

课程名称	操作系统	课程编号	A2130330
实验地点	综合实验楼 A511/A512	实验时间	2019-04-29
校外指导教师		校内指导教师	常光辉
实验名称	实验一 进程控制描述与控制		
评阅人签字		成绩	

一、实验目的

实验 1.1 Windows “任务管理器” 的进程管理

通过在 Windows 任务管理器中对程序进程进行响应的管理操作，熟悉操作系统进程管理的概念，学习观察操作系统运行的动态性能。

实验 1.2 Windows Server 2016 进程的“一生”

- 1) 通过创建进程、观察正在运行的进程和终止进程的程序设计和调试操作，进一步熟悉 操作系统的进程概念，理解 Windows Server 2016 进程的“一生”；
- 2) 通过阅读和分析实验程序，学习创建进程、观察进程和终止进程的程序设计方法。

二、工具/准备工作

1. 回顾教材相关内容；
2. 在 VMware WorkStation Pro 中安装 Windows Server 2016，如图 1-1 到图 1-4；

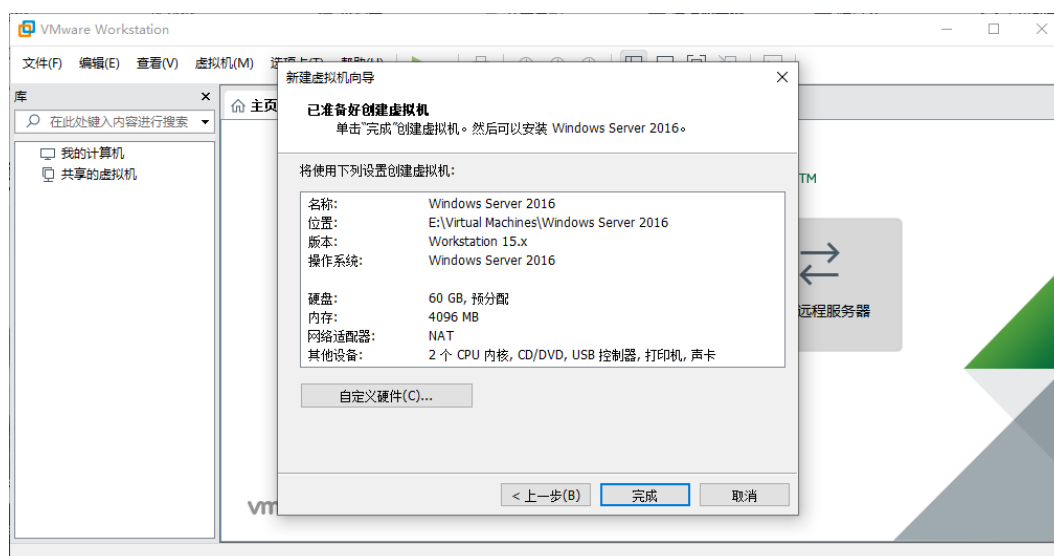


图 1-1 新建虚拟机

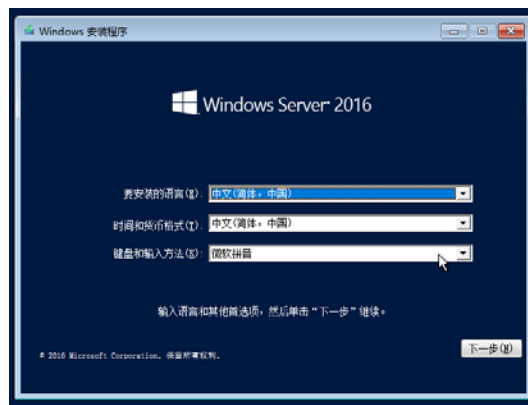


图 1-2 Windows Server 2016 安装程序



图 1-3 Windows Server 2016 安装过程



图 1-4 Windows Server 2016 安装完成

3. 并在系统中安装 Visual Studio 2019 或 Visual C++ 6.0 或其他 C++编译软件。

三、实验环境

操作系统: Windows Server 2016 (虚拟机)

编程语言: C++

集成开发环境: Visual Studio 2019

四、实验步骤与实验过程

实验 1.1 Windows “任务管理器” 的进程管理

启动并进入 Windows 环境，单击 Ctrl + Alt + Del 键¹，或者右键单击任务栏，在快捷菜单中单击“任务管理器”命令，打开“任务管理器”窗口²。

