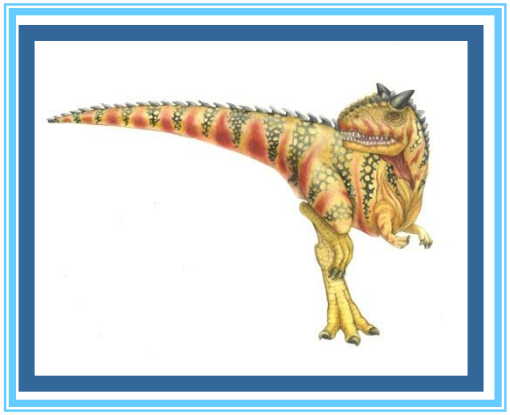
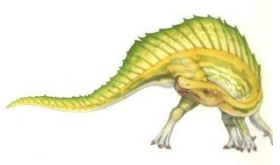
Windows进程通信与线程互斥和同步



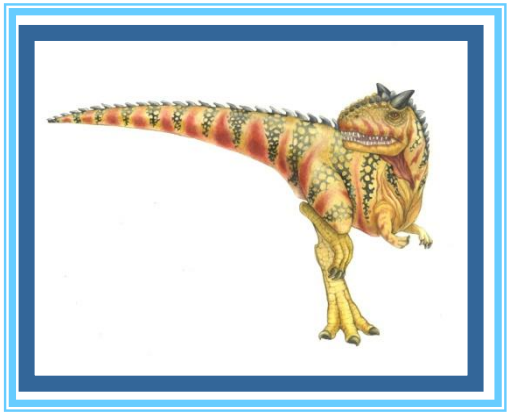
本章内容

- Windows进程通信
- Windows线程互斥和同步



进程间通信

Interprocess Communication (IPC)



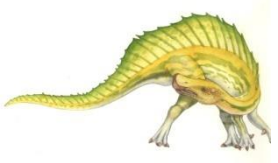
常用通信机制

- 信号(signal)
- 共享存储区(shared memory)
- 管道(pipe)
- 消息(message)
- 套接字(socket)

返回

Windows 2000/XP进程线程通信机制

- 信号
- 基于文件映射的共享存储区
- 无名管道和命名管道
- 邮件槽
- 套接字
- 剪帖板(Clipboard)
- 其他同步互斥机制



Windows 信号

- windows的信号处理有两组系统调用，分别用于不同的信号；

（1）SetConsoleCtrlHandler和GenerateConsoleCtrlEvent

- SetConsoleCtrlHandler在本进程的处理例程(HandlerRoutine)列表中定义或取销用户定义的处理例程；如：缺省时，它有一个CTRL+C输入的处理例程，我们可利用本调用来忽视或恢复CTRL+C输入的处理；
- GenerateConsoleCtrlEvent发送信号到与本进程共享同一控制台的控制台进程组；

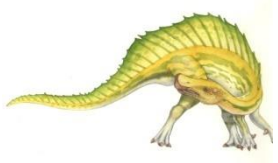
- 这一组系统调用处理的信号包括下表中的5种信号
  - CTRL_C_EVENT　收到CTRL+C信号
  - CTRL_BREAK_EVENT 收到CTRL+BREAK信号
  - CTRL_CLOSE_EVENT 当用户关闭控制台时系统向该控制台的所有进程发送的控制台关闭信号
  - CTRL_LOGOFF_EVENT 用户退出系统时系统向所有控制台进程发送的退出信号
  - CTRL_SHUTDOWN_EVENT 系统关闭时向所有控制台进程发送的关机信号

(2) signal和raise

- signal 设置中断信号处理例程；如：SIGINT（CTRL+C）、SIGABRT异常中止等信号的处理；
- raise给本进程发送一个信号；

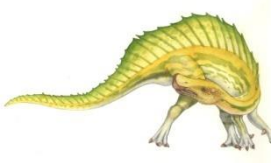
- 6种信号是与传统的UNIX系统相同的，而前面一组系统调用处理的5种信号是Windows 中特有的。处理信号列表（6种）
  - SIGABRT 非正常终止
  - SIGFPE 浮点计算错误
  - SIGILL 非法指令
  - SIGINTCTRL+C 信号(对Win32无效)
  - SIGSEGV 非法存储访问
  - SIGTERM 终止请求

