

2 程序进程与进程间通信

PRINCIPLE OF WINDOWS AND ITS APPLICATIONS

School of CS

Jicheng Hu

jicheng @ yahoo . com

<https://gitee.com/wuhanuniversity/>



内容提要 - 程序进程与进程间通信



2.1 程序与进程



2.2 进程间通信机制简介



2.3 进程间通信-消息机制



2.4 进程间通信-重定向机制



2.5 进程间通信-管道

2.1 Program and Process

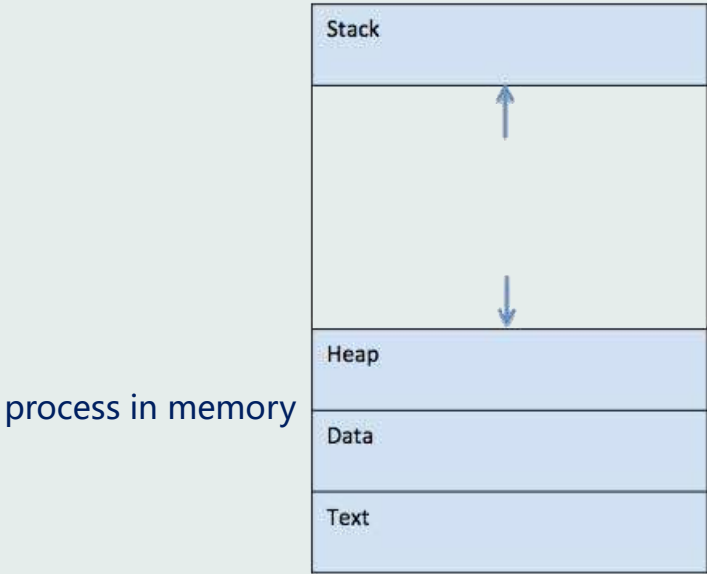
<https://www.infoworld.com/>

A process is an instance of a program in execution.

进程是执行中的程序

创建一个进程后，操作系统就将程序的一个副本装入计算机内存中，然后启动一个线程执行该程序

Linux 进程在内核眼中是什么样子的
<http://news.eeworld.com.cn/mp/rrgeek/a84417.jspx>



| | Component & Description |
|---|---|
| 1 | Stack The process Stack contains the temporary data such as method/function parameters, return address and local variables. |
| 2 | Heap This is dynamically allocated memory to a process during its run time. |
| 3 | Text This includes the current activity represented by the value of Program Counter and the contents of the processor's registers. |
| 4 | Data This section contains the global and static variables. |

操作系统中的进程

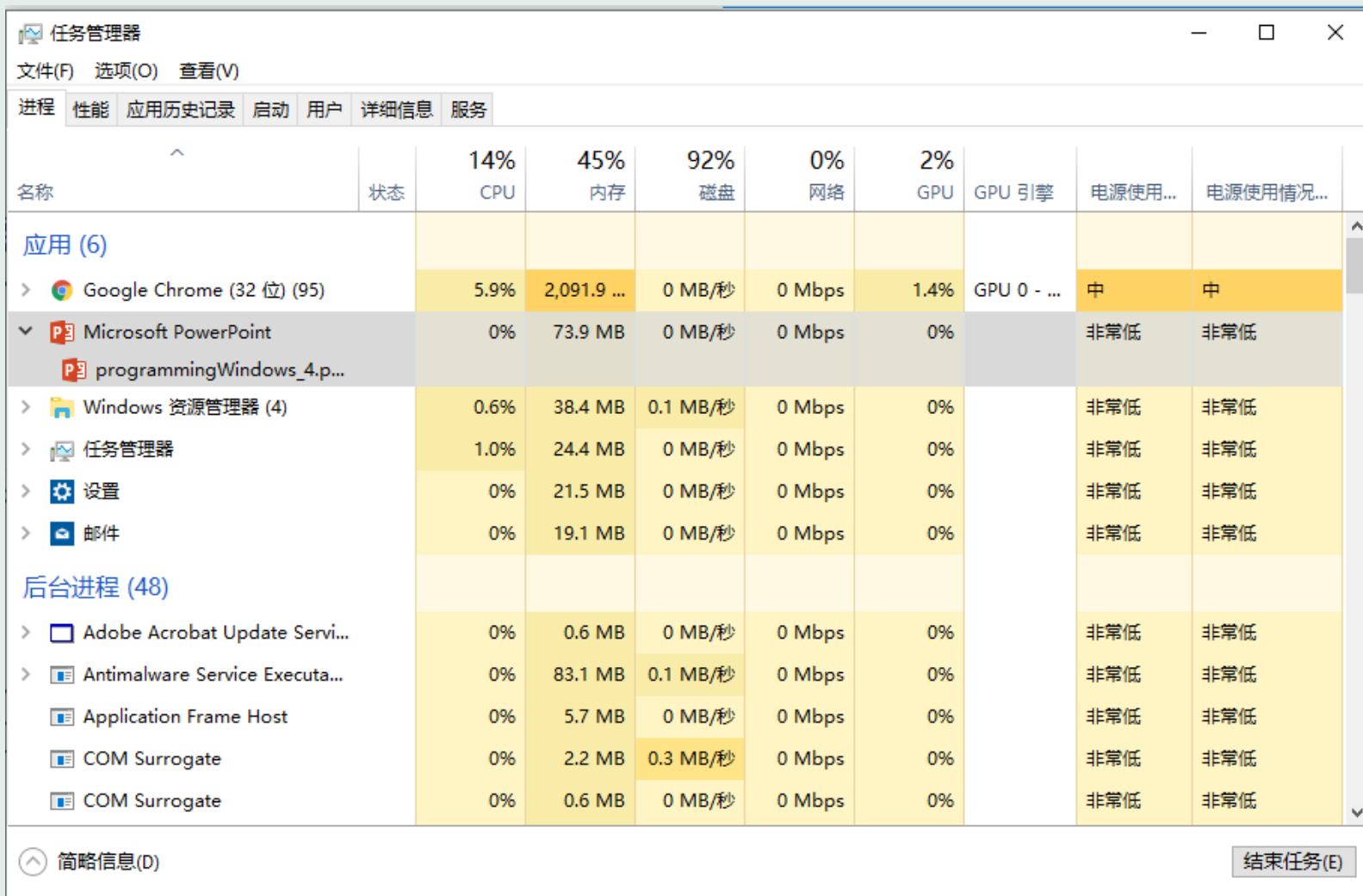
- 操作系统中的进程与用户进程**并发**运行，用户进程是由操作系统创建和调用的
- 用户进程亦可以创建和调用别的进程
- 被创建进程与创建者构成父子关系

➤ 1.进程

Windows是一个多任务的系统，它能够同时运行多个程序，其中的每一个正在运行的程序就称为一个“进程”