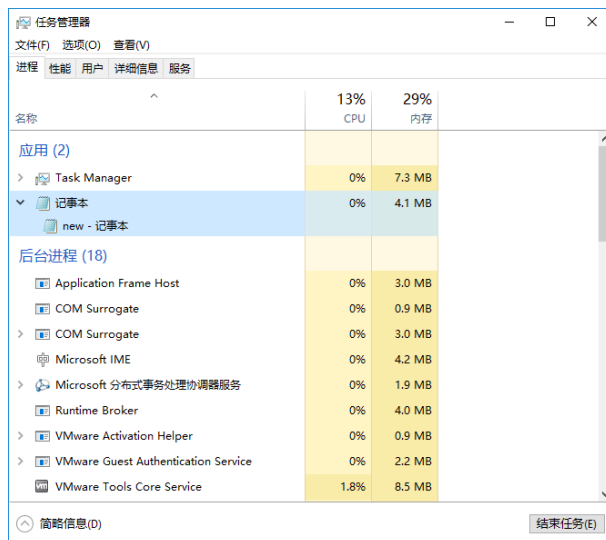


图 1-5 任务管理器

在本次实验中，使用的操作系统版本是：Windows Server 2016

当前机器中由你打开，正在运行的应用程序有：

- 1) Task Manager （任务管理器，即当前应用）
- 2) 记事本（打开了测试文件 new.txt，如图 1-6 所示）

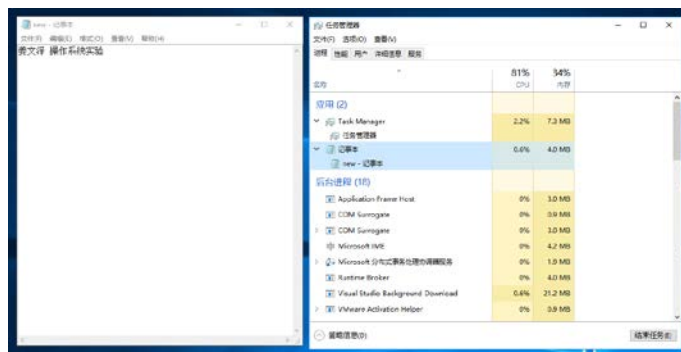


图 1-6 记事本

Windows “任务管理器” 的窗口由5 个选项卡组成，分别是：

- 1) 进程
- 2) 性能
- 3) 用户
- 4) 详细信息
- 5) 服务

当前“进程”选项卡显示的栏目分别是（可移动窗口下方的游标/箭头，或使窗口

最大化进行观察)：

- 1) 应用
- 2) 后台进程
- 3) Windows 进程

1. 使用任务管理器终止进程

步骤 1：单击“进程”选项卡，一共显示了 45 个进程。请试着区分一下，其中：系统（SYSTEM）进程有 25 个，如图 1-7，并将数据填入表 1-1 中。

名称	PID	状态	用户名	CPU	内存(专用...	描述
MsMpEng.exe	1904	正在运行	SYSTEM	01	64,112 K	Antimalware Service Executable
csrss.exe	460	正在运行	SYSTEM	00	1,144 K	Client Server Runtime Process
csrss.exe	584	正在运行	SYSTEM	01	1,324 K	Client Server Runtime Process
dllhost.exe	2412	正在运行	SYSTEM	00	2,872 K	COM Surrogate
lsass.exe	708	正在运行	SYSTEM	00	4,280 K	Local Security Authority Process
System	4	正在运行	SYSTEM	01	28 K	NT Kernel & System
vmacthlp.exe	1076	正在运行	SYSTEM	00	916 K	VMware Activation Helper
VGAUTHService.exe	1932	正在运行	SYSTEM	00	2,188 K	VMware Guest Authentication Service
vmtoolsd.exe	1896	正在运行	SYSTEM	00	3,340 K	VMware Tools Core Service
TiWorker.exe	3824	正在运行	SYSTEM	02	258,376 K	Windows Modules Installer Worker
winlogon.exe	636	正在运行	SYSTEM	00	1,124 K	Windows 登录应用程序
svchost.exe	796	正在运行	SYSTEM	00	4,572 K	Windows 服务主进程
svchost.exe	988	正在运行	SYSTEM	00	6,192 K	Windows 服务主进程
svchost.exe	1200	正在运行	SYSTEM	00	19,984 K	Windows 服务主进程
svchost.exe	1744	正在运行	SYSTEM	00	6,296 K	Windows 服务主进程
svchost.exe	1836	正在运行	SYSTEM	00	3,604 K	Windows 服务主进程
svchost.exe	1844	正在运行	SYSTEM	00	1,588 K	Windows 服务主进程
smss.exe	360	正在运行	SYSTEM	00	268 K	Windows 会话管理器
TrustedInstaller.exe	3708	正在运行	SYSTEM	00	1,264 K	Windows 模块安装程序
wininit.exe	576	正在运行	SYSTEM	00	732 K	Windows 启动应用程序
WmiPrvSE.exe	2936	正在运行	SYSTEM	00	8,144 K	WMI Provider Host
系统空闲进程	0	正在运行	SYSTEM	90	4 K	处理器空闲时间百分比
services.exe	700	正在运行	SYSTEM	00	2,664 K	服务和控制服务应用
spoolsv.exe	1696	正在运行	SYSTEM	00	4,336 K	后台处理程序子系统应用
系统中断	-	正在运行	SYSTEM	00	K	延迟过程调用和中断服务例程

图 1-7 系统进程

表 1-1 实验记录

映像名称	用户名	CPU	内存使用
MsMpEng.exe	SYSTEM	01	64112 K
csrss.exe	SYSTEM	00	1144 K
csrss.exe	SYSTEM	01	1324 K
dllhost.exe	SYSTEM	00	2872 K
lsass.exe	SYSTEM	00	4280 K
System	SYSTEM	01	28 K
vmacthlp.exe	SYSTEM	00	916 K
VGAUTHService.exe	SYSTEM	00	2188 K
vmtoolsd.exe	SYSTEM	00	3340 K
TiWorker.exe	SYSTEM	02	258376 K
winlogon.exe	SYSTEM	00	1124 K
svchost.exe	SYSTEM	00	4572 K
svchost.exe	SYSTEM	00	6192 K
svchost.exe	SYSTEM	00	19984 K

svchost.exe	SYSTEM	00	6296 K
svchost.exe	SYSTEM	00	3604 K
svchost.exe	SYSTEM	00	1588 K
smss.exe	SYSTEM	00	268 K
TrustedInstaller.exe	SYSTEM	00	1264 K
wininit.exe	SYSTEM	00	732 K
WmiPrvSE.exe	SYSTEM	00	8144 K
系统空闲进程	SYSTEM	90	4 K
services.exe	SYSTEM	00	2664 K
spoolsv.exe	SYSTEM	00	4336 K
系统中断	SYSTEM	00	K

