**目录**

[1 VirtualAlloc() 1](#_Toc475649155)

[2 VirtualFree() 1](#_Toc475649156)

[3 VirtualLock() 1](#_Toc475649157)

[4 CreateEvent() 2](#_Toc475649158)

[5 WaitForSingleObject() 2](#_Toc475649159)

[6 SetFilePointer() 2](#_Toc475649160)

[7 CreateThread() 3](#_Toc475649161)

[8 SetThreadPriority() 4](#_Toc475649162)

[9 ZeroMemory 4](#_Toc475649163)

# 1 VirtualAlloc()

VirtualAlloc是一个Windows API函数，该函数的功能是在调用进程的虚地址空间,预定或者提交一部分页。简单点的意思就是申请内存空间。

LPVOID WINAPI VirtualAlloc(

\_\_in\_opt LPVOID lpAddress, //分配的起始位置。

//如果要保留一段内存区域，函数会将其自动向最近的一个分配粒度对齐；

//如果要提交一段内存区域，函数将会向最近的一个页面对齐；

//如果为NULL，系统自行决定在什么地方分配

\_\_in SIZE\_T dwSize, //所需要分配的内存字节大小，

\_\_in DWORD flAllocationType, //分配类型：MEM\_COMMIT(提交)、MEM\_RESERVED(保留)..

\_\_in DWORD flProtect //内存保护属性：PAGE\_READWRITE、PAGE\_EXECUTE…

);

返回值：

成功时，返回指向分配到的内存的起始地址的指针；

失败时，返回NULL。

LPVOID VirtualAlloc{

LPVOID lpAddress, // 要分配的内存区域的地址

DWORD dwSize, // 分配的大小

DWORD flAllocationType, // 分配的类型

DWORD flProtect // 该内存的初始保护属性

};

# 2 VirtualFree()

VirtualFree函数将当前进程内存状态从提交变为保留，或将保留变为空闲，或同时进行：

BOOL WINAPI VirtualFree(

\_\_in LPVOID lpAddress, //需要改变状态的内存区域的起始地址

\_\_in SIZE\_T dwSize, //需要改变状态的内存区域的字节大小

\_\_in DWORD dwFreeType //MEM\_DECOMMIT—将内存变为保留状态

//MEM\_RELEASE—释放内存，将内存变为空闲状态

);

返回值：

成功时，返回非零值；

失败时，返回零值。

# 3 VirtualLock()

VirtualLock() 函数可以将虚拟地址空间中内存页面的一片区域锁定在物理内存里。它有两个参数，第 1 个参数 lpAddress 是要锁定内存页的基地址，第 2 个参数 dwSize 是要锁定的区域大小，以字节为单位。在锁定的过程中，会按照页面的对齐机制进行锁定。比如说，给定的基地址 lpAddress 如果落在一个页面的中间，而 dwSize 所指的偏移不超过该页面，那么就将 lpAddress + dwSize 所在的那个页面锁起来；若是 dwSize 已经造成跨页(假设跨到下一页)，那么就将这两个页都锁到内存中。

注意，所有要锁定的页面必须是已”提交的“。使用 VirtualLock() 可能会造成性能的损失，因为它会造成可用内存减少，这样一来对于另外的一些关键内存页就不得不交换到磁盘上。但是如果程序所运行的机器拥有巨大的物理内存，那么从这个角度来讲，还提高了性能。像在这些拥有巨大物理内存的机器上，我们可能会比较喜欢手动禁用掉所有的虚拟内存，也就是设置磁盘上的分页文件为 0 MB，这就意味着，我们自信物理内存足以胜任运行我们所需要的工作程序。然而，像一些游戏，他们依然会需要虚拟内存，如果之前就已经禁止掉所有的虚拟内存，那么游戏就会运行失败。

实际上，每个版本的 Windows 都有一个底线，即对于一个进程来说，它能够锁定的内存是有个上限值的，这也避免了造成系统性能的巨大损失。

解除锁定使用 VirtualUnlock() 函数，一般的，该函数中的参数要和 VirtualLock() 一样，它们成对使用。

# 4 CreateEvent()

用来创建或打开一个命名的或无名的事件对象

HANDLE CreateEvent(

LPSECURITY\_ATTRIBUTES lpEventAttributes,// 安全属性

BOOL bManualReset,// 复位方式

BOOL bInitialState,// 初始状态

LPCTSTR lpName // 对象名称

);

# 5 WaitForSingleObject()

1. 格式

DWORD WaitForSingleObject( HANDLE hHandle, DWORDdwMilliseconds);

有两个参数，分别是THandle和Timeout(毫秒单位)。

如果想要等待一条线程，那么你需要指定线程的Handle，以及相应的Timeout时间。当然，如果你想无限等待下去，Timeout参数可以指定系统常量INFINITE。

2. 使用对象

它可以等待如下几种类型的对象：

Event，Mutex，Semaphore，Process，Thread

3. 返回类型

有三种返回类型：

WAIT\_OBJECT\_0, 表示等待的对象有信号（对线程来说，表示执行结束）；

WAIT\_TIMEOUT, 表示等待指定时间内，对象一直没有信号（线程没执行完）；

WAIT\_ABANDONED 表示对象有信号，但还是不能执行 一般是因为未获取到锁或其他原因

# 6 SetFilePointer()

DWORD SetFilePointer(

HANDLE hFile, // 文件句柄

LONG lDistanceToMove, // 偏移量(低位)

