


三、Windows里面的进程（Process）和线程（Thread）

原创 huichangxindong 2018-04-16 14:57:48 1599 收藏 4 版权

分类专栏：[深入浅出MFC](#)

 深入浅出MFC 专栏收录该内容

0 订阅 3 篇文章 [订阅专栏](#)

首先，我们应该知道，windows调度的单位是线程而不是进程！

从而，我们认识一下核心对象（系统资源）

核 心 对 象	产 生 方 法
event	CreateEvent
mutex	CreateMutex
semaphore	CreateSemaphore
file	CreateFile
file-mapping	CreateFileMapping
process	CreateProcess
thread	CreateThread

进程的生死周期：

1. shell 调用 CreateProcess 激活 App.exe。
2. 系统产生一个“进程核心对象”，计数值为 1。
3. 系统为此进程建立一个 4GB 地址空间。
4. 加载器将必要的代码加载到上述地址空间中，包括 App.exe 的程序、数据，以及所需的动态链接函数库（DLLs）。加载器如何知道要加载哪些 DLLs 呢？它们被记录在可执行文件（PE 文件格式）的 .idata section 中。
5. 系统为此行程建立一个线程，称为主线程（primary thread）。线程才是 CPU 时间的分配对象。
6. 系统调用 C runtime 函数库的 Startup code。
7. Startup code 调用 App 程序的 WinMain 函数。
8. App 程序开始运行。
9. 使用者关闭 App 主窗口，使 WinMain 中的消息循环结束掉，于是 WinMain 结束。
10. 回到 Startup code。
11. 回到系统，系统调用 ExitProcess 结束进程。

当然，你也可以编写一个程序，使用CreateProcess函数创建一个新的进程去调用其他的程序。

```
1  BOOL CreateProcess(  
2      LPCTSTR lpApplicationName, // pointer to name of executable module  
3      LPTSTR lpCommandLine, // pointer to command line string  
4      LPSECURITY_ATTRIBUTES lpProcessAttributes, // process security attributes  
5      LPSECURITY_ATTRIBUTES lpThreadAttributes, // thread security attributes  
6      BOOL bInheritHandles, // handle inheritance flag  
7      DWORD dwCreationFlags, // creation flags  
8      LPVOID lpEnvironment, // pointer to new environment block  
9      LPCTSTR lpCurrentDirectory, // pointer to current directory name  
10     LPSTARTUPINFO lpStartupInfo, // pointer to STARTUPINFO  
11     LPPROCESS_INFORMATION lpProcessInformation // pointer to PROCESS_INFORMATION  
12 );
```

以下是《深入浅出MFC中的参数解析》，详细的可查MSDN

第一个参数 *lpApplicationName* 指定可执行文件名。第二个参数 *lpCommandLine* 指定欲传给新进程的命令行（command line）参数。如果你指定了 *lpApplicationName*，但没有扩展名，系统并不会主动为你加上 .EXE 扩展名；如果没有指定完整路径，系统就只在当前工作目录中寻找。但如果你指定 *lpApplicationName* 为 *NULL* 的话，系统会以 *lpCommandLine* 的第一个“段落”（我的意思其实是术语中所谓的 token）作为可执行文件名；如果这个文件名没有指定扩展名，就采用默认的 “.EXE” 扩展名；如果没有指定路径，Windows 就依照五个搜寻路径来寻找可执行文件，分别是：blog.csdn.net/huichangxindong

- 1. 调用者的可执行文件所在目录
- 2. 调用者的当前工作目录
- 3. Windows 目录
- 4. Windows System 目录
- 5. 环境变量中的 path 所设定的各目录

让我们看看实例：

```
CreateProcess("E:\\CWIN95\\NOTEPAD.EXE", "README.TXT",...);
```

系统将执行 E:\CWIN95\NOTEPAD.EXE，命令行参数是 “README.TXT”。如果我们这样子调用：

```
CreateProcess(NULL, "NOTEPAD README.TXT",...);
```

系统将依照搜寻次序，执行第一个被找到的 NOTEPAD.EXE，并传送命令行参数 "README.TXT" 给它。<https://blog.csdn.net/huichangxindong>

建立新进程之前，系统必须做出两个核心对象，也就是“进程对象”和“线程对象”。*CreateProcess* 的第三个参数和第四个参数分别指定这两个核心对象的安全属性。至于第五个参数（*TRUE* 或 *FALSE*）则用来设定这些安全属性是否要被继承。关于安全属性及其可被继承的性质，碍于本章的定位，我不打算在此介绍。

