操作系统实验指导书

软件学院

2020 年 2 月

目录

《操作系统》实验教学大纲

课程编号：A2130330

课程名称：操作系统

实验学时：8

一、本实验课的性质、任务与目的

操作系统作为软件工程专业的一门专业基础课，是软件工程专业的核心课程之一，学好与否直接关系到学生是否能更好理解软件专业的专业理论。通过本实验课程的学习，使学生理解与掌握操作系统设计所遵循的基本原理，基本方法，建立多道程序设计环境下的并行程序设计的思维方式。此外，操作系统用到的各种计算机方法也是学生加强软件认识、编程锻炼的好机会，对日后从事系统开发方面的工作有直接的借鉴作用。

本实验课程在操作系统原理课程教学中占有重要地位，目的是让学生及时掌握和巩固所学的基本原理和基础理论，加深理解。提高学生自适应能力，为将来设计和开发系统级程序打下良好的基础。

二、本实验课所依据的课程基本理论

计算机操作系统及其介绍的相关系统设计算法。

三、实验类型与要求

序号	实验内容	内容提要	实验时数	实验类型
一	进程控制描述与控制	操作系统界面、进程管理	2	验证
二	并发与调度	进程并发、进程状态转换	2	验证
三	存储管理	内存空间分配及虚拟存储器	2	验证
四	设备管理	磁盘的物理组织	2	验证

四、考核方式与评分办法

综合每次实验课堂表现以及实验完成情况（包含程序的质量、完成及时性及实验报告）。

五、实验报告要求

实验题目、问题描述、算法说明、算法框图、程序清单及运行结果、个人体会。

六、实验考核方式与基本要求

1） 按要求设计相应的模拟系统并上机调试运行

2） 写出详细的实验报告，实验报告要求如下：

（1）实验题目。

（2）实验目的及内容。

（3）步骤及流程图。

（4）源程序（含注释）。

（5）程序运行时的初值和运行结果。

（6）实验心得及体会。

基本要求：采取课内上机和业余上机相结合的方式进行，在规定时间内上交实验（设计）结果并上机演示说明。

实验一 进程控制描述与控制

[1] Windows“任务管理器”的进程管理

背景知识

Windows Server 2003 的任务管理器提供了用户计算机上正在运行的程序和进程的相关信息，也显示了最常用的度量进程性能的单位。使用任务管理器，可以打开监视计算机性能的关键指示器，快速查看正在运行的程序的状态，或者终止已停止响应的程序。也可以使用多个参数评估正在运行的进程的活动，以及查看 CPU 和内存使用情况的图形和数据。其中：

1) “应用程序”选项卡显示正在运行程序的状态，用户能够结束、切换或者启动程序。

2) “进程”选项卡显示正在运行的进程信息。例如，可以显示关于 CPU 和内存使用情况、页面错误、句柄计数以及许多其他参数的信息。

3) “性能”选项卡显示计算机动态性能，包括 CPU 和内存使用情况的图表，正在运行的句柄、线程和进程的总数，物理、核心和认可的内存总数 (KB) 等。

实验目的

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。

工具/准备工作

在开始本实验之前，请回顾教科书的相关内容。

需要准备一台运行 Windows 系列操作系统的计算机。

实验内容与步骤

1. 使用任务管理器终止进程
2. 显示其他进程计数器
3. 更改正在运行的程序的优先级

启动并进入 Windows 环境，单击 Ctrl + Alt + Del 键，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。

在本次实验中，你使用的操作系统版本是：

_____

当前机器中由你打开，正在运行的应用程序有：

1) _____

2) _____

3) _____

4) _____

5) _____

Windows“任务管理器”的窗口由_____个选项卡组成，分别是：

1) _____

2) _____

3) _____

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) _____

2) _____

3) _____

4) _____

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_____个进程。请试着区分一下，其中：系统 (SYSTEM) 进程有_____个，填入表 3-1 中。