PLONG lpDistanceToMoveHigh, // 偏移量(高位)

DWORD dwMoveMethod // 基准位置FILE\_BEGIN:文件开始位置 FILE\_CURRENT:文件当前位置 FILE\_END:文件结束位置

说明:移动一个打开文件的指针

# 7 CreateThread()

在主线程的基础上创建一个新线程。

WINBASEAPI

\_Ret\_maybenull\_

HANDLE

WINAPI

CreateThread(

\_In\_opt\_ LPSECURITY\_ATTRIBUTES lpThreadAttributes,

\_In\_ SIZE\_T dwStackSize,

\_In\_ LPTHREAD\_START\_ROUTINE lpStartAddress,

\_In\_opt\_ \_\_drv\_aliasesMem LPVOID lpParameter,

\_In\_ DWORD dwCreationFlags,

\_Out\_opt\_ LPDWORD lpThreadId

);

HANDLECreateThread(

LPSECURITY\_ATTRIBUTESlpThreadAttributes,//线程安全属性

DWORDdwStackSize,//堆栈大小

LPTHREAD\_START\_ROUTINElpStartAddress,//线程函数

LPVOIDlpParameter,//线程参数

DWORDdwCreationFlags,//线程创建属性

LPDWORDlpThreadId//线程ID

);

参数说明：

lpThreadAttributes：指向SECURITY\_ATTRIBUTES型态的结构的指针。在Windows 98中忽略该参数。在Windows NT中，NULL使用默认安全性，不可以被子线程继承，否则需要定义一个结构体将它的bInheritHandle成员初始化为TRUE

dwStackSize，设置初始栈的大小，以字节为单位，如果为0，那么默认将使用与调用该函数的线程相同的栈空间大小。任何情况下，Windows根据需要动态延长堆栈的大小。

lpStartAddress，指向线程函数的指针，形式：@函数名，函数名称没有限制，但是必须以下列形式声明：

DWORD WINAPI 函数名 (LPVOID lpParam) ，格式不正确将无法调用成功。

//也可以直接调用void类型

//但lpStartAddress要这样通过LPTHREAD\_START\_ROUTINE转换如： (LPTHREAD\_START\_ROUTINE)MyVoid

//然后在线程声明为：

void MyVoid()

{

return;

}

lpParameter：向线程函数传递的参数，是一个指向结构的指针，不需传递参数时，为NULL。

dwCreationFlags ：线程标志,可取值如下

（1）CREATE\_SUSPENDED(0x00000004)：创建一个挂起的线程，

（2）0：表示创建后立即激活。

（3）STACK\_SIZE\_PARAM\_IS\_A\_RESERVATION(0x00010000)：dwStackSize参数指定初始的保留堆栈 的大小，否则,dwStackSize指定提交的大小。该标记值在Windows 2000/NT and Windows Me/98/95上不支持。

lpThreadId:保存新线程的id。

返回值：函数成功，返回线程句柄；函数失败返回false。若不想返回线程ID,设置值为NULL。

函数说明：

创建一个线程。

语法：

hThread = CreateThread (&security\_attributes, dwStackSize, ThreadProc,pParam, dwFlags, &idThread) ;

一般并不推荐使用 CreateThread函数，而推荐使用RTL库里的System单元中定义的 BeginThread函数，因为这除了能创建一个线程和一个入口函数以外，还增加了几项保护措施。

在MFC程序中，应该调用AfxBeginThread函数，在Visual C++程序中应调用\_beginthreadex函数。

# 8 SetThreadPriority()

设置指定线程的优先级

WINBASEAPI

BOOL

WINAPI

SetThreadPriority(

\_In\_ HANDLE hThread, //线程句柄

\_In\_ int nPriority //线程的优先级

);



# 9 ZeroMemory

ZeroMemory宏用0来填充一块内存区域。

为了避免优化编译器的意外的影响，请使用SecureZeroMemory函数。

void ZeroMemory(

PVOID Destination,

SIZE\_T Length

);

参数：

Destination :指向一块准备用0来填充的内存区域的开始地址。

Length :准备用0来填充的内存区域的大小，按字节来计算。

返回值：无

* CopyMemory

CopyMemory()函数功能描述：将一块内存的数据从一个位置复制到另一个位置

VOID CopyMemory(

PVOID Destination,

CONST VOID \*Source,

SIZE\_T Length

);

Destination

要复制内存块的目的地址。

Source

要复制内存块的源地址。

Length

指定要复制内存块的大小，单位为字节

返回值

该函数为VOID型，没有返回值。

1. DirectoryInfo

公开用于创建、移动和枚举目录和子目录的实例方法。无法继承此类。

命名空间:System.IO

程序集:mscorlib（在 mscorlib.dll 中）

|  |  |
| --- | --- |
| 方法名 | 说明 |
| GetFiles() | 返回当前目录的文件列表 |