可以通过任务管理器查看Windows系统中当前运行的程序和进程

进程可以理解为一个程序的基本边界。是应用程序的一个运行例程，是应用程序的一次动态执行过程



任务管理器

文件(F) 选项(O) 查看(V)

| 名称 | 状态 | 14% CPU | 45% 内存 | 92% 磁盘 | 0% 网络 | 2% GPU | GPU 引擎 | 电源使用... | 电源使用情况... |
|----------------------------------|----|---------|-------------|----------|--------|--------|-------------|---------|-----------|
| 应用 (6) | | | | | | | | | |
| > Google Chrome (32 位) (95) | | 5.9% | 2,091.9 ... | 0 MB/秒 | 0 Mbps | 1.4% | GPU 0 - ... | 中 | 中 |
| Microsoft PowerPoint | | 0% | 73.9 MB | 0 MB/秒 | 0 Mbps | 0% | | 非常低 | 非常低 |
| programmingWindows_4.p... | | | | | | | | | |
| > Windows 资源管理器 (4) | | 0.6% | 38.4 MB | 0.1 MB/秒 | 0 Mbps | 0% | | 非常低 | 非常低 |
| > 任务管理器 | | 1.0% | 24.4 MB | 0 MB/秒 | 0 Mbps | 0% | | 非常低 | 非常低 |
| > 设置 | | 0% | 21.5 MB | 0 MB/秒 | 0 Mbps | 0% | | 非常低 | 非常低 |
| > 邮件 | | 0% | 19.1 MB | 0 MB/秒 | 0 Mbps | 0% | | 非常低 | 非常低 |
| 后台进程 (48) | | | | | | | | | |
| > Adobe Acrobat Update Servi... | | 0% | 0.6 MB | 0 MB/秒 | 0 Mbps | 0% | | 非常低 | 非常低 |
| > Antimalware Service Executa... | | 0% | 83.1 MB | 0.1 MB/秒 | 0 Mbps | 0% | | 非常低 | 非常低 |
| Application Frame Host | | 0% | 5.7 MB | 0 MB/秒 | 0 Mbps | 0% | | 非常低 | 非常低 |
| COM Surrogate | | 0% | 2.2 MB | 0.3 MB/秒 | 0 Mbps | 0% | | 非常低 | 非常低 |
| COM Surrogate | | 0% | 0.6 MB | 0 MB/秒 | 0 Mbps | 0% | | 非常低 | 非常低 |

简略信息(D) 结束任务(E)

➤ 2.线程

对于同一个进程，可以分成若干个独立的执行流，这样的流被称为“线程”