服务 (SERVICE) 进程有 9 个，如图 1-8，填入表 1-2 中。










	msdtc.exe	2748	正在运行	NETWORK SERVICE	00	1,920 K	Microsoft 分布式事务处理协调器服务
	svchost.exe	856	正在运行	NETWORK SERVICE	00	3,680 K	Windows 服务主进程
	svchost.exe	1136	正在运行	NETWORK SERVICE	00	6,168 K	Windows 服务主进程
	WmiPrvSE.exe	2588	正在运行	NETWORK SERVICE	00	8,832 K	WMI Provider Host
	svchost.exe	488	正在运行	LOCAL SERVICE	00	10,880 K	Windows 服务主进程
	svchost.exe	764	正在运行	LOCAL SERVICE	00	6,804 K	Windows 服务主进程
	svchost.exe	1040	正在运行	LOCAL SERVICE	00	6,380 K	Windows 服务主进程
	svchost.exe	1364	正在运行	LOCAL SERVICE	00	1,372 K	Windows 服务主进程
	svchost.exe	2544	正在运行	LOCAL SERVICE	00	1,236 K	Windows 服务主进程

图 1-8 服务进程

表 1-2 实验记录

映像名称	用户名	CPU	内存使用
msdtc.exe	NETWORK SERVICE	00	1920 K
svchost.exe	NETWORK SERVICE	00	3680 K
svchost.exe	NETWORK SERVICE	00	6168 K
WmiPrvSE.exe	NETWORK SERVICE	00	8832 K
svchost.exe	LOCAL SERVICE	00	10880 K
svchost.exe	LOCAL SERVICE	00	6804 K
svchost.exe	LOCAL SERVICE	00	6380 K
svchost.exe	LOCAL SERVICE	00	1372 K
svchost.exe	LOCAL SERVICE	00	1236 K

用户进程有 11 个，如图 1-9，并填入表 1-3 中。












	ApplicationFrameH...	1544	正在运行	Administrator	00	2,884 K	Application Frame Host
	ChsIME.exe	3396	正在运行	Administrator	00	4,296 K	Microsoft IME
	RuntimeBroker.exe	1920	正在运行	Administrator	00	4,120 K	Runtime Broker
	SearchUI.exe	3784	正在运行	Administrator	00	55,696 K	Search and Cortana application
	sihost.exe	1804	正在运行	Administrator	00	3,252 K	Shell Infrastructure Host
	Taskmgr.exe	140	正在运行	Administrator	03	12,580 K	Task Manager
	vmtoolsd.exe	1048	正在运行	Administrator	00	8,700 K	VMware Tools Core Service
	svchost.exe	392	正在运行	Administrator	00	2,428 K	Windows 服务主进程
	taskhostw.exe	3092	正在运行	Administrator	00	2,648 K	Windows 任务的主机进程
	explorer.exe	3388	正在运行	Administrator	00	15,116 K	Windows 资源管理器
	notepad.exe	3728	正在运行	Administrator	00	4,100 K	记事本

图 1-9 用户进程

表 1-3 实验记录			
映像名称	用户名	CPU	内存使用
ApplicationFrameHost.exe	Administrator	00	2884 K
ChsIME.exe	Administrator	00	4296 K
RuntimeBroker.exe	Administrator	00	4120 K
SearchUI.exe	Administrator	00	55696 K
sihost.exe	Administrator	00	3252 K
Taskmgr.exe	Administrator	03	12580 K
vmttoolsd.exe	Administrator	00	8700 K
svchost.exe	Administrator	00	2428 K
taskhostw.exe	Administrator	00	2648 K
explorer.exe	Administrator	00	15116 K
notepad.exe	Administrator	00	4100 K

步骤 2：单击要终止的进程，然后单击“结束进程/任务”按钮。

注意：终止进程时要小心。终止进程有可能导致不希望发生的结果，包括数据丢失和系统不稳定等。因为在被终止前，进程将没有机会保存其状态和数据。如果结束应用程序，您将丢失未保存的数据。如果结束系统服务，系统的某些部分可能无法正常工作。

终止进程，将结束它直接或间接创建的所有子进程。例如，如果终止了电子邮件程序(如 Outlook 98)的进程树，那么同时也终止了相关的进程，如 MAPI 后台处理程序 mapisp32.exe。

请将终止某进程后的操作结果与原记录数据对比，发生了什么：

在进程选项卡中点击相应进程并结束任务的三种方法：

法一：点击记事本，并结束任务，直接关闭记事本程序，未保存数据丢失；

法二：展开记事本，点击具体文件所打开的进程，弹窗提示是否更改保存；

法三：在详细信息中，选择 notepad，右键结束任务，弹出提示窗口，点击结束进程后，直接关闭记事本程序，未保存数据丢失；

使用以上三种方法结束任务均会从当前进程列表中删除该进程。

2. 显示其他进程属性

在“进程”选项卡页面中表头右键可以添加列。单击要增加显示为列标题的项目即可添加相应列。

为对进程列表进行排序，可在“进程”选项卡上单击要根据其进行排序的列标题。而为了要反转排序顺序，可再次单击列标题。

经过调整，“进程”选项卡现在显示的项目分别是：

名称、类型、PID、进程名称、命令行、CPU、内存

通过右键操作，可以在“任务管理器”中更改显示选项：

- 在“详细信息”选项卡上，可以选择列；
- 在“性能”选项卡上，左侧“子选项卡”右键菜单，可以选择摘要视图或显示小缩略图；在具体子选项卡窗口内可以更改 CPU 记录图，并显示内核时间。“显示内核时间”选项在“CPU 使用”和“CPU 使用记录”图表上添加红线。红线指示内核操作占用的 CPU 资源数量；
- 等等

3. 更改正在运行的程序的优先级

要查看正在运行的程序的优先级，可单击“进程”选项卡，单击“查看”菜单，单击“选择列”-“基本优先级”命令，然后单击“确定”按钮。为更改正在运行的程序的优先级，可在“进程”选项卡上右键单击您要更改的程序，指向“设置优先级”，然后单击所需的选项。更改进程的优先级可以使其运行更快或更慢（取决于是提升还是降低了优先级），但也可能对其他进程的性能有相反的影响。记录操作后所体会的结果：