表 3-1 实验记录

映像名称	用户名	CPU	内存使用

服务 (SERVICE) 进程有_____个，填入表 3-2 中。

表 3-2 实验记录

映像名称	用户名	CPU	内存使用

用户进程有_____个，填入表 3-3 中。

表 3-3 实验记录

映像名称	用户名	CPU	内存使用

步骤 2：单击要终止的进程，然后单击“结束进程”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：

_____
_____
_____
_____

2. 显示其他进程属性

在“进程”选项卡上单击“查看”菜单，然后单击“选择列”命令。单击要增加显示为列标题的项目，然后单击“确定”。

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：

_____
_____
_____

通过对“查看”菜单的选择操作，可以在“任务管理器”中更改显示选项：

- 在“应用程序”选项卡上，可以按详细信息、大图标或小图标查看。
- 在“性能”选项卡上，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。

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：

_____
_____
_____
_____
_____

[2]Windows　进程的“一生”

背景知识

1. 创建进程
2. 正在运行的进程
3. 终止进程

Windows 所创建的每个进程都从调用 CreateProcess() API 函数开始，该函数的任务是在对象管理器子系统内初始化进程对象。每一进程都以调用 ExitProcess() 或 TerminateProcess() API 函数终止。通常应用程序的框架负责调用 ExitProcess() 函数。对于 C++ 运行库来说，这一调用发生在应用程序的 main() 函数返回之后。

1. 创建进程

CreateProcess() 调用的核心参数是可执行文件运行时的文件名及其命令行。表 3-4 详细地列出了每个参数的类型和名称。

表 3-4 CreateProcess() 函数的参数

参数名称	使用目的
LPCTSTR lpApplivationName	全部或部分地指明包括可执行代码的 EXE 文件的文件名
LPCTSTR lpCommandLine	向可执行文件发送的参数
LPSECURIITY_ATTRIBUTES lpProcessAttributes	返回进程句柄的安全属性。主要指明这一句柄是否应该由其他子进程所继承
LPSECURIITY_ATTRIBUTES lpThreadAttributes	返回进程的主线程的句柄的安全属性
BOOL bInheritHandle	一种标志，告诉系统允许新进程继承创建者进程的句柄
DWORD dwCreationFlage	特殊的创建标志 (如 CREATE_SUSPENDED) 的位标记
LPVOID lpEnvironment	向新进程发送的一套环境变量；如为 null 值则发送调用者环境
LPCTSTR lpCurrentDirectory	新进程的启动目录
STARTUPINFO lpStartupInfo	STARTUPINFO 结构，包括新进程的输入和输出配置的详情
LPPROCESS_INFORMATION lpProcessInformation	调用的结果块；发送新应用程序的进程和主线程的句柄和 ID

可以指定第一个参数，即应用程序的名称，其中包括相对于当前进程的当前目录的全路径或者利用搜索方法找到的路径；lpCommandLine 参数允许调用者向新应用程序发送数据；接下来的三个参数与进程和它的主线程以及返回的指向该对象的句柄的安全性有关。

然后是标志参数，用以在 dwCreationFlags 参数中指明系统应该给予新进程什么行为。经常使用的标志是 CREATE_SUSPNDED，告诉主线程立刻暂停。当准备好时，应该使用 ResumeThread() API 来启动进程。另一个常用的标志是 CREATE_NEW_CONSOLE，告诉新进程启动自己的控制台窗口，而不是利用父窗口。这一参数还允许设置进程的优先级，用以向系统指明，相对于系统中所有其他的活动进程来说，给此进程多少 CPU 时间。

接着是 CreateProcess() 函数调用所需要的三个通常使用缺省值的参数。第一个参数是 lpEnvironment 参数，指明为新进程提供的环境；第二个参数是 lpCurrentDirectory，可用于向主创进程发送与缺省目录不同的新进程使用的特殊的当前目录；第三个参数是 STARTUPINFO 数据结构所必需的，用于在必要时指明新应用程序的主窗口的外观。

CreateProcess() 的最后一个参数是用于新进程对象及其主线程的句柄和 ID 的返回值缓冲区。以 PROCESS_INFORMATION 结构中返回的句柄调用 CloseHandle() API 函数是重要的，因为如果不将这些句柄关闭的话，有可能危及主创进程终止之前的任何未释放的资源。