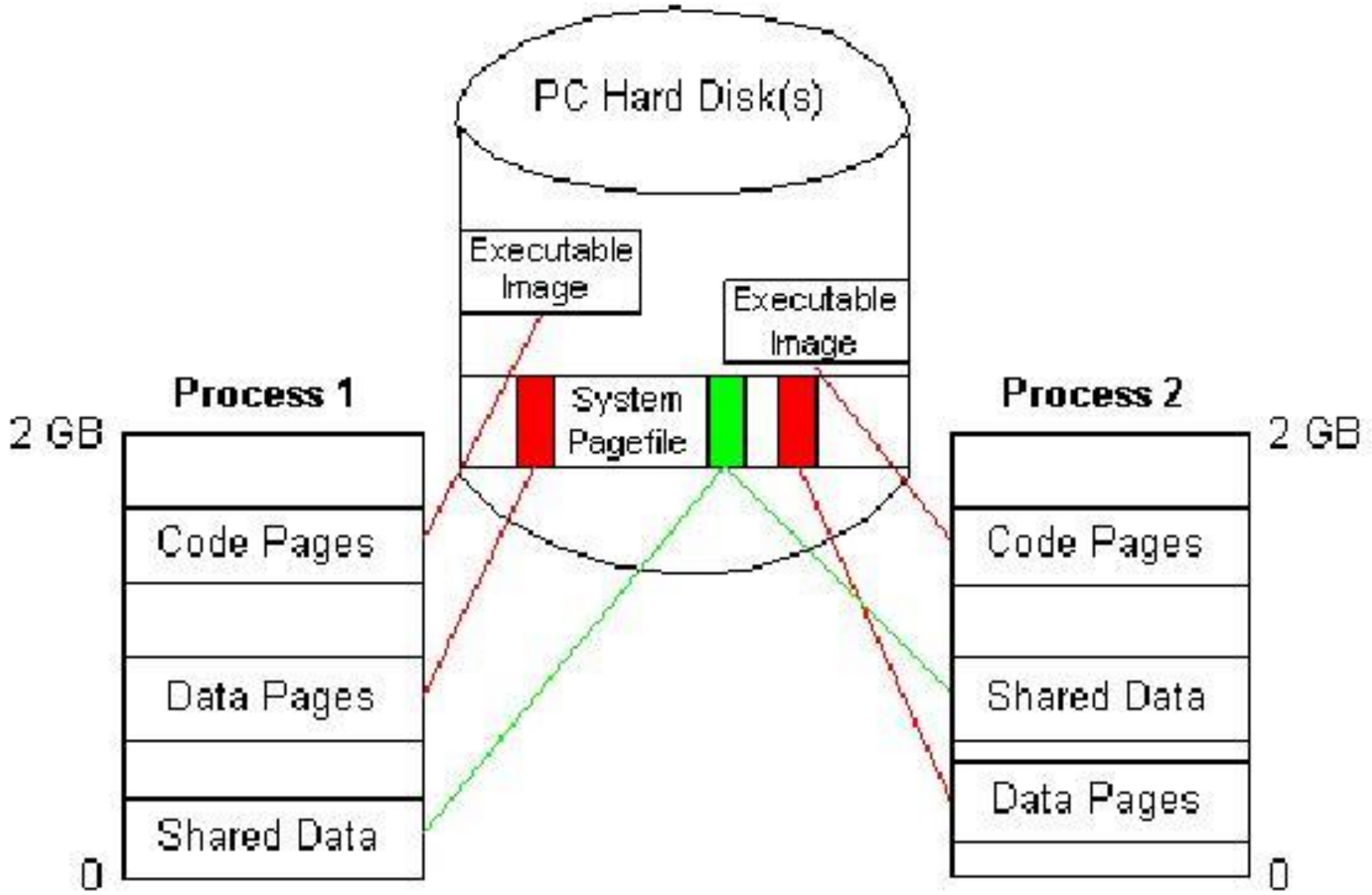
实例：raise.cpp

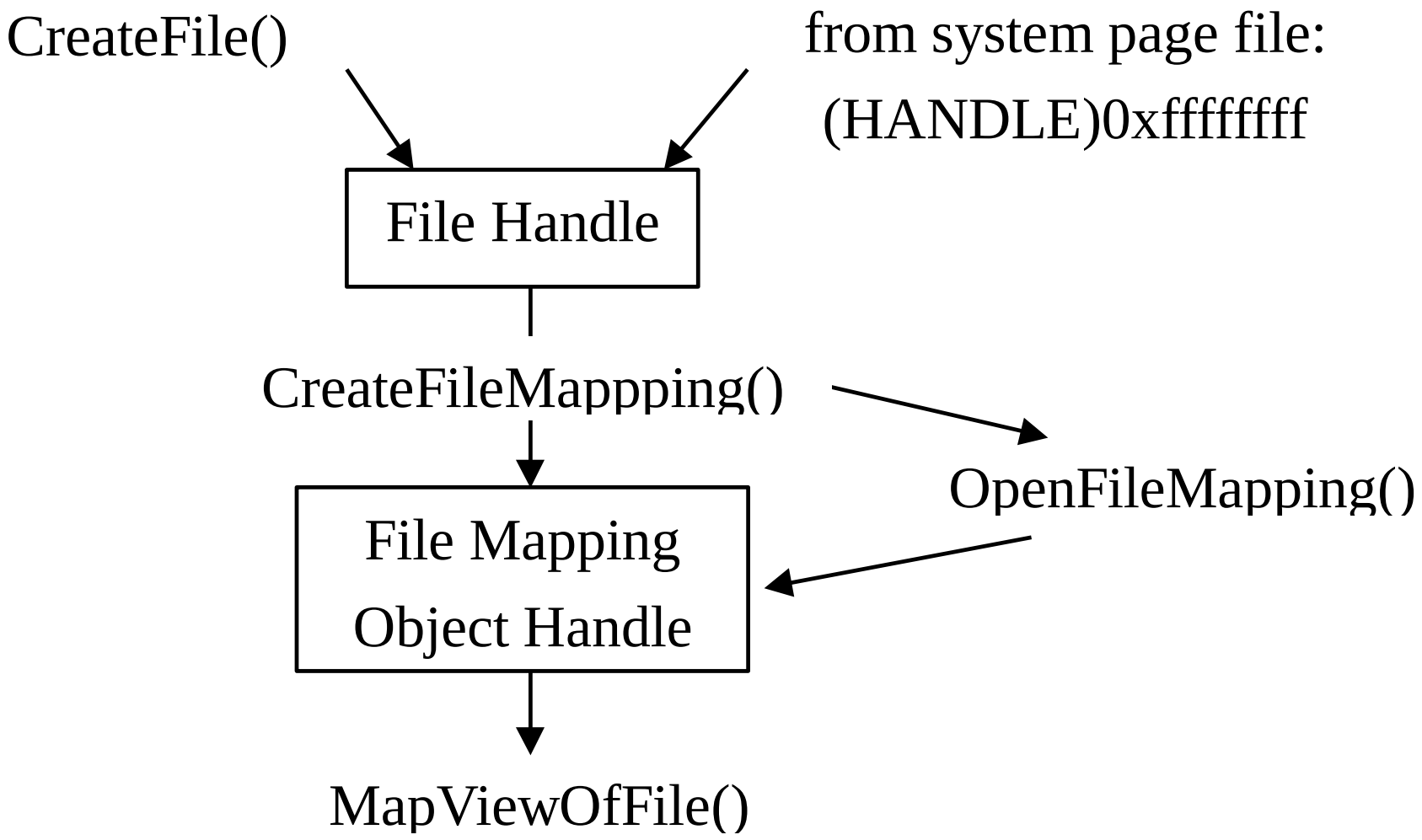


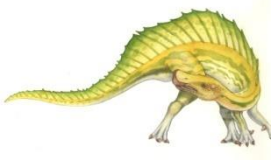
Windows 的文件映射

- 采用文件映射(file mapping)机制：可以将整个文件映射为进程虚拟地址空间的一部分来加以访问。
  - CreateFileMapping为指定文件创建一个文件映射对象，返回对象指针；
  - OpenFileMapping打开一个命名的文件映射对象，返回对象指针；
  - MapViewOfFile把文件映射到本进程的地址空间，返回映射地址空间的首地址；
- 利用首地址进行读写；
  - FlushViewOfFile可把映射地址空间的内容写到物理文件中；
  - UnmapViewOfFile拆除文件映射与本进程地址空间间映射关系；
- 利用CloseHandle关闭文件映射对象




PC Hard Disk(s)
Executable Image
Executable Image
System Pagefile
Process 1
2 GB
Code Pages
Data Pages
Shared Data
0
Process 2
2 GB
Code Pages
Shared Data
Data Pages
0

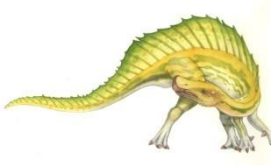

CreateFile()
from system page file:
(HANDLE)0xffffffff
File Handle
CreateFileMappping()
OpenFileMapping()
File Mapping
Object Handle
MapViewOfFile()



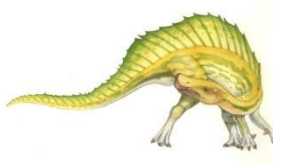
Windows 管道

- 无名管道：类似于UNIX管道，CreatePipe可创建无名管道，得到两个读写句柄；利用ReadFile和WriteFile可进行无名管道的读写：

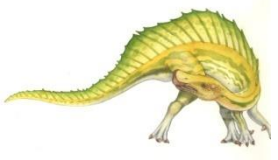
```
BOOL CreatePipe( PHANDLE hReadPipe,
      // address of variable for read handle
    PHANDLE hWritePipe,
      // address of variable for write handle
     LPSECURITY_ATTRIBUTES lpPipeAttributes,
      // pointer to security attributes
     DWORD nSize            // number of bytes reserved for pipe
);
```