第六个参数 *dwCreationFlags* 可以是许多常数的组合，会影响到进程的建立过程。这些常数中比较常用的是 *CREATE_SUSPENDED*，它会使得子进程产生之后，其主线程立刻被暂停执行。

第七个参数 *lpEnvironment* 可以指定进程所使用的环境变量区。通常我们会让子行程继承父行程的环境变量，那么这里要指定 *NULL*。

第八个参数 *lpCurrentDirectory* 用来设定子进程的工作目录与工作驱动器。如果指定 *NULL*，子进程就会使用父进程的工作目录与工作驱动器。

第九个参数 *lpStartupInfo* 是一个指向 *STARTUPINFO* 结构的指针。这是一个庞大的结构，可以用来设定窗口的标题、位置与大小，详情请看 [API 使用手册](https://blog.csdn.net/huichangxindong)。<https://blog.csdn.net/huichangxindong>

最后一个参数是一个指向 *PROCESS_INFORMATION* 结构的指针：

```
typedef struct _PROCESS_INFORMATION {
    HANDLE hProcess;
    HANDLE hThread;
    DWORD dwProcessId;
    DWORD dwThreadId;
} PROCESS_INFORMATION;
```

当系统为我们产生“进程对象”和“线程对象”时，它会把两个对象的 handle 填入此结构的相关字段中，应用程序可以从这里获得这些 handles。

如果一个进程想结束自己的生命，只要调用：

```
VOID ExitProcess(UINT fuExitCode);
```

就可以了。如果进程想结束另一个进程的生命，可以使用：

```
BOOL TerminateProcess(HANDLE hProcess, UINT fuExitCode);
```

很显然，只要你有某个进程的 handle，就可以结束它的生命。*TerminateProcess* 并不被建议使用，倒不是因为它的权力太大，而是因为一般进程结束时，系统会通知该进程所开启（所使用）的所有 DLLs，但如果你以 *TerminateProcess* 结束一个进程，系统不会做这件事，而这恐怕不是你所希望的。<https://blog.csdn.net/huichangxindong>

按理来说，shell创建了一个新进程，那么他就是shell的子进程了，然而实际上，shell在创建了一个新进程之后，它就切断了该进程与自己的父子关系，如下使用CloseHandle就可以实现上面的操作：

```
PROCESS_INFORMATION ProcInfo;
BOOL fSuccess;
fSuccess = CreateProcess(...,&ProcInfo);
if (fSuccess) {
    CloseHandle(ProcInfo.hThread);
    CloseHandle(ProcInfo.hProcess);
}
```

<https://blog.csdn.net/huichangxindong>

线程的生死周期：

执行程序代码，是线程的工作。当一个进程建立起来后，主线程也产生。所以每一个Windows 程序一开始就有了一个线程。我们可以调用 *CreateThread* 产生额外的线程，系统会帮我们完成下列事情：

- 1. 配置“线程对象”，其 `handle` 将成为 *CreateThread* 的返回值。
- 2. 设定计数值为 1。
- 3. 配置线程的 `context`。
- 4. 保留线程的堆栈。
- 5. 将 `context` 中的堆栈指针寄存器（SS）和指令指针寄存器（IP）设定妥当。

我们使用CreateThread函数创建一个线程在调用进程的地址空间。

```
1 HANDLE CreateThread(  
2     LPSECURITY_ATTRIBUTES lpThreadAttributes, // 线程的安全属性  
3     DWORD dwStackSize, // 初始化线程的栈大小  
4     LPTHREAD_START_ROUTINE lpStartAddress, // 线程需要处理的函数（content）  
5     LPVOID lpParameter, // 线程处理函数的参数  
6     DWORD dwCreationFlags, // 线程创建的附加标识  
7     LPDWORD lpThreadId // 返回线程的id  
8 );
```