优先级是在打开应用程序时系统为该进程分配的优先级属性，方便为其分配 CPU 和内存等系统资源。但由于本次实验开启进程不多，因此并无直观感觉。

实验 1.2 Windows Server 2016 进程的“一生”

Windows 所创建的每个进程都是以调用 CreateProcess() API 函数开始和以调用 ExitProcess() 或 TerminateProcess() API 函数终止。

1. 创建进程

本实验显示了创建子进程的基本框架。该程序只是再一次地启动自身，显示它的系统进程 ID 和它在进程列表中的位置。

步骤 1：登录进入 Windows Server 2016。

步骤 2：在“开始”菜单中单击“程序”-“Microsoft Visual Studio 2019”，进入 Visual Studio 2019 的窗口。

步骤 3：在工具栏单击“文件”按钮，在“文件”-“新建”-“项目”中创建一个 OSLab 解决方案，并创建一个位于该解决方案下的项目 Lab1_1，并在源文件中添加新建项，命名为 Lab1_1.cpp，将实验指导书中的代码拷贝至该文件中。

步骤 4：右键 Lab1_1，选择编译，VS 对 Lab1_1.cpp 进行编译。

步骤 5：编译完成后，在工具栏单击生成按钮，选择生成 Lab1_1（快捷键 Ctrl + B）命令，建立 Lab1_1.exe 可执行文件。

操作能否正常进行？如果不行，则可能的原因是什么？

由于程序代码是由 pdf 文件复制出来的，调整缩进与分号等规范后，进一步调整了安全性，如使用 `sprint_s` 等，并使用了命名空间 `std`，减少代码冗余，调整过后，程序代码可以调通。

步骤 6：在工具栏单击“调试” - “开始执行（不调试）”按钮，或者按 `Ctrl + F5` 键，或者单击“调试”菜单中的“开始调试”命令，执行 `Lab1_1.exe` 程序。

步骤 7：按 `Ctrl + S` 键可暂停程序的执行，按 `Ctrl + Pause (Break)` 键可终止程序的执行。

清单 1-1 展示的是一个简单的使用 `CreateProcess()` API 函数的例子。首先形成简单的命令行，提供当前的 EXE 文件的指定文件名和代表生成克隆进程的号码。大多数参数都可取缺省值，但是创建标志参数使用了：`CREATE_NEW_CONSOLE` 标志，指示新进程分配它自己的控制台，这使得运行示例程序时，在任务栏上产生许多活动标记。然后该克隆进程的创建方法关闭传递过来的句柄并返回 `main()` 函数。在关闭程序之前，每一进程的执行主线程暂停一下，以便让用户看到其中的至少一个窗口。

`CreateProcess()` 函数有 10 个核心参数，本实验程序中设置的各个参数的值是：

- a) `szFilename`
- b) `szCmdLine`
- c) `NULL`
- d) `NULL`
- e) `FALSE`
- f) `CREATE_NEW_CONSOLE`
- g) `NULL`
- h) `NULL`
- i) `&si`
- j) `&pi`

程序运行时屏幕显示的信息是：

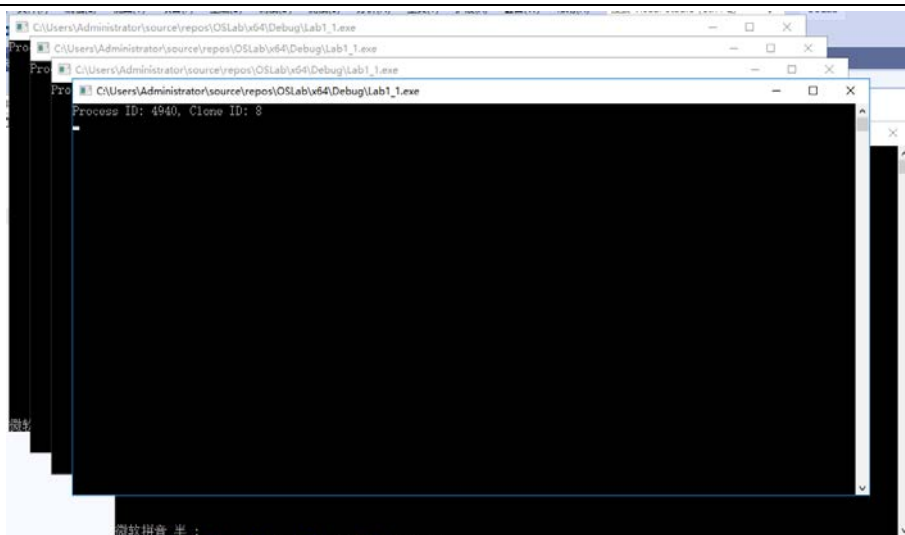


图 1-10 程序 Lab1_1 运行过程

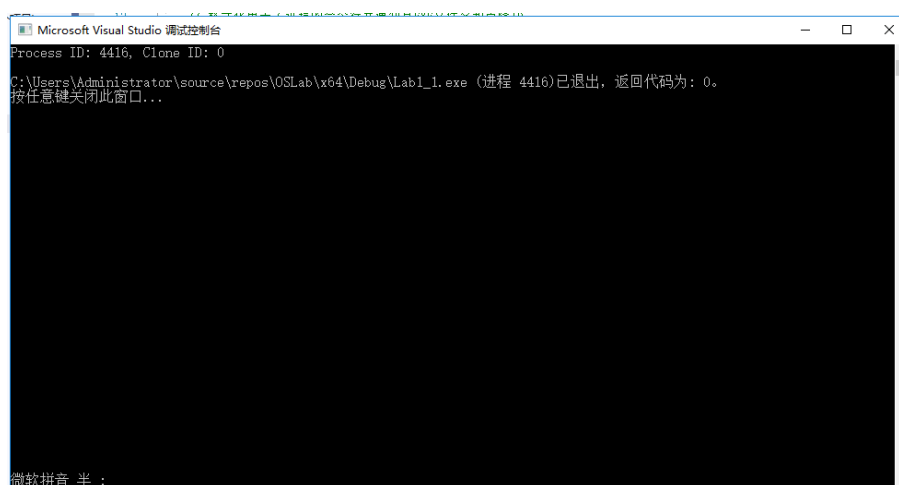


图 1-11 程序 Lab1_1 运行结果

提示: 部分程序在 Visual Studio 2019 环境完成编译、链接之后,还可以在 Windows Server 2016 的“命令提示符”状态下尝试执行该程序,看看与在可视化界面下运行的结果有没有不同?为什么?