2. 正在运行的进程

如果一个进程拥有至少一个执行线程，则为正在系统中运行的进程。通常，这种进程使用主线程来指示它的存在。当主线程结束时，调用 ExitProcess() API 函数，通知系统终止它所拥有的所有正在运行、准备运行或正在挂起的其他线程。当进程正在运行时，可以查看它的许多特性，其中少数特性也允许加以修改。

首先可查看的进程特性是系统进程标识符 (PID) ，可利用 GetCurrentProcessId() API 函数来查看，与 GetCurrentProcess() 相似，对该函数的调用不能失败，但返回的 PID 在整个系统中都可使用。其他的可显示当前进程信息的 API 函数还有 GetStartupInfo() 和 GetProcessShutdownParameters() ，可给出进程存活期内的配置详情。

通常，一个进程需要它的运行期环境的信息。例如 API 函数 GetModuleFileName() 和 GetCommandLine() ，可以给出用在 CreateProcess() 中的参数以启动应用程序。在创建应用程序时可使用的另一个 API 函数是 IsDebuggerPresent() 。

可利用 API 函数 GetGuiResources() 来查看进程的 GUI 资源。此函数既可返回指定进程中的打开的 GUI 对象的数目，也可返回指定进程中打开的 USER 对象的数目。进程的其他性能信息可通过 GetProcessIoCounters()、GetProcessPriorityBoost() 、GetProcessTimes() 和 GetProcessWorkingSetSize() API 得到。以上这几个 API 函数都只需要具有 PROCESS_QUERY_INFORMATION 访问权限的指向所感兴趣进程的句柄。

另一个可用于进程信息查询的 API 函数是 GetProcessVersion() 。此函数只需感兴趣进程的 PID (进程标识号) 。本实验程序清单 3-6 中列出了这一 API 函数与 GetVersionEx() 的共同作用，可确定运行进程的系统的版本号。

3. 终止进程

所有进程都是以调用 ExitProcess() 或者 TerminateProcess() 函数结束的。但最好使用前者而不要使用后者，因为进程是在完成了它的所有的关闭“职责”之后以正常的终止方式来调用前者的。而外部进程通常调用后者即突然终止进程的进行，由于关闭时的途径不太正常，有可能引起错误的行为。

TerminateProcess() API 函数只要打开带有 PROCESS_TERMINATE 访问权的进程对象，就可以终止进程，并向系统返回指定的代码。这是一种“野蛮”的终止进程的方式，但是有时却是需要的。

如果开发人员确实有机会来设计“谋杀”(终止别的进程的进程) 和“受害”进程 (被终止的进程) 时，应该创建一个进程间通讯的内核对象——如一个互斥程序——这样一来，“受害”进程只在等待或周期性地测试它是否应该终止。

实验目的

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 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。

工具/准备工作

在开始本实验之前，请回顾教科书的相关内容。

需要做以下准备：

1) 一台运行 Windows　操作系统的计算机。

2) 计算机中需安装 Visual C++ 6.0 专业版或企业版或 VS 其他版本。

实验内容与步骤

1. 创建进程
2. 正在运行的进程
3. 终止进程

请回答：

Windows 所创建的每个进程都是以调用_____ API 函数开始和以调用_____ 或 _____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 2003 。

步骤 2：在“开始”菜单中单击“程序”-“Microsoft Visual Studio 6.0”–“Microsoft Visual C++ 6.0”命令，进入 Visual C++窗口。