- 命名管道：一个服务器端与一个客户进程间的通信通道；可用于不同机器上进程通信；
  - 类型分为：字节流，消息流（报文）；
  - 访问分为：单向读，单向写，双向；
  - 还有关于操作阻塞(wait/nowait)的设置。相应有一个文件句柄
  - 通常采用client-server模式，连接本机或网络中的两个进程
  - 管道名字：作为客户方（连接到一个命名管道实例的一方）时，可以是"\\serverName\pipe\pipename"；作为服务器方(创建命名管道的一方)时，只能取serverName为\\.\pipe\PipeName，不能在其它机器上创建管道；



- 命名管道服务器支持多客户：为每个管道实例建立单独线程或进程。
  - CreateNamedPipe在服务器端创建并返回一个命名管道句柄；
  - ConnectNamedPipe在服务器端等待客户进程的请求；
  - CallNamedPipe从管道客户进程建立与服务器的管道连接；
  - ReadFile、WriteFile（用于阻塞方式）、ReadFileEx、WriteFileEx（用于非阻塞方式）用于命名管道的读写；



命名管道实例

- 通过命名管道进行多个进程间的通信。其中的主要内容有：
  - Server32.c中ServerProc函数是服务器进程中的主要过程。
    - 它创建命名管道（CreateNamedPipe），并等待客户连接（ConnectNamedPipe）。
    - 它为每一个用户连接建立一个线程（CreateThread）。
    - 服务器进程会每个客户送来的数据（ReadFile），并发给每个客户（WriteFile）。

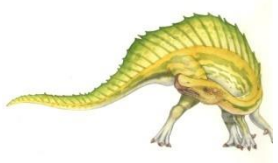
例：server32.c、client32.c

- 启动多个client进程进行通信

Windows 邮件槽

- Windows 邮件槽是一种消息通信机制
- 邮件槽(mailslot)：驻留在OS核心，不定长数据块（报文），不可靠传递
- 通常采用client-server模式：单向的，从client发往server；server负责创建邮件槽，它可从邮件槽中读消息；client可利用邮件槽的名字向它发送消息；

- 有关的API如：
  - CreateMailslot服务器方创建邮件槽，返回其句柄；
  - GetMailslotInfo服务器查询邮件槽的信息，如：消息长度、消息数目、读操作等待时限等；
  - SetMailslotInfo服务器设置读操作等待时限；
  - ReadFile服务器读邮件槽；
  - CreateFile客户方打开邮件槽；
  - WriteFile客户方发送消息；
- 在邮件槽的所有服务器句柄关闭后，邮件槽被关闭。这时，未读出的报文被丢弃，所有客户句柄都被关闭。
- 例：mailslotCLIENT.Cpp、mailslotSERVER.Cpp



套接字(socket)

- 双向的，数据格式为字节流（一对一）或报文（多对一，一对多）；主要用于网络通信；
- 支持client-server模式和peer-to-peer模式，本机或网络中的两个或多个进程进行交互。提供TCP/IP协议支持
- 在Windows 中的规范称为"Winsock"(与协议独立，或支持多种协议)：WSASend, WSASendto, WSARecv, WSARecvfrom;