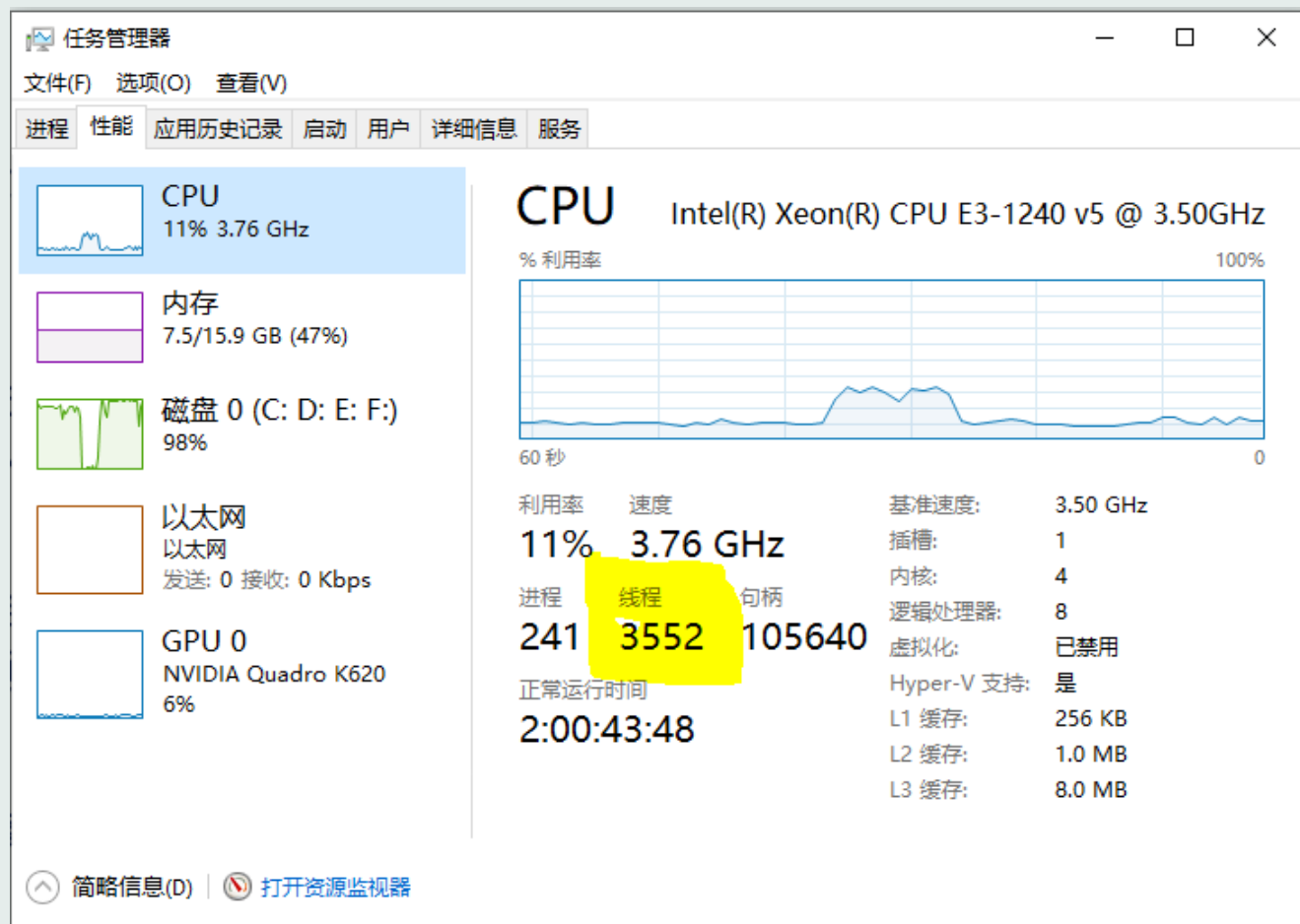
线程是操作系统分配处理器时间的基本单位，可以独立占用处理器的时间片

同一进程中的线程可以共享进程的资源 and 内存空间

每一个进程至少包含一个线程。

在.NET应用程序中，都是以Main()方法作为入口的，当调用此方法时系统就会自动创建一个主线程

process in memory while
threads in CPU kernels

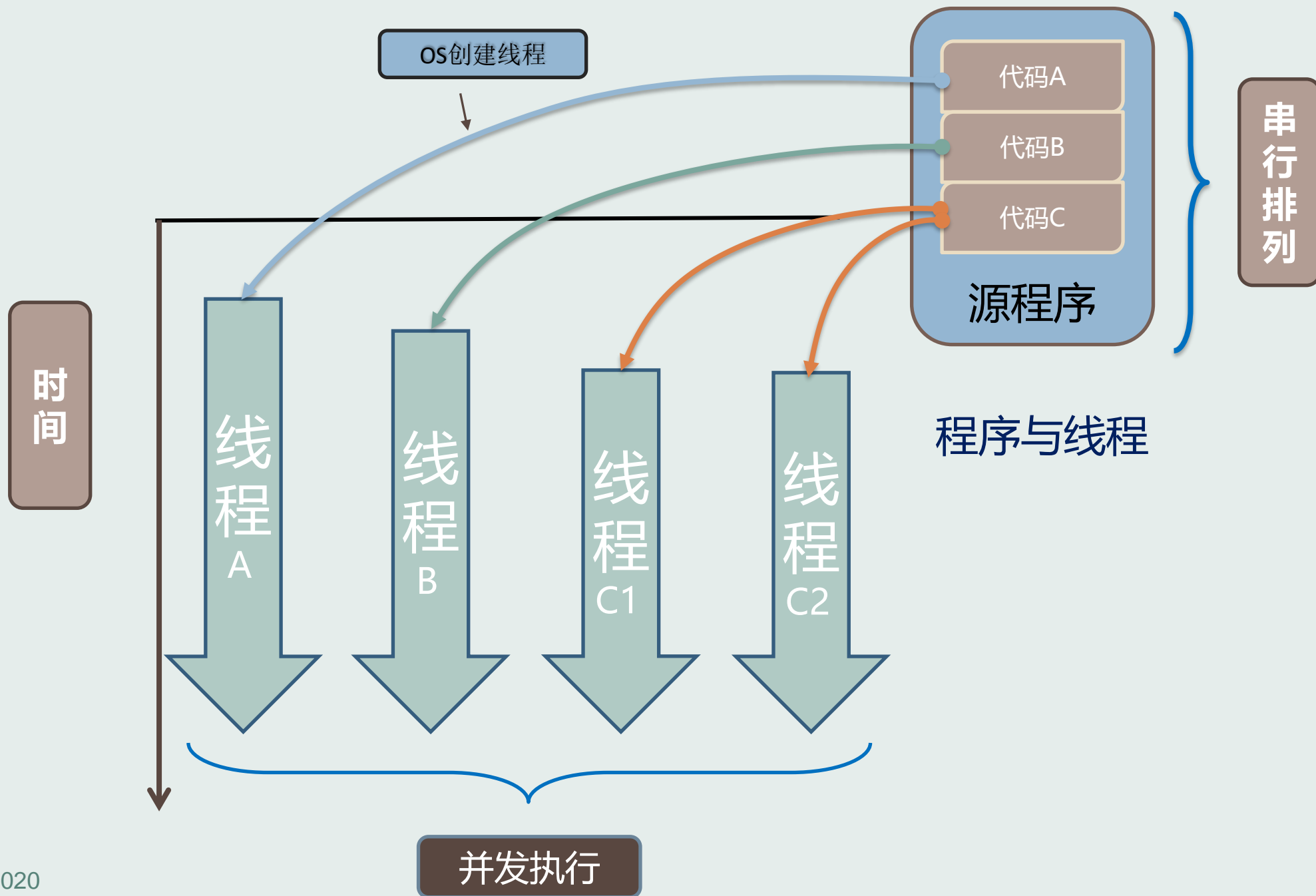


并发与并行 (concurrency & parallel)

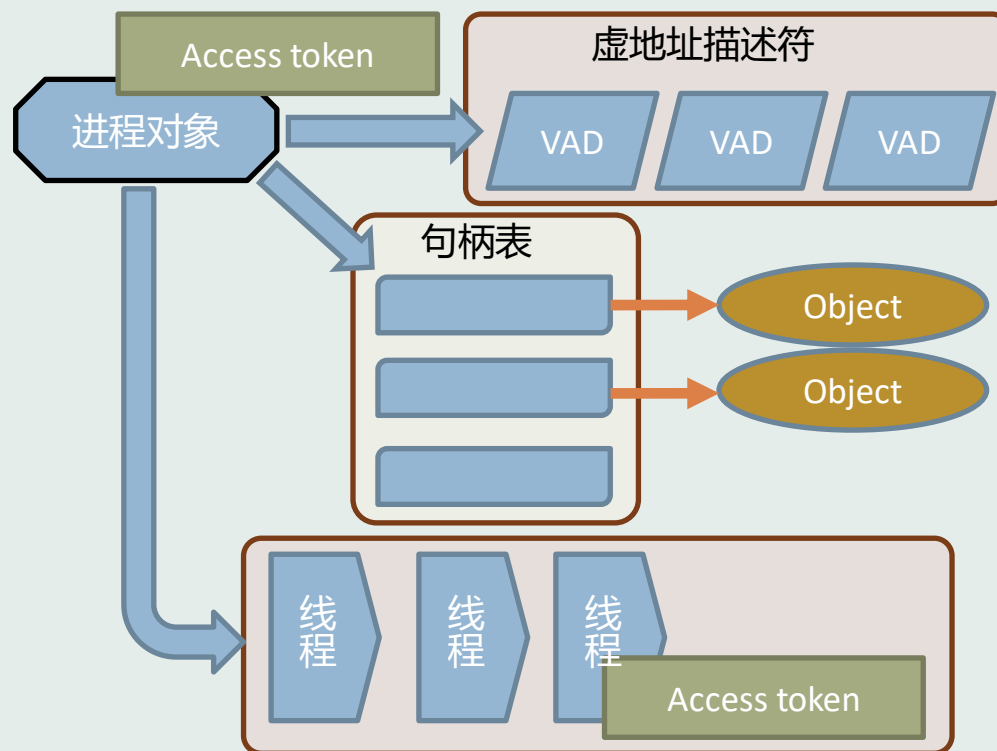
- 进程和线程技术是实现系统或应用程序并行性的重要基础
- “并发” 指系统或应用程序在某一时间段内同时处理多个事务的过程
 - 对于单核单处理器的计算机系统，由于单个CPU在任何时刻只能执行一个线程，所以这种计算机系统的并发，实际上是通过操作系统在各个正在执行的线程之间切换CPU，以分时处理的方式实现表面形式上的并发，只是因为其切换的速度快且处理能力强时，用户直观感觉不到而已
- 对于多处理器或多核的计算机系统，其多个CPU之间或多个核之间既有相互协作，又有独立分工，它们在各自执行一个相应线程时可以互不影响同时进行，实现并行处理
- 除了CPU之外，GPU也是多核系统，通常其并行计算能力非常强

