmdP.StartInfo.RedirectStandardInput = true;
        cmdOutput = new StringBuilder("");
        //输出数据异步处理事件
        cmdP.OutputDataReceived +=
            new DataReceivedEventHandler(strOutputHandler);
        //异步处理中通知您的应用程序某个进程已退出
        cmdP.EnableRaisingEvents = true;
        cmdP.Start();
        //重定向进程输入流，其它方法将向此流中写入数据
        cmdStreamInput = cmdP.StandardInput;
        //开始异步输出的读入
        cmdP.BeginOutputReadLine();
    }
```

控制台进程对象在执行 cmdP.Start() 代码后就会启动进程，执行 cmdP.BeginOutputReadLine() 方法后进程输出数据就会以异步的方式开始运作，这个异步方法不会有等待，因此不会影响窗体界面响应。异步数据接收要添加的回调函数名为 strOutputHandler，这个函数属用户自定义，但是函数的参数列表则是固定的，当控制台进程有数据输出时，就会调用这个回调函数，传入的 outLine 参数对象将携带产生的数据，strOutputHandler 的参考代码如下：

```
    private static void strOutputHandler(object sendingProcess,
        DataReceivedEventArgs outLine)
    {
        //将每次输出产生的数据附加到结果字符串中
        cmdOutput.AppendLine(outLine.Data);
        //设置输出文本框内容
        SendMessage(main_whandle, TRAN_FINISHED, 0, 0);
```

```
}
```

strOutputHandler 函数需要将新数据产生通知到窗体以便界面更新，使用了方法 SendMessage，它属内核函数，需要作下面的声明：

```
//动态链接库引入
[DllImport("User32.dll", EntryPoint = "SendMessage")]
private static extern int SendMessage(
IntPtr hWnd, // handle to destination window
int Msg, // message
int wParam, // first message parameter
int lParam // second message parameter
);
```

SendMessage 的功能是向特定窗体发送消息值，以待窗体处理，要发送的消息值 TRAN_FINISHED 是自定义窗体消息，需作如下声明：

```
public const int TRAN_FINISHED = 0x500;
```

使用 SendMessage 函数需要主窗体和文本控件的 handle 值，需要定义如下的变量：

```
public static IntPtr main_whandle;
public static IntPtr text_whandle;
```

在窗体的 Load 事件中添加下面的代码，取得窗体和文本框对象的 handle 值：

```
main_whandle = this.Handle;
text_whandle = textBox2.Handle;
```

窗体对象要能够接收和处理用户自定消息，需要对 DefWndProc 函数进行重载，参考代码如下：

```
protected override void DefWndProc(ref Message m)
{//窗体消息处理重载
    switch (m.Msg)
    {
        case TRAN_FINISHED:
        textBox2.Text = cmdOutput.ToString();
            break;
        default:
            base.DefWndProc(ref m);
        break;
    }
}
```

回调函数 strOutputHandler 实际是以工作线程的方式执行，工作线程被限制为不能直接调用窗体控件，因此还不能在这个函数中直接设置 textbox 的文本属性，在些向主窗体发送消息值代表进程产生了数据，设置 textbox 控件的文本是在 DefWndProc 函数实现中的。利用 textbox 控件输出文本有两种方法，方法一是 textbox 控件在显示文本时可使用控件方法 AppendText 来添加文本内容，ScrollToCaret 方法可使文本控件滚动到文本输入点，这部分由学生自行完成；方法二步骤比较多，但能通过这个方法深入了解 windows 控件的特性。数据更

新时给 textbox 控件的 text 属性赋新值来显示结果，这种方法没有使文本滚动的功能调用，如果输出信息较多时会略显得不够方便，可通过向控件发送 WM_VSCROLL 消息来实现这个功能；由于给文本框赋新值频繁更新，这引起文本框的闪动，解决方法是控制其重绘属性，并调用 RedrawWindow 内核函数定制其绘制过程，RedrawWindow 函数的声明如下：

[DllImport("User32.dll", EntryPoint = "RedrawWindow")]

public static extern bool RedrawWindow(IntPtr hWnd, IntPtr prect, IntPtr hrgnUpdate, uint flags);

要定义的消息常量如下所示，消息常量的值来自 Windows 的 SDK：

public const int WM_VSCROLL = 0x0115;

public const int SB_BOTTOM = 0x0007;

public static int WM_SETREDRAW = 0x0B;

自定义的消息处理逻辑参考代码如下：

```
SendMessage(text_whandle, WM_SETREDRAW,0,0);
textBox2.Text = cmdOutput.ToString();
//文本框控件无滚动事件，可向文本框发送滚动消息，令其文本滚动为最后
SendMessage(text_whandle, WM_VSCROLL, SB_BOTTOM, 50);
    SendMessage(text_whandle, WM_SETREDRAW,1, 0);
RedrawWindow(text_whandle, IntPtr.Zero, IntPtr.Zero, 1 | 4 | 128);
```

7.4.3 进程的异步执行

进程的同步或异步执行都可完成设定任务，在用户的输入与程序输出的交互性方面，异步执行将更符合用户习惯。

为了能深刻对比同步与异步执行效果，编辑一个名为 pp.bat 的批处理文件，进程运行时执行不会终止的 Ping 任务。采用同步执行方式，用户界面将不会有任何响应，发生“假死”现象，而异步方式则能很好运行。批处理文件内容如下：

```
@echo off
ping www.163.com -t
@echo on
```

在窗体中添加 openFileDialog 对象，在 setUserCmd 按钮添加适当代码，当用户点击 setUserCmd 按钮后，选择 pp.bat 批处理文件，确定后将文件名保存到变量中，点击 runUserCMD 按钮后调用 StartCmd 函数启动批处理文件，代码从略。pp.bat 批处理文件中的命令使用的是无限循环参数，因些这个进程将不会终止。对于控制台程序，任务执行完可自动终止进程；程序运行未自行终止时在用户操作状态下，可使用 CTRL+C 键盘组合命令来强制终止进程，在程序中，可使用 cmdP.Kill() 方法来杀死进程，暂时没有其它方法在其代码没有执行完毕时强行终止。

7.5 作业

1. 调用 getmac 获取网卡 mac

2. 调用 shutdown 命令关闭或重启电脑