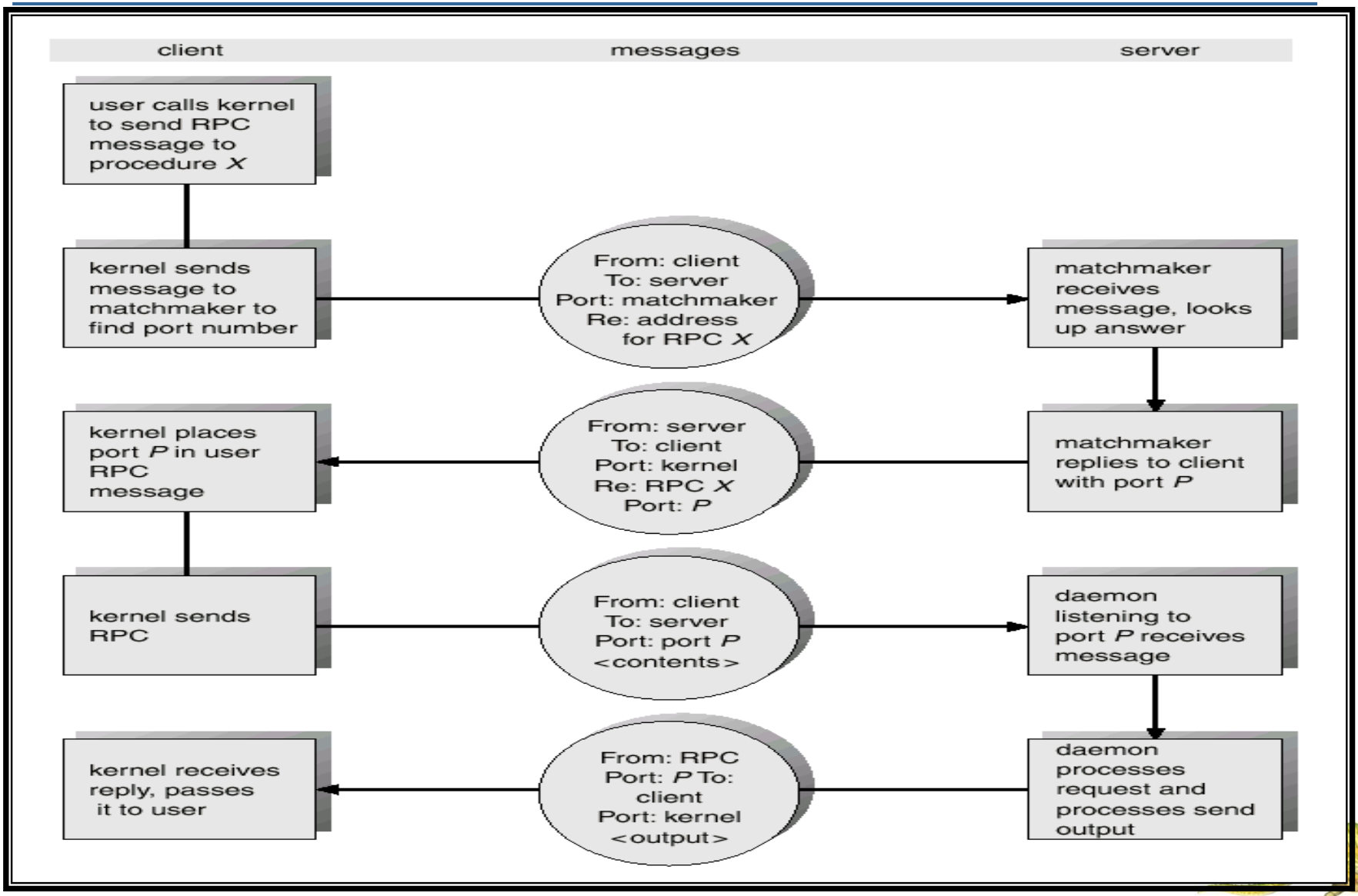
- 例：ServerSocket.cpp、 ClientSocket.cpp

返回

Client-Server Communication

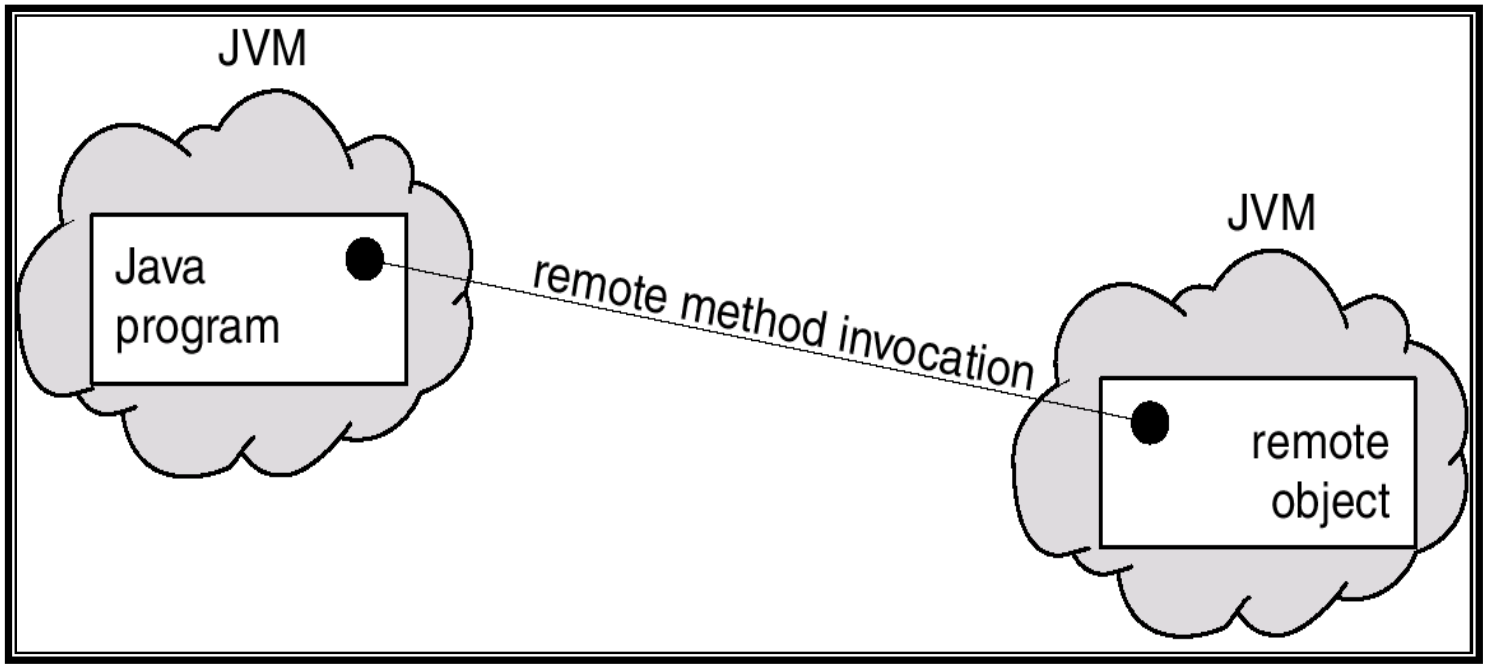
- 远程过程调用Remote Procedure Calls （RPC）
- 远程方法启用Remote Method Invocation (Java)