步骤 3：在工具栏单击“打开”按钮，在“打开”对话框中找到并打开实验源程序 3-5.cpp。

清单 3-5 创建子进程

```
// proccreate 项目
# include <windows.h>
# include <iostream>
# include <stdio.h>
// 创建传递过来的进程的克隆过程并赋与其 ID 值
void StartClone(int nCloneID)
{
  // 提取用于当前可执行文件的文件名
  TCHAR szFilename[MAX_PATH] ;
  :: GetModuleFileName(NULL, szFilename, MAX_PATH) ;

  // 格式化用于子进程的命令行并通知其 EXE 文件名和克隆 ID
  TCHAR szCmdLine[MAX_PATH] ;
  :: sprintf(szCmdLine, "\"%s\" %d", szFilename, nCloneID) ;

  // 用于子进程的 STARTUPINFO 结构
  STARTUPINFO si;
  :: ZeroMemory(reinterpret_cast <void*> (&si) , sizeof(si) ) ;
  si.cb = sizeof(si) ;                    // 必须是本结构的大小

  // 返回的用于子进程的进程信息
  PROCESS_INFORMATION pi;

  // 利用同样的可执行文件和命令行创建进程，并赋于其子进程的性质
  BOOL bCreateOK = :: CreateProcess(
      szFilename,                         // 产生这个 EXE 的应用程序的名称
      szCmdLine,                          // 告诉其行为像一个子进程的标志
      NULL,                               // 缺省的进程安全性
      NULL,                               // 缺省的线程安全性
      FALSE,                              // 不继承句柄
```

```
            CREATE_NEW_CONSOLE,      // 使用新的控制台
            NULL,                    // 新的环境
            NULL,                    // 当前目录
            &si,                     // 启动信息
            &pi) ;                   // 返回的进程信息

        // 对子进程释放引用
        if (bCreateOK)
        {
            :: CloseHandle(pi.hProcess) ;
            :: CloseHandle(pi.hThread) ;
        }
    }

    int main(int argc, char* argv[] )
    {
        // 确定进程在列表中的位置
        int nClone(0) ;
        if (argc > 1)
        {
            // 从第二个参数中提取克隆 ID
            :: sscanf(argv[1] , "%d" , &nClone) ;
        }

        // 显示进程位置
        std :: cout << "Process ID: " << :: GetCurrentProcessId()
                    << ", Clone ID: " << nClone
                    << std :: endl;

        // 检查是否有创建子进程的需要
        const int C_nCloneMax = 25;
        if (nClone < C_nCloneMax)
        {
            // 发送新进程的命令行和克隆号
            StartClone(++nClone) ;
        }
        // 在终止之前暂停一下 (l/2 秒)
        :: Sleep(500) ;

        return 0;
    }
```

步骤 4：单击“Build”菜单中的“Compile 3-5.cpp”命令，系统显示：

This build command requires an active project workspace. Would you like to create a default project workspace ?

(build 命令需要一个活动的项目工作空间。你是否希望建立一个缺省的项目工作空间？)

单击“是”按钮确认。系统对 3-5.cpp 进行编译。

步骤 5：编译完成后，单击“Build”菜单中的“Build 3-5.exe”命令，建立 3-5.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？

_____

步骤 6：在工具栏单击“Execute Program”(执行程序) 按钮，或者按 Ctrl + F5 键，或者单击“Build”菜单中的“Execute 3-5.exe”命令，执行 3-5.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 3-5 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：

_____

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_____个核心参数？本实验程序中设置的各个参数的值是：

a. _____;
b. _____;
c. _____;
d. _____;
e. _____;
f. _____;
g. _____;
h. _____。

程序运行时屏幕显示的信息是：

_____
_____

> **提示：**部分程序在 Visual C++环境完成编译、链接之后，还可以在 Windows Server 2003 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？

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的版本号。

步骤 8：在 Visual C++ 窗口的工具栏中单击“打开”按钮，在“打开”对话框中找到并打开实验源程序 3-6.cpp。

清单 3-6 使用进程和操作系统的版本信息

```
// version 项目
# include <windows.h>
# include <iostream>

// 利用进程和操作系统的版本信息的简单示例
void main()
{
    // 提取这个进程的 ID 号
    DWORD dwIdThis = :: GetCurrentProcessId() ;

    // 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
    DWORD dwVerReq = :: GetProcessVersion(dwIdThis) ;
    WORD wMajorReq = (WORD) (dwVerReq > 16) ;
    WORD wMinorReq = (WORD) (dwVerReq & 0xffff) ;
    std :: cout << “Process ID: “ << dwIdThis
        << “, requires OS: “ << wMajorReq << wMinorReq << std :: endl ;

    // 设置版本信息的数据结构，以便保存操作系统的版本信息
    OSVERSIONINFOEX osvix;
    :: ZeroMemory(&osvix, sizeof(osvix) ) ;
```

```
        osvix.dwOSVersionInfoSize = sizeof(osvix) ;

        // 提取版本信息和报告
        :: GetVersionEx(reinterpret_cast < LPOSVERSIONINFO > (&osvix) ) ;
        std :: cout << "Running on OS: " << osvix.dwMajorVersion << "."
            << osvix.dwMinorVersion << std :: endl;

        // 如果是 NTS (Windows  Server 2003) 系统，则提高其优先权
        if (osvix.dwPlatformId == VER_PLATFORM_WIN32_NT &&
            osvix.dwMajorVersion >= 5)
        {
            // 改变优先级
            :: SetPriorityClass(
                :: GetCurrentProcess( ) ,                    // 利用这一进程
                HIGH_PRIORITY_CLASS) ;              // 改变为 high

            // 报告给用户
            std :: cout << "Task Manager should now now indicate this"
                "process is high priority." << std :: endl;
        }
    }
```