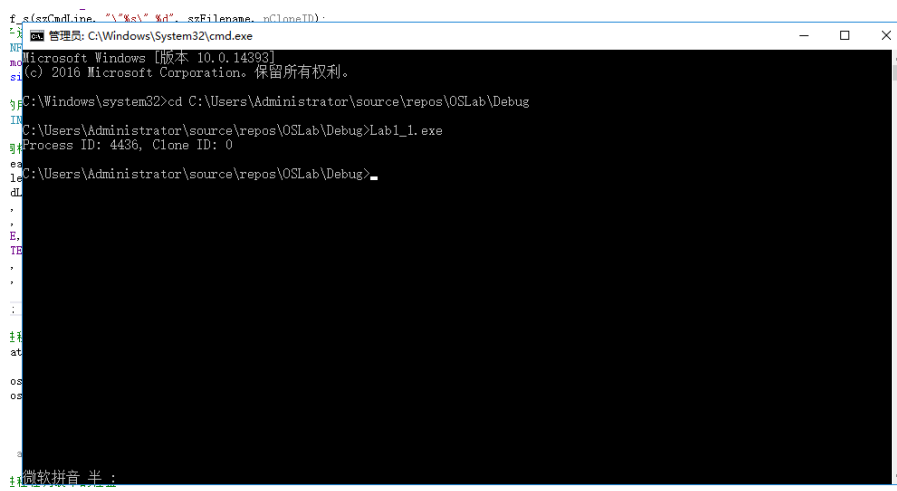


图 1-12 Lab1_1 控制台运行结果

除分配的 Process ID 不同外,其他运行过程类似。

原因：每次运行程序，系统为其分配的 ID 不同。

2. 正在运行的进程

本实验的程序中列出了用于进程信息查询的 API 函数 `GetProcessVersion()` 与 `GetVersionEx()` 的共同作用，可确定运行进程的操作系统版本号。

步骤 8：在 Visual Studio 2019 窗口的工具栏中单击“文件”按钮，在解决方案 OSLab 中新建一个项目，命名为 Lab1_2，并在源文件中添加新建项，命名为 Lab1_2.cpp，将实验指导书中的代码拷贝至该文件中。

步骤 9：单击“生成”菜单中的“编译”命令，系统对 Lab1_2.cpp 进行编译。

步骤 10：编译完成后，单击“生成”菜单中的“生成”命令，建立 Lab1_2.exe 可执行文件。

操作能否正常进行？如果不行，则可能的原因是什么？

由于程序代码是由 pdf 文件复制出来的，调整缩进与分号等规范后，使用了命名空间 std，减少代码冗余，调整过后，程序代码可以调通。

步骤 11：在解决方案资源管理器中右键当前项目，并设置为启动项目。在工具栏单击“调试” - “开始执行（不调试）”按钮，执行 Lab1_2.exe 程序。

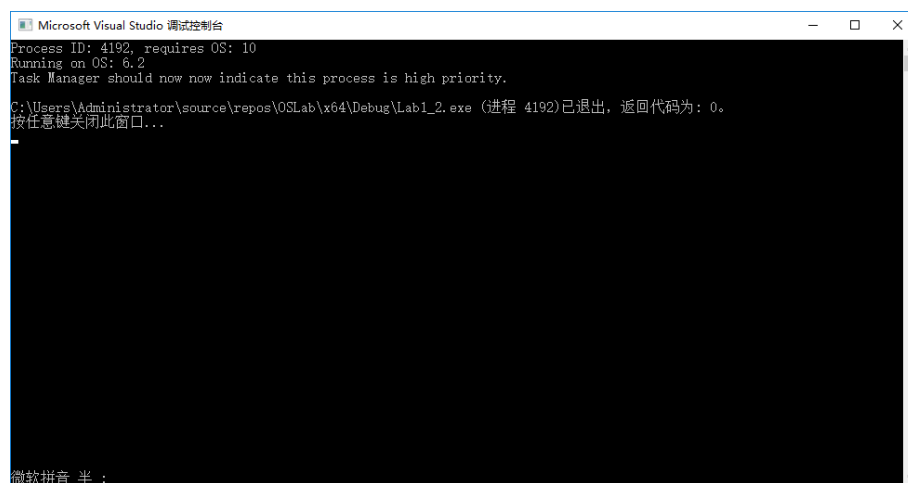


图 1-13 程序 Lab1_2 运行结果

运行结果：

当前 PID 信息：4192

当前操作系统版本：6.2

系统提示信息：

Task Manager should now now indicate this process is high priority.