并发：交替做多个

并行：同时做多/一个



进程对象结构



进程对象数据结构

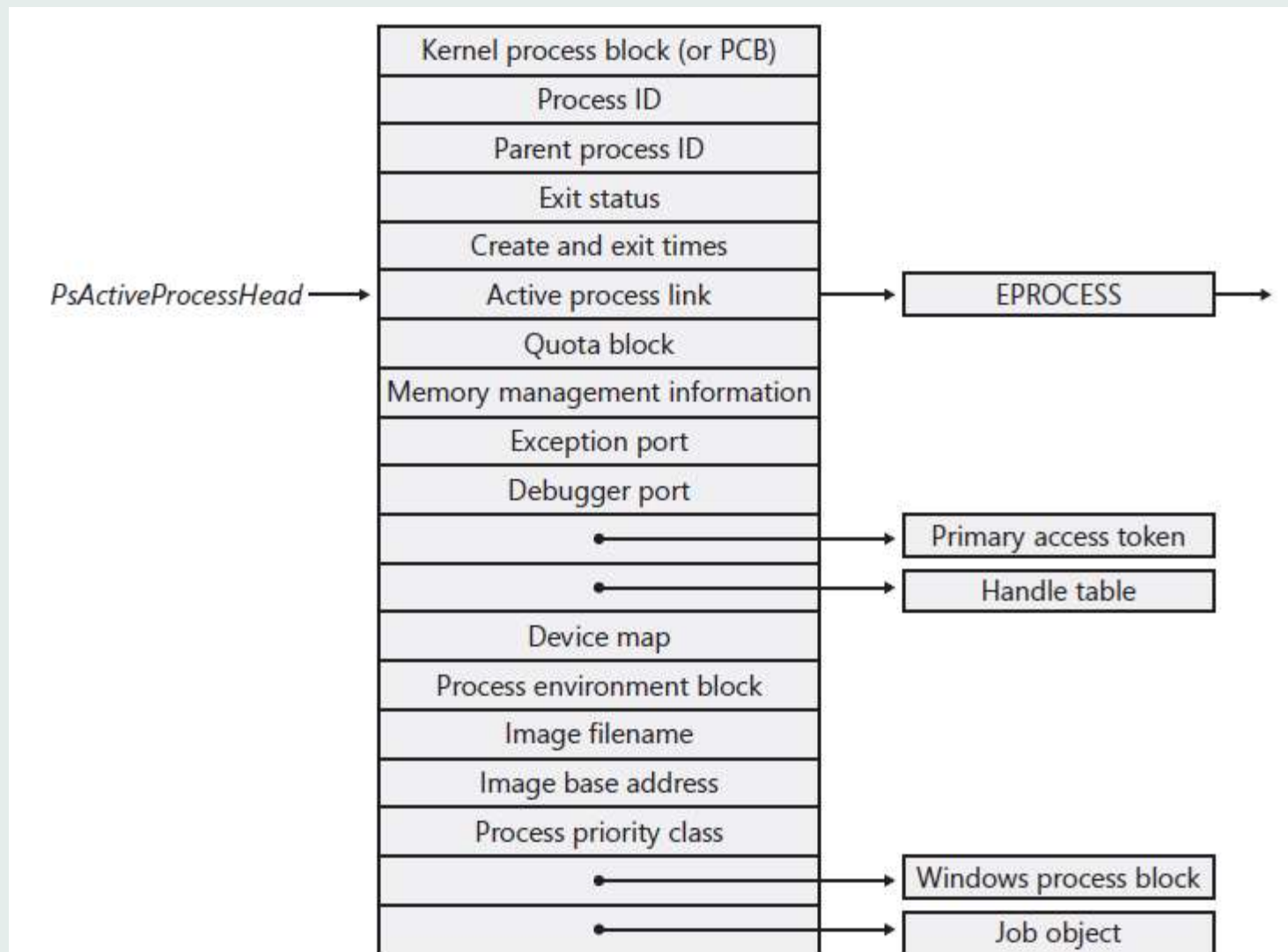


Figure 6-2 Structure of an executive process block

线程对象数据结构

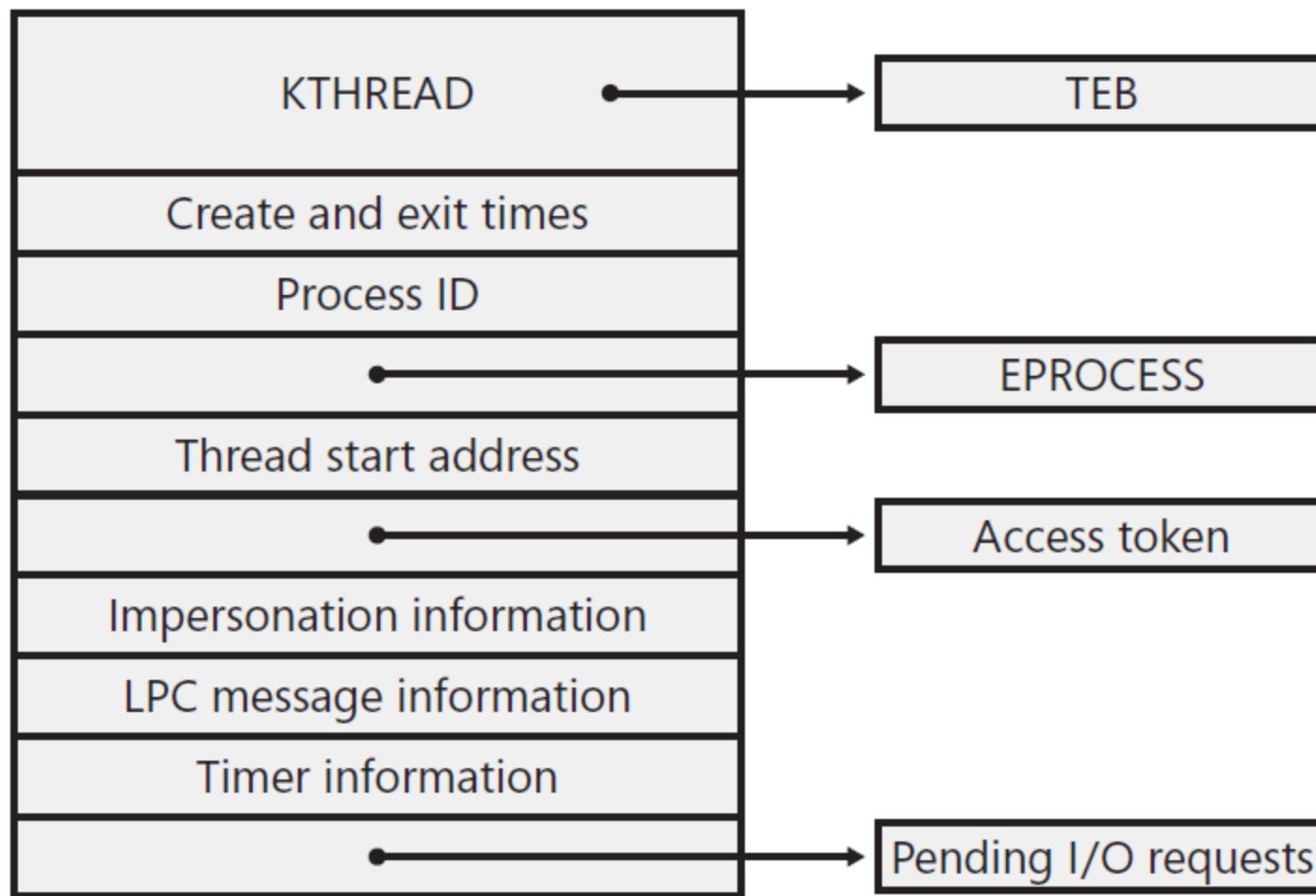
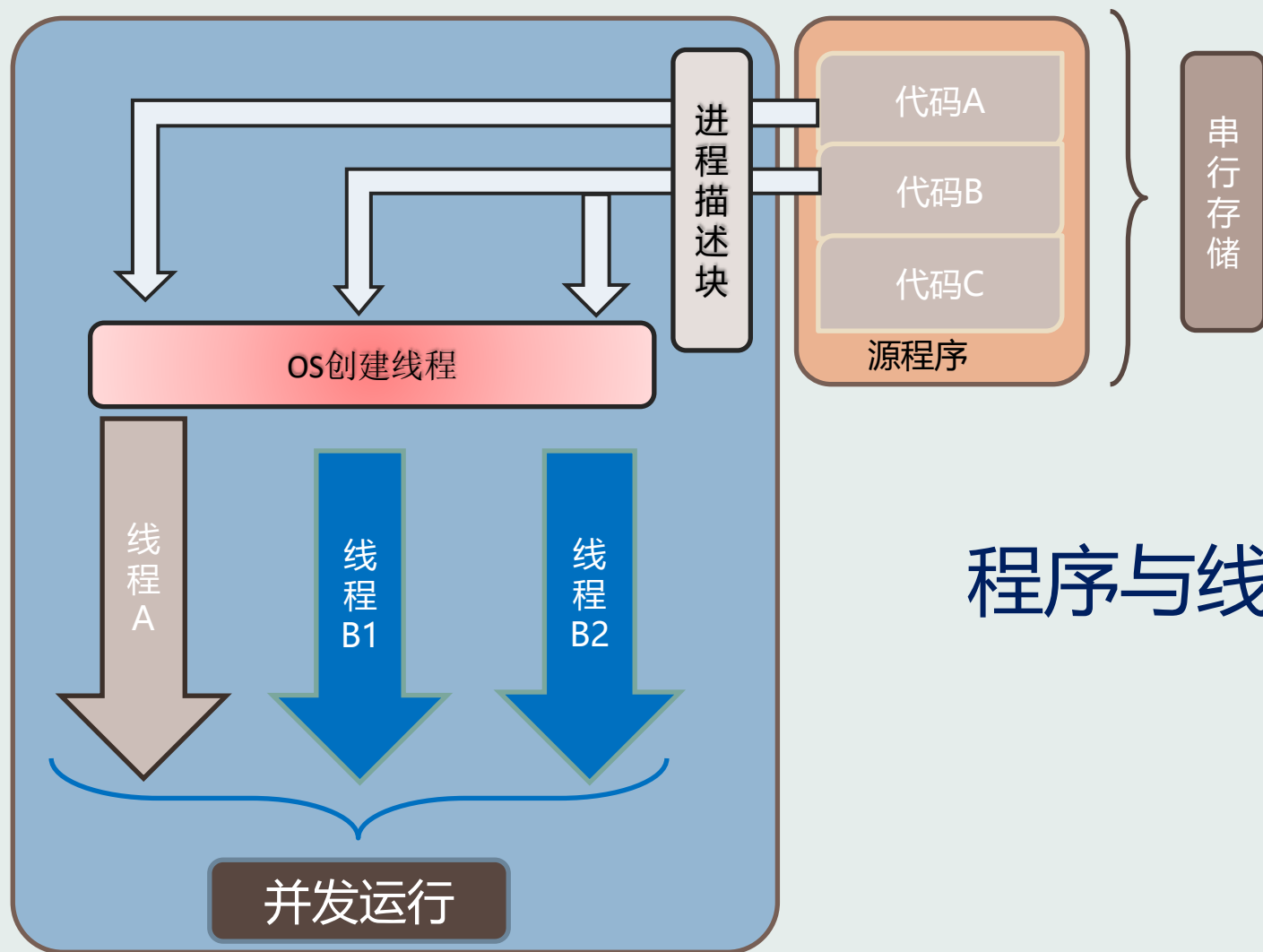