下面是MSDN里面的参数解析

Parameters

lpThreadAttributes

指向一个SECURITY_ATTRIBUTES 结构的 that确定返回的句柄是否可以由子进程继承。如果被子进程是null，手柄不能被继承。

Windows NT: 该结构的lpSecurityDescriptor成员指定为新的线程安全描述符。如果被子进程是空的，线程获取默认安全描述符。

dwStackSize

指定堆栈的初始提交大小，以字节为单位。系统将该值旋转到最近的页面。如果该值为零，或者小于默认提交大小，则默认使用与调用线程相同的大小。有关更多信息，请参见线程堆栈大小。

lpStartAddress

应用程序定义的函数指针类型LPTHREAD_START_ROUTINE被线程执行的线程的起始地址。在线程函数的更多信息。

lpParameter

指定传递给线程的单个32位参数值。

dwCreationFlags

指定控制线程创建的附加标志。如果create_suspended标志指定, 线程处于挂起状态了，也不会跑到resumethread函数被调用。如果此值为零，则线程在创建后立即运行。此时，没有其他值被支持。

lpThreadId

指向接收线程标识符的32位变量的指针。.

Windows NT：如果此参数为空，则不返回线程标识符。

Windows 95和Windows 98：此参数可能不是null。

返回值：

如果函数创建成功，则返回该线程句柄，如果失败，返回NULL。错误信息调取函数GetLastError。

与此同时，我们不能忘了C runtime函数库里面也有一个创建线程的方法：_beginthreadex

```
1 uintptr_t _beginthread( // NATIVE CODE  
2     void( __cdecl *start_address )( void * ),  
3     unsigned stack_size,  
4     void *arglist  
5 );  
6 uintptr_t _beginthread( // MANAGED CODE  
7     void( __clrcall *start_address )( void * ),  
8     unsigned stack_size,  
9     void *arglist  
10 );  
11 uintptr_t _beginthreadex( // NATIVE CODE  
12     void *security,  
13     unsigned stack_size,  
14     unsigned ( __stdcall *start_address )( void * ),  
15     void *arglist,  
16     unsigned initflag,  
17     unsigned *thrdaddr  
18 );  
19 uintptr_t _beginthreadex( // MANAGED CODE  
20     void *security,  
21     unsigned stack_size,  
22     unsigned ( __clrcall *start_address )( void * ),  
23     void *arglist,  
24     unsigned initflag,  
25     unsigned *thrdaddr  
26 );
```

参数

start_address

启动开始执行新线程的例程的地址。对于 _beginthread，调用约定是 __cdecl（针对本机代码）或 __clrcall（针对托管代码）；对于 _beginthreadex，它是 __stdcall（针对本机代码）或 __clrcall（针对托管代码）。

stack_size

新线程的堆栈大小或 0。

arglist

要传递到新线程的参数列表或 NULL。

Security

指向 SECURITY_ATTRIBUTES 结构的指针，此结构确定返回的句柄是否由子进程继承。如果 Security 为 NULL，则不能继承句柄。对于 Windows 95 应用程序，必须为 NULL。

initflag

控制新线程的初始状态的标志。将 initflag 设置为 0 以立即运行，或设置为 CREATE_SUSPENDED 以在挂起状态下创建线程；使用 ResumeThread来执行此线程。将 initflag 设置为 STACK_SIZE_PARAM_IS_A_RESERVATION 标志以将 stack_size 用作堆栈的初始保留大小（以字节计）；如果未指定此标志， stack_size 将指定提交大小。

thrdaddr

指向接收线程标识符的 32 位变量。如果此变量为 NULL，则不可用。

返回值

如果成功，则这些函数中的每一个都会返回一个句柄到新创建的线程；但是，如果新创建的线程退出过快，则 _beginthread 可能不会返回有效句柄。（请参见“备注”节中的讨论。）发生错误时，_beginthread 返回 -1L，并在线程过多的情况下将 errno 设置为 EAGAIN；如果参数无效或堆栈大小错误，则设置为 EINVAL；如果资源（如内存）不足，则设置为 EACCES。发生错误时，_beginthreadex 返回 0 并设置 errno 和 _doserrno。

如果 startaddress 为 NULL，则会调用无效的参数处理程序，如 Parameter Validation所述。如果允许执行继续，则这些功能将 errno 设置为EINVAL 并返回 -1。

线程优先级：

线程的优先级范围从 0（最低）到 31（最高）。当你产生线程时，并不是直接以数值指定其优先级，而是采用两个步骤。第一个步骤是指定“优先级等级（Priority Class）”给行程，第二步是指定“相对优先级”给该进程所拥有的线程。**图 1-7** 是对优先级等级的描述，其中的代码在 *CreateProcess* 的 *dwCreationFlags* 参数中指定。如果你不指定，系统默认给的是 *NORMAL_PRIORITY_CLASS*，除非父进程是 *IDLE_PRIORITY_CLASS*（那么子进程也会是 *IDLE_PRIORITY_CLASS*）。