Lab1_2 中的程序向读者表明了如何获得当前的 PID 和所需的进程版本信息。为了运

行这一程序，系统处理了所有的版本不兼容问题。

接着，程序演示了如何使用 `GetVersionEx()` API 函数来提取 `OSVERSIONINFOEX` 结构。这一数据块中包括了操作系统的版本信息。其中，“OS:6.2”表示当前运行的操作系统是：Windows Server 2016

Lab1_2 的最后一段程序利用了操作系统的版本信息，以确认运行的是 Windows Server 2016。（程序源代码对版本号进行了判断，OS 大于等于 5 即可）代码接着将当前进程的优先级提高到比正常级别高。

步骤 12: 单击 `Ctrl + Alt + Del` 键，进入“Windows 任务管理器”，在“详细信息”选项卡中右键单击“Lab1_2”任务，在快捷菜单中选择“转到进程”命令³。

在“Windows 任务管理器”的“进程”选项卡中，与“Lab1_2”任务对应的进程映像名称是(为什么?)：

VsDebugConsole.exe（原因：Lab1_2 的运行是在 VS 集成开发环境中的，依赖于 VS 调试控制台。）

右键单击该进程名，在快捷菜单中选择“设置优先级”命令，可以调整该进程的优先级，如设置为“高”后重新运行 Lab1_2.exe 程序，屏幕显示有变化吗？为什么？

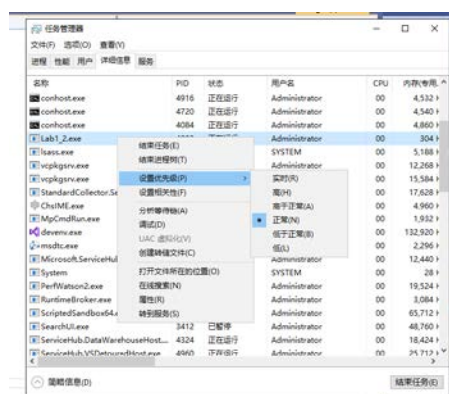


图 1-14 Lab1_2 设置优先级



图 1-15 程序 Lab1_2 运行结果

屏幕显示无变化。

原因：改变优先级操作之前未进行优先级判断，改变优先级时将其修改为高优先级（该进程原先也为高优先级），屏幕紧接着仍会输出报告语句。

除了改变进程的优先级以外，还可以对正在运行的进程执行几项其他的操作，只要获得其进程句柄即可。SetProcessAffinityMask() API 函数允许开发人员将线程映射到处理器上；SetProcessPriorityBoost() API 可关闭前台应用程序优先级的提升；而 SetProcessWorkingSet() API 可调节进程可用的非页面 RAM 的容量；还有一个只对当前进程可用的 API 函数，即 SetProcessShutdownParameters()，可告诉系统如何终止该进程。

3. 终止进程

在清单 1-3 列出的程序中，先创建一个子进程，然后指令它发出“自杀弹”互斥体去终止自身的运行。

步骤 13：在 Visual Studio 2019 窗口的工具栏中单击“文件”按钮，在解决方案 OSLab 中新建一个项目，命名为 Lab1_3，并在源文件中添加新建项，命名为 Lab1_3.cpp，将实验指导书中的代码拷贝至该文件中。

清单 1-3 中的程序说明了一个进程从“生”到“死”的整个一生。第一次执行时，它创建一个子进程，其行为如同“父亲”。在创建子进程之前，先创建一个互斥的内核对象，其行为对于子进程来说，如同一个“自杀弹”。当创建子进程时，就打开了互斥体并在其他线程中进行别的处理工作，同时等待着父进程使用 ReleaseMutex() API 发出“死亡”信号。然后用 Sleep() API 调用来模拟父进程处理其他工作，等完成时，指令子进程终止。

当调用 ExitProcess() 时要小心，进程中的所有线程都被立刻通知停止。在设计应用程序时，必须让主线程在正常的 C++ 运行期关闭（这是由编译器提供的缺省行为）之后来调用这一函数。当它转向受信状态时，通常可创建一个每个活动线程都可等待和停止的终止事件。

在正常的终止操作中，进程的每个工作线程都要终止，由主线程调用 ExitProcess()。接着，管理层对进程增加的所有对象释放引用，并将用 GetExitCodeProcess() 建立的退出代码从 STILL_ACTIVE 改变为在 ExitProcess() 调用中返回的值。最后，主线程对象也如同进程对象一样转变为受信状态。

等到所有打开的句柄都关闭之后，管理层的对象管理器才销毁进程对象本身。还没有一种函数可取得终止后的进程对象为其参数，从而使其“复活”。当进程对象引用一

个终止了的对象时，有好几个 API 函数仍然是有用的。进程可使用退出代码将终止方式通知给调用 `GetExitCodeProcess()` 的其他进程。同时，`GetProcessTimes()` API 函数可向主调者显示进程的终止时间。

步骤 14: 单击“生成”菜单中的“编译”命令，系统对 `Lab1_3.cpp` 进行编译。

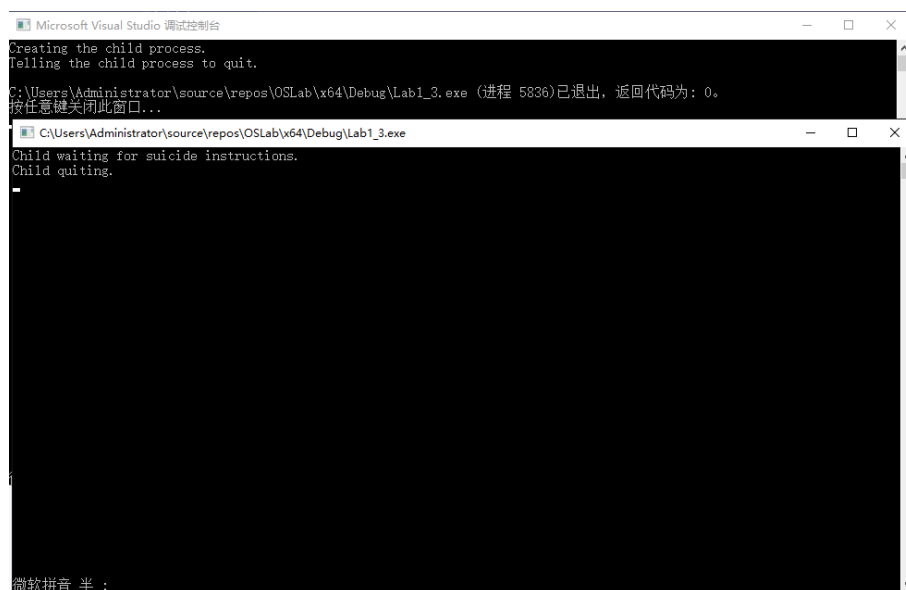
步骤 15: 编译完成后，单击“生成”菜单中的“生成”命令，建立 `Lab1_3.exe` 可执行文件。