Figure 6-7 Structure of the executive thread block



程序与线程

创建进程过程

1. 打开文件映像 (.exe)
2. 创建windows进程对象
3. 创建初始线程对象，包括上下文，堆栈
4. 通知内核系统为进程运行作准备
5. 执行初始线程
6. 导入需要的DLL，初始化地址空间，由程序入口地址开始执行进程

进程的创建与启动代码-c#

- C#的System.Diagnostics命名空间下的Process类专门用于完成系统的进程管理任务，通过实例化一个Process类，就可以启动一个独立进程。

```
Process cmdP = new Process();  
cmdP.StartInfo.FileName = "cmd.exe";  
cmdP.StartInfo.CreateNoWindow = true;  
cmdP.StartInfo.UseShellExecute = false;  
cmdP.StartInfo.RedirectStandardOutput = true;  
cmdP.StartInfo.RedirectStandardInput = true;  
cmdP.Start();
```

- ▶ ProcessStartInfo类，则可以为Process定制启动参数
 - ▶ 比如RedirectStandardInput、RedirectStandardOutput、RedirectStandardError，分别重定向了进程的输入、输出、错误流

进程的其它操作 - C#

➤ 打开应用程序

```
Process.Start("calc");      // 计算器  
Process.Start("mspaint");   // 画图工具  
Process.Start("notepad");   // 记事本  
Process.Start("iexplore", "http://www.baidu.com");
```

➤ 关闭应用程序

```
// 得到程序中所有正在运行的进程  
Process[] preo = Process.GetProcesses();  
foreach (var item in preo)  
{  
    Console.WriteLine(item);  
    item.Kill(); // 杀死进程  
}
```

工具实验(包管理工具):

➤ winget

<https://docs.microsoft.com/en-us/windows/package-manager/winget/>

<https://github.com/microsoft/winget-cli/>

➤ Scoop & chocolatey

<https://github.com/principleWindows/scoop>

<https://www.oschina.net/p/chocolatey?hmsr=aladdin1e1>

工具实验：

➤ Pull Request

➤ DUMPBIN

<https://docs.microsoft.com/en-us/cpp/build/reference/dumpbin-reference?view=vs-2019>

<https://www.cnblogs.com/zhaotianff/p/10637397.html>

➤ readelf

<https://www.cnblogs.com/gatsby123/p/9750187.html>

objdump, nm

2.2 进程间通信机制简介

进程在运行时需要与其它进程通信

WINDOWS 进程间数据共享和通信的机制：

- IPC (Inter-Process Communications)
- IPC 经常使用C/S模式

通信目的及数据传输量考虑

- 高级通信 (IPC)
 - 传输的数据量大, 超过几十个字节
- 低级通信 (同步控制)
 - 传输的数据量小, 少于数个字节, 或仅是位单位

进程间通信方法分类

1. 共享内存（剪贴板、COM、DLL、DDE、文件映射）
2. 消息 WM_COPYDATA
3. 邮箱 
4. 管道，分有名管道与无名管道、进程重定向
5. Windows套接字 
6. NetBIOS特殊的网络应用

IPC需要考虑内容

1. 进程是否会通过网络与其它机器上的进程通信，仅使用本机通信机制是否满足应用需求；
2. 通信中的进程是否是处于不同的操作系统平台例如Windows与UNIX平台；
3. 有些进程通信机制是只用于图形化窗体界面的，而不适用于控制台程序；
4. 通信目的是用于同步控制还是数据的传送；
5. 数据传输量考虑；

IPC是否需要网络

仅适用本机内

文件

剪切板

无名管道

事件对象

消息队列

进程重定向

采用网络连接

邮槽

套接字

有名管道

NetBios

2.3 消息机制实现进程通讯

```
[DllImport("User32.dll", EntryPoint = "SendMessage")]
private static extern int SendMessage(IntPtr wnd,int msg,IntPtr wP,IntPtr lP);

// 或
[DllImport("User32.dll", EntryPoint = "SendMessage")]
private static extern int SendMessage(IntPtr wnd,int msg,IntPtr wP, ref
COPYDATASTRUCT lParam);

[DllImport("User32.dll", EntryPoint = "PostMessage")]
private static extern int PostMessage(IntPtr wnd,int msg,IntPtr wP, ref
COPYDATASTRUCT lParam);
```

```
#region 定义结构体
public struct COPYDATASTRUCT
{
    public IntPtr dwData;
    public int cbData;

    [MarshalAs(UnmanagedType.LPStr)]
    public string lpData;
}
#endregion
```