Execution of RPC (RPC执行)



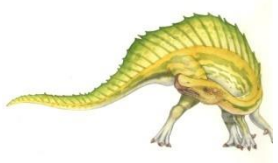
远程方法调用

- 远程方法调用(RMI)是类似与RPCs的一种Java 机制。
- RMI允许在一台机器上一个 Java 程序调用在远程对象上的一个方法。



剪帖板(Clipboard)

- 当进程间的复杂信息交流需要约定交流信息的格式。剪贴板就是Windows 2000/XP提供的一种信息交流方式，可增强进程的信息交流能力。
- Windows 2000/XP提供了一组相关的API来完成应用进程与剪贴板间的格式化信息交流。
- 当执行复制操作时，应用程序将选中的数据以标准的格式或者应用程序定义的格式放到剪贴板中，然后其他的应用程序可以从剪贴板中以其可以支持的格式获取所需要的数据。



剪贴板相关的API

- OpenClipboard: 打开剪贴板；
- CloseClipboard：关闭剪贴板；
- EmptyClipboard：清空剪贴板；
- SetClipboardData：把数据及其格式加入剪贴板；
- GetClipboardData：从剪贴板读取数据；
- RegisterClipboardFormat：注册剪贴板格式；

应用程序向剪贴板发送文本

- 应用程序向剪贴板发送文本的操作过程可分为如下5个步骤：

1. 拷贝文本到全局内存

   - 调用GlobalAlloc函数为文本分配全局存储空间

     HGLOBAL GlobalAlloc

     （UINT uFlags, //函数分配内存形式标识。

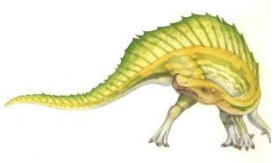
     DWORD dwBytes) //分配字节数

   - 然后调用 GlobalLock 函数锁定分配的内存块

     LPVOID GlobalLock（HGLOBAL hMem） //hMem为内存块地址

   - 把数据拷贝到内存以后，应及时调用GlobalUnLock 解锁内存句柄

     BOOL GlobalUnLock （HGLOBAL hMem） //hMem为内存句柄



应用程序向剪贴板发送文本

2. 打开剪贴板

- 应用程序调用OpenClipboard 函数

BOOL OpenClipboard （HWND hwnd）

- //hwnd为打开剪贴板的窗口句柄。

3. 清除剪贴板中的所有句柄

- 应用程序调用函数EmptyClipboard

BOOL EmptyClipboard （VOID）

4. 向剪贴板传送文本全局内存句柄

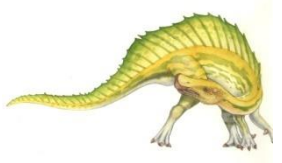
- 应用程序调用函数 SetClipboardData

HANDLE SetClipboardData

（UINT uFormat, //数据格式标识

HANDLE hMem) //数据句柄

- 应用程序一旦将文本内存句柄传递给剪贴板，则该句柄属于剪贴板，应用程序不能再对其进行操作。

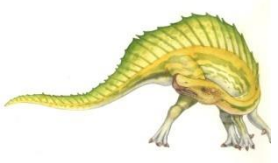


应用程序向剪贴板发送文本

5. 关闭剪贴板

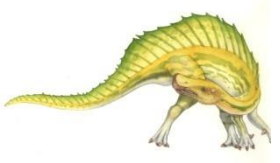
- 应用程序调用函数CloseClipboard

  BOOL CloseClipboard（VOID）



- 下面的程序段是应用程序向剪贴板发送文本的一般过程

```
HANDLE hText;  LPTSTR lpString，lpText;
…
case  IDM_COPY;  //分配全局内存
if（！（hText=GloalbAlloc（GWND，Sizeof（lpSring））））
{ MessageBox（hwnd,`“全局内存分配失败！”，“提示”，MB_OK）;break;}
lpText=GlobalLock(hText);   //锁定文本内存句柄并返回文本指针
lstrcpy（lpText，lpString）; //拷贝文本
GlobalUnlock（hText）;              //解锁文本内存句柄
If（！OpenClipboard（hwnd））
 { MessageBox(hwnd,"剪贴板打开失败！","提示",MB_OK);break；}
EmptyClipboard( );                  //清除剪贴板
SetCilpboardData（CF_TEXT,hText）;  //设置剪贴板文本
CloseClipboard（）;                 //关闭剪贴板
hText=NULL  //以避免应用程序再通过该句柄执行其他操作
break;
```