操作能否正常进行？如果不行，则可能的原因是什么？

由于程序代码是由 pdf 文件复制出来的，调整缩进与分号等规范后，使用了命名空间 `std`，减少代码冗余，调整过后，程序代码可以调通。

步骤 16: 在解决方案资源管理器中右键当前项目，并设置为启动项目。在工具栏单击“调试” - “开始执行（不调试）”按钮，执行 `Lab1_3.exe` 程序。

运行结果（当前控制台窗口，其他控制台信息见图片）：



```
Microsoft Visual Studio 调试控制台
Creating the child process.
Telling the child process to quit.
C:\Users\Administrator\source\repos\OSLab\x64\Debug\Lab1_3.exe (进程 5836)已退出，返回代码为: 0。
按任意键关闭此窗口...

C:\Users\Administrator\source\repos\OSLab\x64\Debug\Lab1_3.exe
Child waiting for suicide instructions.
Child quitting.
```

图 1-16 程序 `Lab1_3` 运行过程

1) Creating the child process.

表示： 创建子进程，并生成新的子进程控制台窗口

2) Telling the child process to quit.

表示： 子进程等待一会，收到消息后结束自身进程

步骤 17: 在熟悉清单 1-3 源代码的基础上，利用本实验介绍的 API 函数来尝试改进本程序（例如使用 `GetProcessTimes()` API 函数）并运行。

可以使用 `GetProcessTimes()` 来输出进程的具体存在时间（创建时间、死亡时间、内核时间、用户时间等）。

五、实验结果与分析

见实验步骤与实验过程以及注释。

六、实验心得体会

通过本次实验，更加透彻地理解了 Windows 进程控制的应用，如使用任务管理器查看进程、结束进程等等；学会了如何使用 Windows API 函数创建与结束进程。同时，较为深入地理解了 FILETIME 的时间结构与使用。

附录 程序清单

清单 1-1

```
1.  // procreate 项目
2.  #include <windows.h>
3.  #include <iostream>
4.  #include <stdio.h>
5.  using namespace std;
6.
7.  // 创建传递过来的进程的克隆过程并赋与其 ID 值
8.  void StartClone(int nCloneID) {
9.      // 提取用于当前可执行文件的文件名
10.     TCHAR szFilename[MAX_PATH];
11.     ::GetModuleFileName(NULL, szFilename, MAX_PATH);
12.     // 格式化用于子进程的命令行并通知其 EXE 文件名和克隆 ID
13.     TCHAR szCmdLine[MAX_PATH];
14.     ::sprintf_s(szCmdLine, "\\\"%s\" %d", szFilename, nCloneID);
15.     // 用于子进程的 STARTUPINFO 结构
16.     STARTUPINFO si;
17.     ::ZeroMemory(reinterpret_cast<void*>(&si), sizeof(si));
18.     si.cb = sizeof(si); // 必须是本结构的大小
19.
20.     // 返回的用于子进程的进程信息
21.     PROCESS_INFORMATION pi;
22.
23.     // 利用同样的可执行文件和命令行创建进程，并赋予其子进程的性质
24.     BOOL bCreateOK = ::CreateProcess(
25.         szFilename,    // 产生这个 EXE 的应用程序的名称
26.         szCmdLine,     // 告诉其行为像一个子进程的标志
27.         NULL,          // 缺省的进程安全性
28.         NULL,          // 缺省的线程安全性
29.         FALSE,         // 不继承句柄
30.         CREATE_NEW_CONSOLE, // 使用新的控制台
31.         NULL,          // 新的环境
```

```

32.     NULL,        // 当前目录
33.     &si,         // 启动信息
34.     &pi);        // 返回的进程信息
35.
36. // 对子进程释放引用
37. if (bCreateOK)
38. {
39.     ::CloseHandle(pi.hProcess);
40.     ::CloseHandle(pi.hThread);
41. }
42. }
43.
44. int main(int argc, char* argv[]) {
45.
46.     // 确定进程在列表中的位置
47.     int nClone(0);
48.     if (argc > 1)
49.     {
50.         // 从第二个参数中提取克隆 ID
51.         ::sscanf_s(argv[1], "%d", &nClone);
52.     }
53.
54.     // 显示进程位置
55.     cout << "Process ID: " << ::GetCurrentProcessId()
56.          << ", Clone ID: " << nClone
57.          << endl;
58.
59.     // 检查是否有创建子进程的需要
60.     const int C_nCloneMax = 25;
61.     if (nClone < C_nCloneMax)
62.     {
63.         // 发送新进程的命令行和克隆号
64.         StartClone(++nClone);
65.     }
66.     // 在终止之前暂停一下 (1/2 秒)
67.     ::Sleep(500);
68.
69.     return 0;
70. }

```

清单 1-2

```

1. // version 项目
2. #include <windows.h>
3. #include <iostream>
4. using namespace std;