发送消息实现进程通讯：SendMessage ? PostMessage

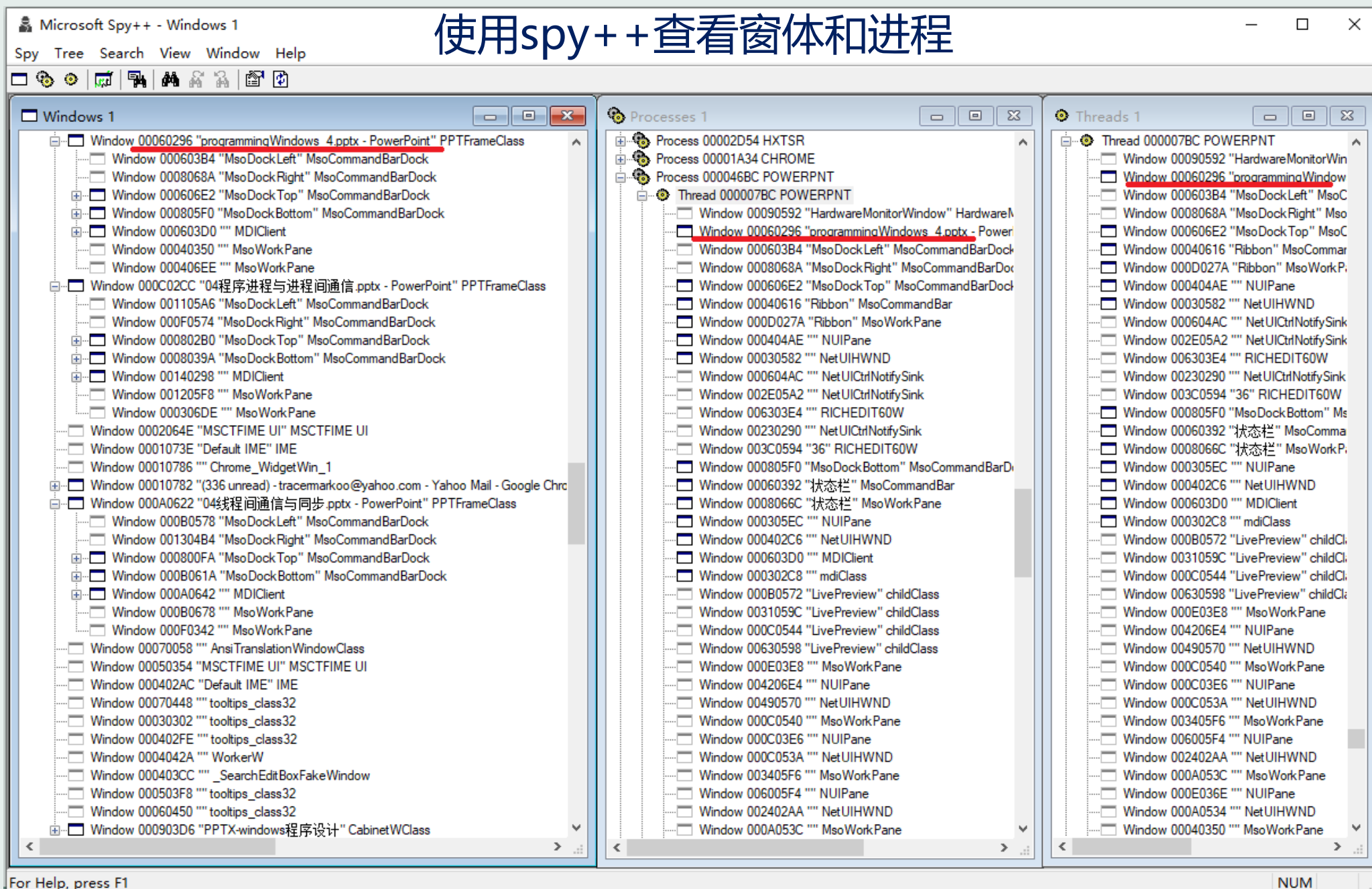
- SendMessage和PostMessage，这两个函数虽然功能非常相似，都是负责向指定的窗口发送消息，
- 但是SendMessage() 函数发出消息后一直等到接收方的消息响应函数处理完之后才能返回，并能够得到返回值，在此期间发送方程序将被阻塞，**SendMessage() 后面的语句不能被继续执行，即说此方法是同步的。**
- 而PostMessage() 函数在发出消息后马上返回，其后语句能够被立即执行，但是无法获取接收方的消息处理返回值，**即说此方法是异步的**

只能由SendMessage()来发送，而不能使用PostMessage()

因为系统必须管理用以传递数据的缓冲区生命期，如果使用了PostMessage()，数据缓冲区会在接收方（线程）有机会处理该数据之前，就被系统清除和回收。此外如果lpData指向一个带有指针或某一拥有虚函数的对象时，也要小心处理

如果传入的句柄不是一个有效的窗口或当接收方进程意外终止时，SendMessage() 会立即返回，因此发送方在这种情况下不会陷入一个无穷的等待状态中

使用spy++查看窗体和进程



消息机制实现进程通信实例-winform

➤在winform中

- 在发送进程application1,接收进程application2中定义相同的消息类型
WM_COPYDATA = 0x002;
- 在发送进程application1中, 通过窗体标题找到接收进程application2, 并调用windows api SendMessage, 向application2窗体句柄发送指定消息类型和内容
- 在接收进程application2中, 通过重载函数DefWndProc实现对消息的接收和处理
- 参考1 <https://blog.csdn.net/feiren127/article/details/5459827>
- 参考2 <https://www.cnblogs.com/MRRAOBX/articles/6626264.html>

```
protected override void DefWndProc(ref System.Windows.Forms.Message m)
{
    switch(m.Msg)
    {
        case WM_COPYDATA:
            COPYDATASTRUCT mystr = new COPYDATASTRUCT();
            Type mytype = mystr.GetType();
            mystr =(COPYDATASTRUCT)m.GetLParam(mytype);
            this.textBox1.Text =mystr.lpData;
            break;
        default:
            base.DefWndProc(ref m);
            break;
    }
}
```

消息机制实现进程通信实例-wpf

➤在wpf中

- 在发送进程application1,接收进程application2中定义相同的消息类型
WM_COPYDATA = 0x002;
- 在发送进程application1中, 通过窗体标题找到接收进程application2, 并调用windows api SendMessage, 向application窗体句柄发送指定消息类型和内容
- 在接收进程application2中, 通过自定义窗体钩子程序截获消息, 并进行处理

```
// 页面加载时, 添加消息处理钩子函数
private void ChildPage_Loaded(object sender, RoutedEventArgs e)
{
    HwndSource hwndSource;
    WindowInteropHelper wih = new WindowInteropHelper(this.parentWindow);
    hwndSource = HwndSource.FromHwnd(wih.Handle);
    // 添加处理程序
    hwndSource.AddHook(MainWindowProc);
}
// 钩子函数, 处理所收到的消息
private IntPtr MainWindowProc(IntPtr hwnd, int msg, IntPtr wParam, IntPtr lParam, ref bool handled)
{
    switch (msg)
    {
        case WM_COPYDATA:
            COPYDATASTRUCT mystr = new COPYDATASTRUCT();
            Type mytype = mystr.GetType();
            COPYDATASTRUCT MyKeyboardHookStruct = (COPYDATASTRUCT)Marshal.PtrToStructure (
                lParam, typeof(COPYDATASTRUCT));
            showComment(MyKeyboardHookStruct.lpData);
            break;
        default:
            break;
    }

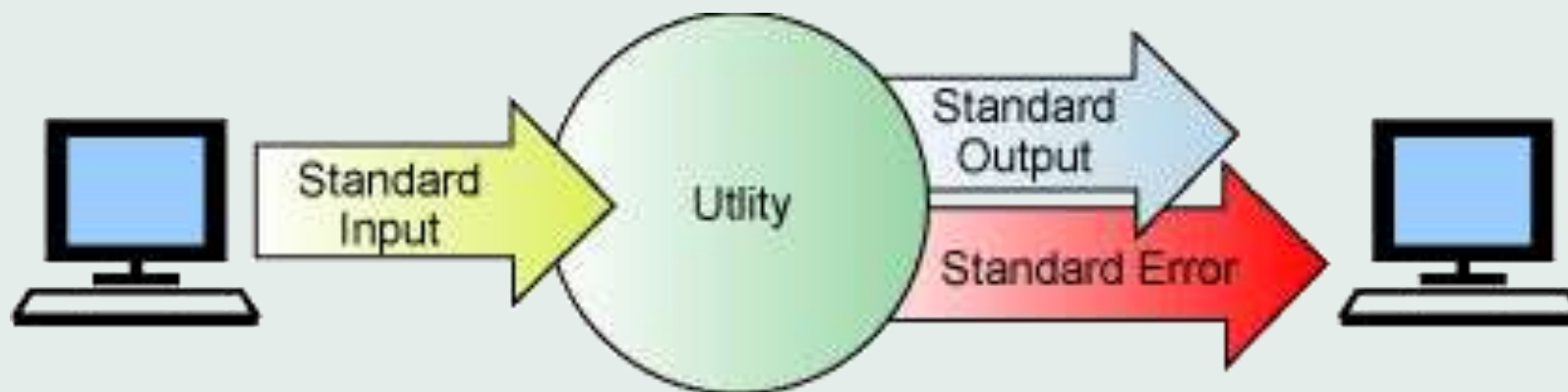
    return hwnd;
}
```