等 级	代 码	优 先 级 值
idle	IDLE_PRIORITY_CLASS	4
normal	NORMAL_PRIORITY_CLASS	9（前台）或 7（后台）
high	HIGH_PRIORITY_CLASS	13
realtime	REALTIME_PRIORITY_CLASS	24

- “idle” 等级只有在 CPU 时间将被浪费掉时（也就是前一节所说的空闲时间）才执行。该等级最适合于系统监视软件，或屏幕保护软件。
- “normal” 是默认等级。系统可以动态改变优先级，但只限于 “normal” 等级。当进程变成前台时，线程优先级提升为 9，当进程变成后台时，优先级降低为 7。
- “high” 等级是为了满足立即反应的需要，例如使用者按下 Ctrl+Esc 时立刻把工作管理器（task manager）带出场。
- “realtime” 等级几乎不会被一般的应用程序使用。就连系统中控制鼠标、键盘、驱动器状态重新扫描、Ctrl+Alt+Del 等的线程都比 “realtime” 的优先级还低。这种等级使用在“如果不在某个时间范围内被执行的话，数据就要遗失”的情况。这个等级一定得在正确评估之下使用，如果你把这样的等级指定给一般的（并不会常常被阻塞的）线程，多任务环境恐怕会瘫痪，因为这个线程有如此高的优先级，其它线程再没有机会被执行。

<i>SetThreadPriority</i> 的 参 数	微 调 幅 度
<i>THREAD_PRIORITY_LOWEST</i>	-2
<i>THREAD_PRIORITY_BELOW_NORMAL</i>	-1
<i>THREAD_PRIORITY_NORMAL</i>	不变
<i>THREAD_PRIORITY_ABOVE_NORMAL</i>	+1
<i>THREAD_PRIORITY_HIGHEST</i>	+2

<i>SetThreadPriority</i> 的 参 数	面对任何等级的调整结果：	面对“realtime”等级的调整结果：
<i>THREAD_PRIORITY_IDLE</i>	1	16
<i>THREAD_PRIORITY_TIME_CRITICAL</i>	15	31

总结来看，其优先级可以看做：

优先级等级	idle	lowest	below normal	normal	above normal	highest	time critical
idle	1	2	3	4	5	6	15
normal (后台)	1	5	6	7	8	9	15
normal (前台)	1	7	8	9	10	11	15
high	1	11	12	13	14	15	15
realtime	16	22	23	24	25	26	31

ProcessWindow 08-03
一个包含进度条 取消按钮的对话框程序

windows 一个进程(Process)最多可以生成多少个线程(Thread) huapeng_guo的专栏 9452
1.进程中创建线程的限制 默认情况下，一个线程的栈要预留1M的内存空间,而一个进程中可用的内存空间只有2G， ...

 优质评论可以帮助作者获得更高权重

抢沙发

评论

windows中的进程和线程_GoodIdea 10-22
在讨论windows下的进程和线程时,我们先回顾下通用操作系统的进程和线程。之所以称之为通用是因为一贯的本科...

Windows和Linux进程与线程的区别_a3192048的博客 8-16
对于windows来说,进程和线程的概念都是有着明确定义的,进程的概念对应于一个程序的运行实例(instance),而线程...

C# 进程(Process)与线程(Thread)的理解及运用 最新发布 小程小程，永不消沉 331
线程的理解及运用一、进程、线程和协程的理解1、进程、线程、协程的定义2、串行，并行和并发的基本概念二、 ...

进程（process）和线程（thread）介绍 liitdar的博客 5537
本文主要介绍进程（process）和线程（thread）的相关知识。 1 Why 1.1 为了整合资源 一开始，CPU 只有在执行...

进程和线程的区别(Windows)_zephyr_be_brave的专栏 10-2
进程和线程的区别(Windows) 转自http://www.cnblogs.com/lmule/archive/2010/08/18/1802774.html 简而言之,一个程...

Windows进程与线程---1_For Geek 9-30
相比于Linux中的进程管理,即创建一个线程的同时创建一个进程。而Windows中却不是这样,它是先创建一个进程作...

