**第一次课后作业**

1. **进程终止函数ExitProcess和TerminateProcess 有啥区别**

|  |  |  |
| --- | --- | --- |
| 函数名称 | ExitProcess | TerminateProcess |
| 功能描述 | 用于结束当前进程 | 用于结束指定的其他进程及其所有线程 |
| 调用方式 | 直接调用即可，无需参数或需指定退出代码 | 需要提供要结束的进程的句柄和退出代码 |
| 使用范围 | 只能用于结束调用它的进程本身 | 可以用于结束系统中的任何进程（需要适当的权限） |
| 清理工作 | 提供一个干净的进程关闭过程，包括调用所有附加的动态链接库（DLLs）的入口函数并通知它们进程即将终止 | 不通知附加的DLLs进程即将终止，可能导致资源未正确释放 |
| 返回值 | 不返回值 | 返回一个布尔值，指示进程是否成功终止 |
| 进程句柄管理 | 不涉及进程句柄管理 | 需要通过OpenProcess等函数获取要结束的进程的句柄 |
| 对子进程的影响 | 终止当前进程不会影响其子进程 | 终止一个进程不会自动终止其子进程 |
| 进程对象删除 | 进程对象在最后一个句柄关闭后才会被删除 | 同上 |

ExitProcess是结束进程最完美的方法，它会提供一个干净的进程关闭过程。这包括调用所有附加的动态链接库的入口函数，并传递一个值以指示进程正在从DLL分离。这样，DLL有机会执行任何必要的清理工作。然而，TerminateProcess则不会通知附加的DLL进程即将终止，因此可能会导致资源未正确释放。

此外，ExitProcess只能用于结束调用它的进程本身，而TerminateProcess则可以用于结束系统中的任何进程（当然，这需要适当的权限）。在调用TerminateProcess时，需要提供要结束的进程的句柄和退出代码。而ExitProcess则通常不需要提供这些参数，或者只需要指定一个退出代码即可。

1. **进程状态信息显示可以采用哪几个链接库**

**(1)PS API和Tool help API**

PS API是进程状态信息的一系列API合称，这些API在Psapi.h头文件和Psapi.lib 中

Tool help API 提供了一系列API用于获取执行中的应用程序的信息，一般与PS API配合使用，由Tlhelp32.h头文件，kernel32.dll

**(2)CreateToolhelp32Snapshot**

HANDLE WINAPI CreateToolhelp32Snapshot( DWORD dwFlags, DWORD th32ProcessID );

用于获取进程、线程或特定进程模块的快照

dwFlags 指定快照类型 例如TH32CS\_SNAPPROCESS 进程快照TH32CS\_SNAPTHREAD 线程快照

th32ProcessID 指定进程ID，当为0表示当前进程，该参数只有为TH32CS\_SNAPHEAPLIST 或者TH32CS\_SNAPMODULE 才有效，其它忽略。

**(3)Process32First**

用于获取系统快照中第一个进程

BOOL WINAPI Process32First( HANDLE hSnapshot, LPPROCESSENTRY32 lppe );

第一个参数hSnapshot 为快照句柄

第二个参数lppe指向结构体PROCESSENTRY32

BOOL WINAPI Process32Next( HANDLE hSnapshot, LPPROCESSENTRY32 lppe );

**(4)OpenProcess**

HANDLE OpenProcess(

DWORD dwDesiredAccess, // 指定进程访问标志

BOOL bInheritHandle, // 指定返回的句柄是否能被当做当前进程创建的新进程基础

DWORD dwProcessId // 打开进程的ID );

**(5)EnumProcesses**

获取当前所有进程的PID

BOOL EnumProcesses(

DWORD \*lpidProcess, // 输出参数，保持获取进程PID地址

DWORD cb, // lpidProcess所指向的数组大小

DWORD \*cbNeeded // 输出参数，获取PID的个数 );

备注：如果数组足够大，那么cbNeeded返回当前系统中的进程数，如果cbNeeded与cb相等，那么可能数组不能容纳所有进程PID

1. **临界区、信号量、互斥量、事件实现线程同步有啥区别？各自特点**

(1)临界区（Critical Section）是一种防止多个线程同时执行一个特定代码段的机制,多个线程访问同一个临界区的原则：

一次最多只能一个线程停留在临界区内；

不能让一个线程无限地停留在临界区内，否则其它线程将不能进入该临界区

缺点：不能在多进程之间对线程进行同步

(2)信号量内核对象用来对资源进行计数。与其他内核对象相同，它包括一个使用计数。但是它还包含另外两个值：一个是最大资源计数和当前资源计数。最大资源计数表示信号量可以控制的最大资源数量。当前资源计数表示信号量当前可用资源的数量

(3)互斥量内核对象用来确保一个线程独占第一个资源的访问。互斥量对象包括一个使用计数、线程ID以及一个递归计数。

互斥量与关键段行为相同，但是比关键段慢。同时意味着可以在不同进程中访问同一个互斥量

线程ID用来标识当前占有这个互斥量的是系统中哪个线程。递归计数表示这个线程占用该互斥量的次数。互斥量跟事件内核对象一样被经常使用。

(4)事件(Event)是WIN32提供的最灵活的线程间同步方式，事件存在两种状态：

触发状态(signaled or true)和未触发状态(unsignal or false)

一般设置为未触发状态，并由SetEvent设置为已触发状态。

当然也可以反着做，CreateEvent时设置为已触发状态，然后由ResetEvent设置为未触发状态

BOOL SetEvent(HANDLE hEvent);

BOOL ResetEvent(HANDLE hEvent);

1. 区别

临界区适用于同一进程内的快速同步；信号量适用于控制对有限数量资源的访问；互斥量适用于确保资源在同一时间内只能被一个线程（或进程）访问；事件适用于在特定事件发生时通知其他线程进行相应操作。

1. **事件的自动事件和手动事件设置有啥区别**

手动：这种对象只能用程序来手动设置，在需要该事件或者事件发生时，当手动事件被触发，正在等待事件所有线程都变得可调度

手动重置则需要调用ResetEvent函数设置为未通知状态

自动：一旦事件发生并被处理后，自动恢复到没有事件状态， 当一个自动事件被触发，只有一个正在等待线程可调度

如果跨进程访问事件，必须对事件命名