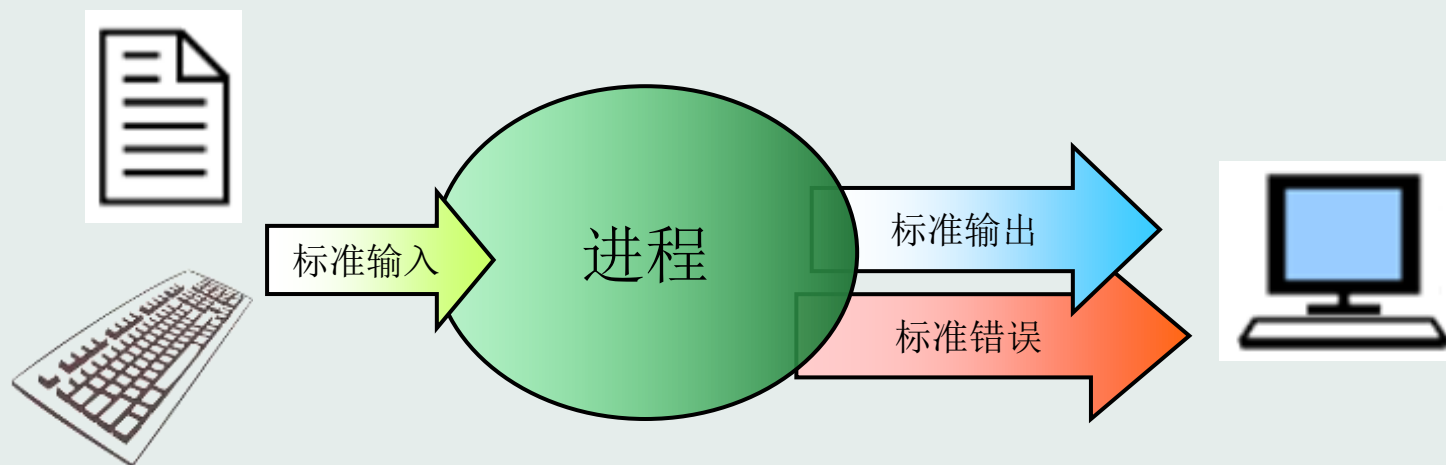
2.4 进程重定向实现进程通讯

- 概述
- 进程重定向意义
- 重定向回调函数
- 一个重定向例子

进程重定向概述

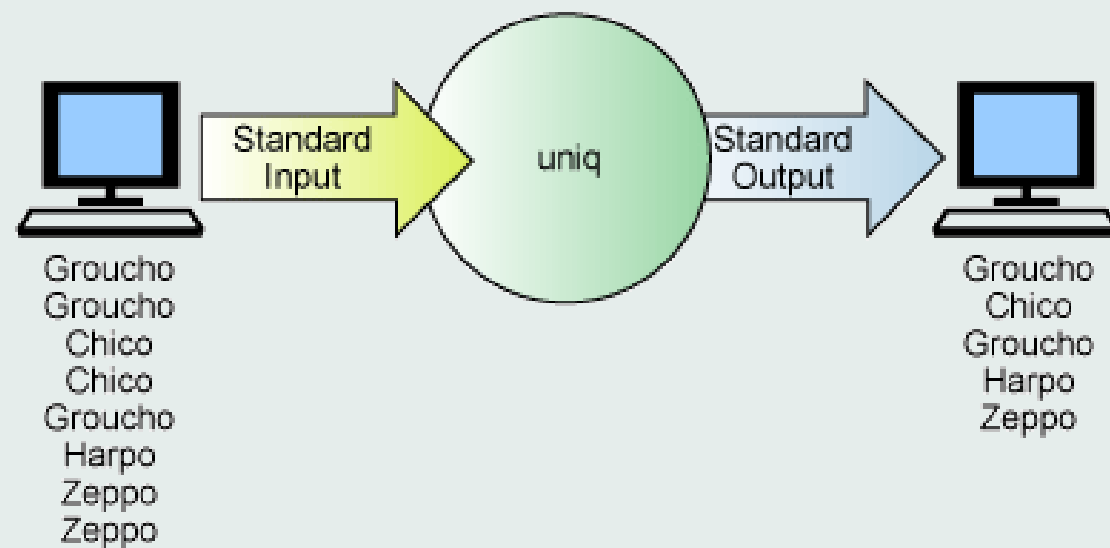
- 控制台进程文件描述符
 - 标准输入
 - 标准输出
 - 标准错误输出





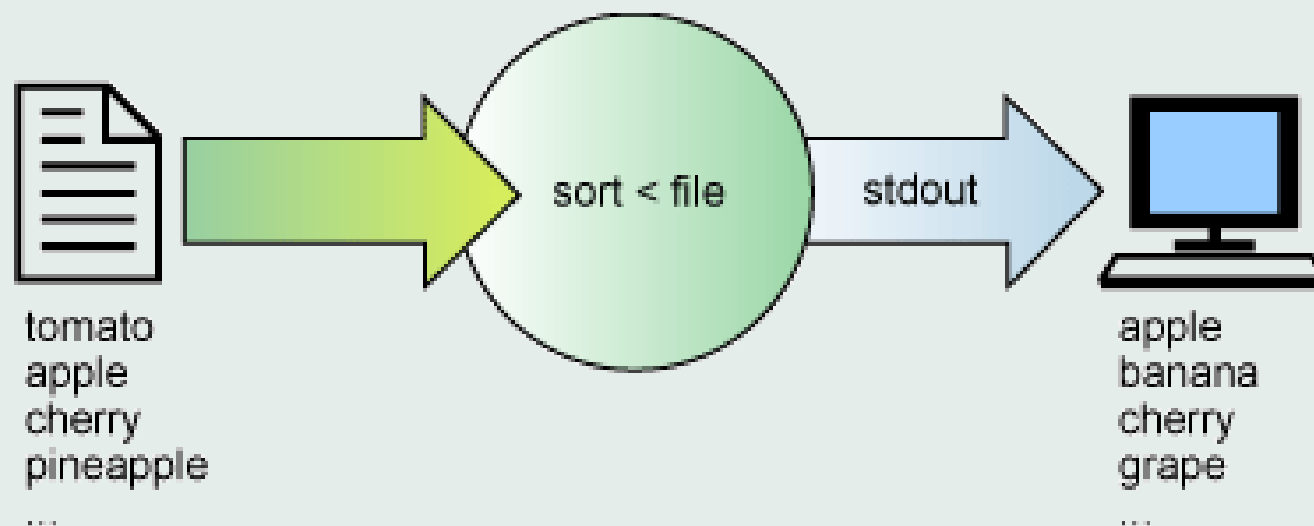
进程重定向概述

- 普通进程从键盘接收输入，输出到屏幕



进程重定向概述

- 使用文件作为进程的输入称为输入重定向



进程重定向概述

- 使用重定向符方法
- `dir > list.txt`
- `cmd >> file`
- `cmd < file`

进程重定向意义

- 调用控制台进程
 - Ping远程主机
 - 获取MAC地址getmac
 - 关机shutdown
 - 服务管理

重定向应用程序示例

- 界面设计
- 两种方式
- 内核函数使用

程序界面设计



重定向的两种方式

- 同步
- 异步方式

重定向同步读写方式

```
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命令执行九次
string strCmd = "ping www.whu.edu.cn -n 9";
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();
```