一个进程(Process)最多可以生成多少个线程(Thread) 路在脚下 5394
1.进程中创建线程的限制 默认情况下，一个线程的栈要预留1M的内存空间,而一个进程中可用的内存空间只有2G， ...

进程(Process)和线程(Thread) Forrest He的专栏 2370
进程(Process)和线程(Thread)1.Process特点(1)进程在执行过程中有内存单元的初始入口点，并且进程存活过程中始...

Windows进程与线程_cw312644167的博客 9-25
Windows进程与线程 进程的组成 进程是惰性的进程要做任何事情，都必须让线程来运行线程执行进程地址空间中包...

Windows Process内存组织结构及重要域解析 lbq1221119 730
最近恶补操作系统和一些底层的知识。遂写篇文章来说从操作系统的角度来研究Process的一些结构，实现，Porc...

全量窗口函数ProcessWindowFunction
ProcessWindowFunction 在某些情况下，统计更复杂的指标可能需要

 huichangxindong

关注

 1

 0

 4



进程process pid与线程 (一) ——进程

一直走，不要回头 4071

进程，是对正在运行的程序的一个抽象。进程由三部分组成 程序段 相关的数据段 PCB ——进程控制块，即 P...

Windows下查看进程中的线程

12-22

Windows下查看进程中的线程

windows线程与进程的关系

hubinbin595959的专栏 898

线程与进程的关系 一般将**进程**定义成一个正在运行的程序的一个实例，它由以下两部分构成。一个内核对象，操作...

windows中的进程和线程

u012252959的博客 1786

在讨论windows下的进程和线程时，我们先回顾下通用操作系统的进程和线程。之所以称之为通用是因为一贯的本...

windows下进程与线程

weixin_30369087的博客 30

Windows 下进程与线程 Windows 是一个单用户多任务的操作系统，同一时间可有多个进程在执行。进程是应用程序...

Windows下进程与线程

HHHUANG 743

进程 1. 进程的概念 进程就是操作系统上一个正在运行的程序的一个实例。由两部分构成：一个内核对象，操作系...

Windows和Linux下进程、线程理解

super_chris的专栏 3813

对于windows来说，进程和线程的概念都是有着明确定义的，进程的概念对应于一个程序的运行实例(instance)，而...

Thread.currentThread () 方法、进程、线程、多线程相关总结 (二)

he_zhen_的博客 6629

`Thread.currentThread`方法 `Thread`的静态方法`currentThread`方法可以用于获取运行当前代码片段的线程

用dos 命令查看进程中的线程

dolphin_ygj 246

pslist是用命令行查看线程; ProcessExplorer是图形化的查看线程, 都在附件中。 1.查看进程 &...

C#与NET实战 第5章 进程、线程与同步 节选

weixin_30628801的博客 254

5.1 简介 进程 (process) 是一块包含了某些资源的内存区域。操作系统利用进程把它的工作划分为一些功能单元。 ...

©2021 CSDN 皮肤主题: 大白 设计师: CSDN官方博客 返回首页

关于我们 招贤纳士 广告服务 开发助手 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00

公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心
网络110报警服务 中国互联网举报中心 家长监护 Chrome商店下载 ©1999-2021北京创新乐知网络技术有限公司 版权与免责声明 版权申诉
出版物许可证 营业执照



huichangxindong

码龄5年

暂无认证

12

96万+

143万+

1万+



原创

周排名

总排名

访问

等级

287

5

5

2

16

积分

粉丝

获赞

评论

收藏



私信

关注

搜博主文章



热门文章

C/C++类型别名  4068

C++中的文件尾条件--EOF  3965

C/C++ 数据格式化  2226

三、Windows里面的进程（Process）和线程（Thread）  1596

B树B+树  1483

分类专栏



数据结构与算法

1篇



C-C++

4篇



数据库

1篇



网络编程

1篇



编程之美

1篇



深入浅出MFC

3篇

huichangxindong 关注 1 0 4

C++网络编程之Socket编程
~~~\*\*\*~~~: 请问硬件电路怎么连接呢，网线什么的  
B树B+树  
Super-Child: 请问作者有什么好的学习数据结构的书推荐的吗?

您愿意向朋友推荐“博客详情页”吗?











强烈不推荐 不推荐 一般般 推荐 强烈推荐

**最新文章**

二、 Console 程序

一、 WIN32程序

中国象棋将帅问题

2018年 5篇

2017年 7篇