```

```

5.  #pragma warning(disable: 4996)
6.
7.  // 利用进程和操作系统的版本信息的简单示例
8.  int main() {
9.      // 提取这个进程的 ID 号
10.     DWORD dwIdThis = ::GetCurrentProcessId();
11.
12.     // 获得这一进程和报告所需的版本，也可以发送 0 以便指明这一进程
13.     DWORD dwVerReq = ::GetProcessVersion(dwIdThis);
14.     WORD wMajorReq = (WORD)(dwVerReq > 16);
15.     WORD wMinorReq = (WORD)(dwVerReq & 0xffff);
16.     cout << "Process ID: " << dwIdThis << ", requires OS: "
17.          << wMajorReq << wMinorReq << endl;
18.
19.     // 设置版本信息的数据结构，以便保存操作系统的版本信息
20.     OSVERSIONINFOEX osvix;
21.     ::ZeroMemory(&osvix, sizeof(osvix));
22.     osvix.dwOSVersionInfoSize = sizeof(osvix);
23.     // 提取版本信息和报告
24.     ::GetVersionEx(reinterpret_cast<LPOSVERSIONINFO>(&osvix));
25.     cout << "Running on OS: " << osvix.dwMajorVersion << "."
26.          << osvix.dwMinorVersion << endl;
27.
28.     // 如果是 NTS (Windows Server 2003) 系统，则提高其优先权
29.     if (osvix.dwPlatformId == VER_PLATFORM_WIN32_NT &&
30.         osvix.dwMajorVersion >= 5)
31.     {
32.         // 改变优先级
33.         ::SetPriorityClass(
34.             ::GetCurrentProcess(), // 利用这一进程
35.             HIGH_PRIORITY_CLASS // 改变为 high
36.         );
37.
38.         // 报告给用户
39.         cout << "Task Manager should now now indicate this "
40.              << "process is high priority. " << endl;
41.     }
42.
43.     return 0;
44. }

```

清单 1-3

```

1.  // procterm 项目
2.  #include <windows.h>
3.  #include <iostream>

```



```

4.  #include <stdio.h>
5.  static LPCTSTR g_szMutexName = "w2kdg.ProcTerm.mutex.Suicide";
6.  using namespace std;
7.
8.  // 创建当前进程的克隆进程的简单方法
9.  void StartClone() {
10.     // 提取当前可执行文件的文件名
11.     TCHAR szFilename[MAX_PATH];
12.     ::GetModuleFileName(NULL, szFilename, MAX_PATH);
13.
14.     // 格式化用于子进程的命令行, 指明它是一个 EXE 文件和子进程
15.     TCHAR szCmdLine[MAX_PATH];
16.     ::sprintf_s(szCmdLine, "\\\"%s\\\" child", szFilename);
17.
18.     // 子进程的启动信息结构
19.     STARTUPINFO si;
20.     ::ZeroMemory(&si, sizeof(si));
21.     si.cb = sizeof(si); // 应当是此结构的大小
22.
23.     // 返回的用于子进程的进程信息
24.     PROCESS_INFORMATION pi;
25.
26.     // 用同样的可执行文件名和命令行创建进程, 并指明它是一个子进程
27.     BOOL bCreateOK = ::CreateProcess(
28.         szFilename,           // 产生的应用程序名称 (本 EXE 文件)
29.         szCmdLine,           // 告诉我们这是一个子进程的标志
30.         NULL,                 // 用于进程的缺省的安全性
31.         NULL,                 // 用于线程的缺省安全性
32.         FALSE,               // 不继承句柄
33.         CREATE_NEW_CONSOLE, // 创建新窗口, 使输出更直观
34.         NULL,                 // 新环境
35.         NULL,                 // 当前目录
36.         &si,                  // 启动信息结构
37.         &pi);                 // 返回的进程信息
38.
39.     // 释放指向子进程的引用
40.     if (bCreateOK)
41.     {
42.         ::CloseHandle(pi.hProcess);
43.         ::CloseHandle(pi.hThread);
44.     }
45. }
46.
47. void Parent()
48. {
49.     // 创建"自杀"互斥程序体

```

```
50.     HANDLE hMutexSuicide = ::CreateMutex(
51.         NULL, // 缺省的安全性
52.         TRUE, // 最初拥有的
53.         g_szMutexName); // 为其命名
54.
55.     if (hMutexSuicide != NULL)
56.     {
57.         // 创建子进程
58.         cout << "Creating the child process." << endl;
59.         ::StartClone();
60.         // 暂停
61.         ::Sleep(5000);
62.         // 指令子进程"杀"掉自身
63.         cout << "Telling the child process to quit. " << endl;
64.         ::ReleaseMutex(hMutexSuicide);
65.         // 消除句柄
66.         ::CloseHandle(hMutexSuicide);
67.     }
68. }
69.
70. void Child()
71. {
72.     // 打开"自杀"互斥体
73.     HANDLE hMutexSuicide = ::OpenMutex(
74.         SYNCHRONIZE, // 打开用于同步
75.         FALSE, // 不需要向下传递
76.         g_szMutexName); // 名称
77.     if (hMutexSuicide != NULL)
78.     {
79.         // 报告正在等待指令
80.         cout << "Child waiting for suicide instructions. " << endl;
81.         ::WaitForSingleObject(hMutexSuicide, INFINITE);
82.         // 准备好终止, 清除句柄
83.         cout << "Child quitting. " << endl;
84.         ::CloseHandle(hMutexSuicide);
85.         // 等待显示信息
86.         ::Sleep(5000);
87.     }
88. }
89.
90. int main(int argc, char* argv[])
91. {
92.     // 决定其行为是父进程还是子进程
93.     if (argc > 1 && ::strcmp(argv[1], "child") == 0) {
94.         Child();
95.     }
```

```
96.     else {  
97.         Parent();  
98.     }  
99.  
100.    return 0;  
101.}
```

-
- 1 在 Vmware 虚拟机中，为避免按键冲突，使用组合键 **Ctrl + Alt + Insert** 代替组合键 **Ctrl + Alt + Delete**。
 - 2 最多开启一个任务管理器，无法多开。
 - 3 在 Windows Server 2016 中未找到该命令，实际操作通过关闭该进程树进行判断的进程。