重定向同步读写方式

➤造成窗体没有响应的"问题"

- ▶ 不得在窗体线程中构造耗时的操作
- ▶ 窗体控件事件函数都属于窗体线程

特殊的BackGroundWorker控件

特殊的BackGroundWorker控件

- 造成窗体没有响应的"问题"
 - ▶ 不得在窗体线程中构造耗时的操作
 - ▶ 窗体控件事件函数都属于窗体线程

重定向异步读取方式

- 回调函数编写与设置
- 窗体消息处理函数重载



```
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命令执行九次
string strCmd = "ping www.whu.edu.cn -n 9";
process.Start();
process.StandardInput.WriteLine(strCmd);
process.StandardInput.WriteLine("exit");
```

```
process.OutputDataReceived += new  
DataReceivedEventHandler(strOutputHandler);  
process.Start();
```

```
private void strOutputHandler(object sendingProcess,
                             DataReceivedEventArgs outLine)
{
    cmdOutput.AppendLine(outLine.Data);

    // 通过FindWindow API的方式找到目标进程句柄，然后发送消息
    IntPtr WINDOW_HANDLER = FindWindow(null, "demo");
    if (WINDOW_HANDLER != IntPtr.Zero)
    {
        COPYDATASTRUCT mystr = new COPYDATASTRUCT();
        mystr.dwData = (IntPtr)0;
        if (MyStringUtil.IsEmpty(outLine.Data))
        {
            mystr.cbData = 0;
            mystr.lpData = "";
        }
        else
        {
            byte[] sarr = System.Text.Encoding.Unicode.GetBytes(outLine.Data);
            mystr.cbData = sarr.Length + 1;
            mystr.lpData = outLine.Data;
        }

        SendMessage(WINDOW_HANDLER, WM_COPYDATA, 0, ref mystr);
    }
}
```

2.5 管道机制实现进程通讯

管道机制是一种进程间通信 (IPC) 方式

- 操作系统创建管道对象
- 发送进程向管道写入数据
- 接收进程由管道中读出数据

管道可进行跨计算机的通信，可使用网络，也可使用文件等，它屏蔽低层实现机制提供给进程通信机制

有两种形式管道

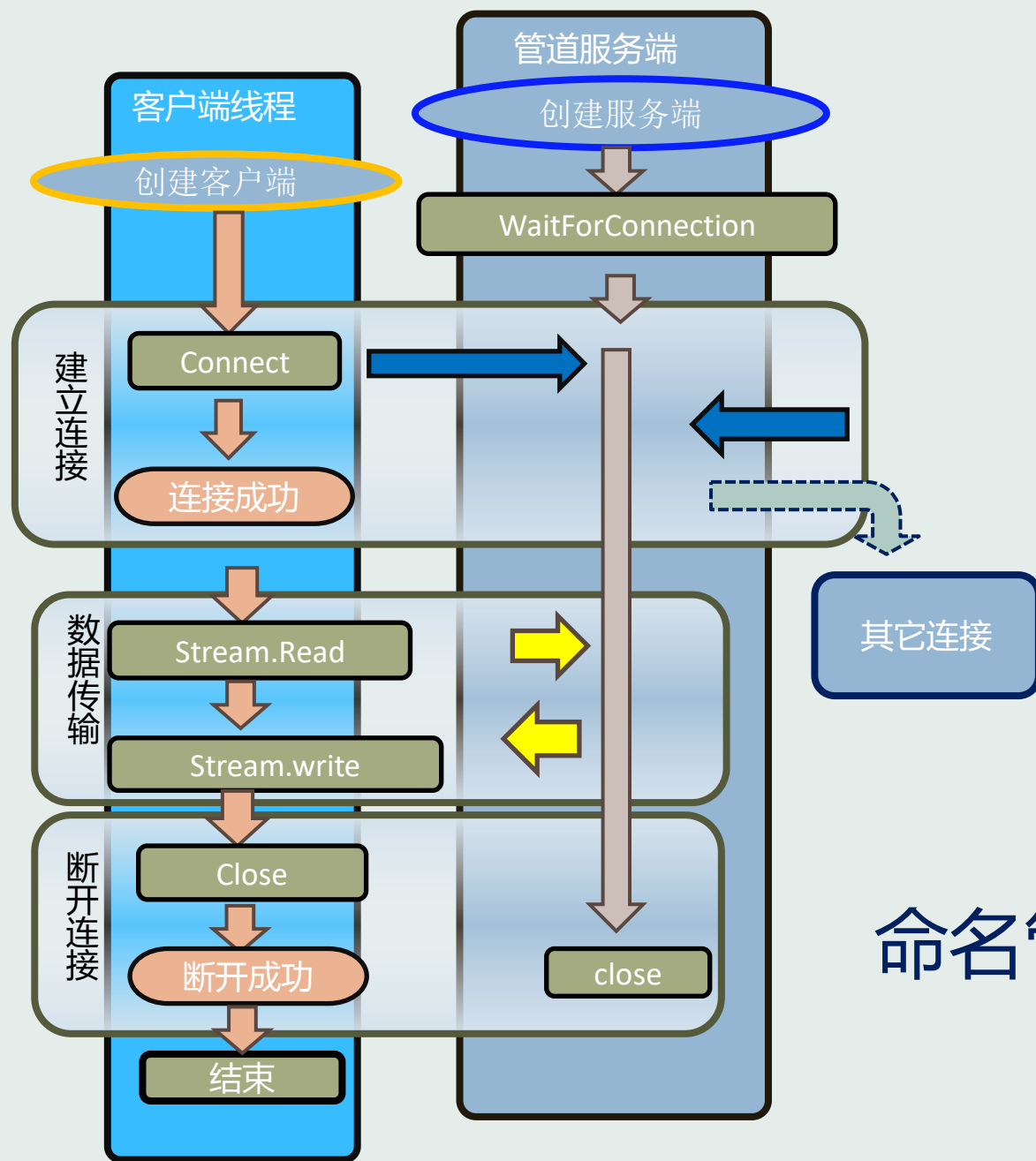
- 有名管道
- 无名管道

管道类

- AnonymousPipeClientStream
- AnonymousPipeServerStream
- NamedPipeClientStream
- NamedPipeServerStream

命名管道通信模式

- 字节模式
- 消息模式
- 管道通信程序示例



命名管道通信模式

上机练习作业

- 通过重定向机制实现进程间通信
 - 调用getmac获取网卡mac
 - 调用shutdown命令关闭或重启电脑
- 通过管道机制实现进程间通信
 - 客户端向服务器端发送数据
 - 服务器显示数据

THANK YOU !