获取剪贴板文本

- 应用程序从剪贴板上获取文本的操作过程可分为如下4个步骤：

1. 打开剪贴板：调用函数OpenClipboard打开剪贴板
2. 检查剪贴板数据格式：

   BOOL IsClipboardFormatAvailable（UINT uformat）

3. 获取剪贴板文本：由函数GetClipboardData（）

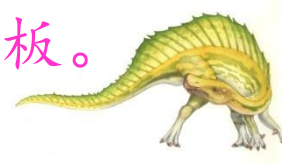
   HANDLE GetClipboardData（UINT uformat)

uformat为数据格式标识

获取剪贴板文本内存句柄后即可调用GlobalLock返回指向文本的指针

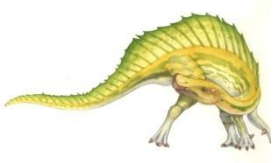
获取的内存句柄属于剪贴板，不属于应用程序，因此应用程序读取剪贴板中的数据后，只能调用函数GlobalUnlock解锁该内存句柄而无法将其释放。

4. 关闭剪贴板：调用函数CloseClipboard关闭剪贴板。



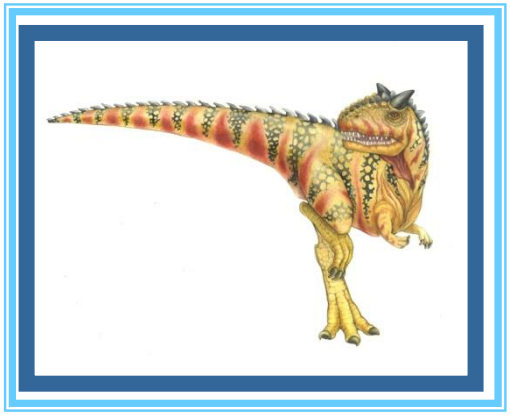
- 下面的程序段是应用程序获取剪贴板文本的一般形式：

```
HANDLE  hText;
LPTSTR   lpString，lpText;
. …
 case IDM_PASTE;
 if(!IsClipboardFormstAvailable(CF_TEXT)) //检查剪贴板的数据格式
 {MessageBox(hwnd,“剪贴板上无文本数据”， “提示”， MB_OK); Break；}
    if（！ OpenClipboard（hwnd）） //打开剪贴板
   {MessageBox（hwnd“剪贴板板开失败”， “提示”， MB_OK);Brdak;}
  //获取剪贴板文本内存句柄
if(!(hText=GetClipboardData(CF_TEXT)))
 {MessageBox(hwnd,“无法读取剪贴板数据”,“提示”,MB_OK);
 CloseClipboard(); Break; }
 lpTexe=GlobalLock(hText);  //锁定文本内存名柄并返回文本指针
 //复制剪贴板文本内容
 lpString=（LPTSTR）malloc（GlobalSize（hText））；
 lpstrcpy（lpString，lpText）；
 GlobalUnlock（hText）；     //解锁文本内存句柄
 CloseClipboard（）；        //关闭剪贴板
 …
 break;
```



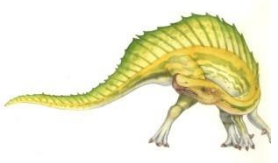
- 例：clipborad.cpp
  - 这个例子可把命令行参数复制到剪帖板中

Windows线程互斥和同步



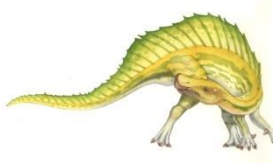
Windows 的进程、线程互斥和同步机制

- 互锁变量访问
- 临界区对象(Critical Section)
- 互斥对象(Mutex)
- 信号量对象(Semaphore)
- 事件对象(Event)



同步对象等待

- Win32 API提供了一组能使线程阻塞其自身执行的等待函数。这些函数只有在作为其参数的一个或多个同步对象产生信号时才会返回。在超过规定的等待时间后，不管有无信号，函数也都会返回。在等待函数未返回时，线程处于等待状态，此时线程只消耗很少的CPU时间。
- 对于上面同步对象，Windows 2000/XP提供了两个统一的等待操作WaitForSingleObject和WaitForMultipleObjects。
  - WaitForSingleObject在指定的时间内等待指定对象为可用状态(signaled state)；
  - WaitForMultipleObjects在指定的时间内等待多个对象为可用状态；



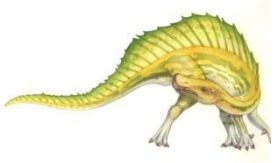
同步对象等待

```
DWORD WaitForSingleObject (
    HANDLE hHandle, // 等待对象句柄
    DWORD dwMilliseconds// 以毫秒为单位的最长等待时间
    );

DWORD WaitForMultipleObjects(
    DWORD nCount,
    //对象句柄数组中的句柄数;
    CONST HANDLE *lpHandles,
    // 指向对象句柄数组的指针，数组中可包括多种对象句柄;
    BOOL bWaitAll,
    // 等待标志：TRUE表示所有对象同时可用，FALSE表示至少一个对象可用
        ;
    DWORD dwMilliseconds// 等待超时时限;
    );
```

互锁变量访问

- 互锁变量访问是最基本的互斥手段，其他的互斥和共享机制都是以它为基础的。它相当于硬件TS指令。用于对整型变量的操作，可避免线程间切换对操作连续性的影响。这组互锁变量访问API包括：
  - InterlockedExchange进行32位数据的先读后写原子操作；
  - InterlockedExchangePointer对指针的Exchange原子操作；
  - InterlockedCompareExchange依据比较结果进行赋值的原子操作；
  - InterlockedCompareExchangePointer对指针的CompareExchange原子操作；
  - InterlockedExchangeAdd先加后存结果的原子操作；
  - InterlockedDecrement先减1后存结果的原子操作；
  - InterlockedIncrement先加1后存结果的原子操作。
- 实例：InterlockedIncrement.cpp



临界区对象(Critical Section)

- 只能用于在同一进程内使用的临界区，同一进程内各线程对它的访问是互斥进行的。
- CRITICAL_SECTION（CS）对象可以被初始化和删除，但是没有句柄，不能被其他进程共享。变量应该被定义成为CRITICAL_SECTION类型。
- 线程可以进入和离开CS,而且在一个特定CS中，每次只能有一个线程。但是，一个线程可以在程序中的几个地方进入和离开CS。

Critical Section

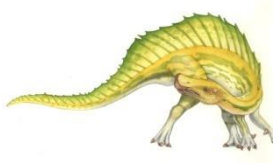
- CRITICAL_SECTION的优点是它不是内核对象，可在用户空间维护。相关API包括：
  - InitializeCriticalSection对临界区对象进行初始化；
  - EnterCriticalSection等待占用临界区的使用权，得到使用权时返回；
  - TryEnterCriticalSection测试或参看是否有一个线程CS对象；如果返回TRUE表明现在调用的线程占用CS，返回FALSE则表明一些其他的线程已经占用CS.
  - LeaveCriticalSection释放临界区的使用权；
  - DeleteCriticalSection释放与临界区对象相关的所有系统资源。
- 实例：CriticalSection.cpp

互斥对象(Mutex)

- 互斥对象相当于互斥信号量，在一个时刻只能被一个线程使用。
- 互斥对象提供的功能要超过CRITICAL_SECTION,因为互斥对象能够被命名和拥有句柄，所以它们也可以用于不同进程内的两个线程之间进行进程同步。例如两个通过内存映射文件共享内存的进程能够使用互斥对象来同步对共享内存的访问。
- 互斥对象类似于CS,但是，除了能够由进程共享之外，互斥对象还允许设置超时值，并且在被一个终止的进程放弃的时候会变成有信号状态。
- 一个线程通过等待互斥对象句柄(WaitForSingleObject或WaitForMultipleObjects)来获得对互斥对象的控制权（或“锁定”互斥对象）。使用ReleaseMutex来释放控制权。

Mutex

- 线程应该一有可能就释放它所占用的资源。一个线程可以多次获得一个特定的互斥对象；线程如果已经获得了控制权就不会被阻塞。最后，它必须以同样的次数释放互斥对象。
- 有关的API：
  - CreateMutex创建一个互斥对象，返回对象句柄；
  - OpenMutex返回一个已存在的互斥对象的句柄，用于后续访问；
  - ReleaseMutex释放对互斥对象的占用，使之成为可用；
- 实例：mutex.cpp



信号量对象(Semaphore)

- 信号量对象的取值在0到指定最大值之间，用于限制并发访问的线程数。
- 有关的API：
  - CreateSemaphore 创建一个信号量对象，指定最大值和初值，返回对象句柄；
  - OpenSemaphore 返回一个已存在的信号量对象的句柄，用于后续访问；
  - ReleaseSemaphore 释放对信号量对象的占用；
- 实例：SimplePC.cpp , Semaphore.cpp

事件对象(Event)

- 事件对象相当于"触发器"，可通知一个或多个线程某事件的出现。有关的API：
  - CreateEvent创建一个事件对象，返回对象句柄；
  - OpenEvent返回一个已存在的事件对象的句柄，用于后续访问；
  - SetEvent和PulseEvent设置指定事件对象为可用状态；
  - ResetEvent设置指定事件对象为不可用状态；手工复位，并唤醒所有等待线程；
- 实例：Event.cpp