步骤 9：单击“Build”菜单中的“Compile 3-6.cpp”命令，再单击“是”按钮确认。系统对 3-6.cpp 进行编译。

步骤 10：编译完成后，单击“Build”菜单中的“Build 3-6.exe”命令，建立 3-6.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？

_____

_____

步骤 11：在工具栏单击“Execute Program” (执行程序) 按钮，执行 3-6.exe 程序。

运行结果：

当前 PID 信息：_____

当前操作系统版本：_____

系统提示信息：_____

_____

清单 3-6 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 : 5.0”表示当前运行的操作系统是：

_____

清单 3-6 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 2003。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“应用程序”选项卡中右键单击“3-6”任务，在快捷菜单中选择“转到进程”命令。

在“Windows 任务管理器”的“进程”选项卡中，与“3-6”任务对应的进程映像名称是(为什么？) ：

_____

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 3-6.exe 程序，屏幕显示有变化吗？为什么？

_____

_____

_____

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 3-7 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 C++ 窗口的工具栏中单击“打开”按钮，在“打开”对话框中找到并打开实验源程序 3-7.cpp。

清单 3-7　指令其子进程来“杀掉”自己的父进程

```
// procterm 项目
# include <windows.h>
# include <iostream>
# include <stdio.h>
static LPCTSTR g_szMutexName = “w2kdg.ProcTerm.mutex.Suicide” ;

// 创建当前进程的克隆进程的简单方法
void StartClone()
{
    // 提取当前可执行文件的文件名
    TCHAR szFilename [MAX_PATH] ;
    :: GetModuleFileName(NULL, szFilename, MAX_PATH) ;

    // 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
    TCHAR szCmdLine[MAX_PATH] ;
    :: sprintf(szCmdLine, “\” %s\“ child” , szFilename) ;

    // 子进程的启动信息结构
    STARTUPINFO si;
    :: ZeroMemory(reinterpret_cast < void* > (&si) , sizeof(si) ) ;
    si.cb = sizeof(si) ;                    // 应当是此结构的大小

    // 返回的用于子进程的进程信息
    PROCESS_INFORMATION pi;

    // 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
    BOOL bCreateOK = :: CreateProcess(
        szFilename,                         // 产生的应用程序名称 (本 EXE 文件)
        szCmdLine,                          // 告诉我们这是一个子进程的标志
        NULL,                               // 用于进程的缺省的安全性
        NULL,                               // 用于线程的缺省安全性
        FALSE,                                  // 不继承句柄
        CREATE_NEW_CONSOLE,                 // 创建新窗口，使输出更直观
        NULL,                               // 新环境
        NULL,                               // 当前目录
        &si,                                    // 启动信息结构
        &pi ) ;                             // 返回的进程信息

    // 释放指向子进程的引用
    if (bCreateOK)
    {
        :: CloseHandle(pi.hProcess) ;
```

```
            :: CloseHandle(pi.hThread) ;
        }
}
void Parent()
{
        // 创建“自杀”互斥程序体
        HANDLE hMutexSuicide = :: CreateMutex(
            NULL,                       // 缺省的安全性
            TRUE,                       // 最初拥有的
            g_szMutexName) ;                // 为其命名
        if (hMutexSuicide != NULL)
        {
            // 创建子进程
            std :: cout << “Creating the child process.” << std :: endl;
            :: StartClone() ;

            // 暂停
            :: Sleep(5000) ;

            // 指令子进程“杀”掉自身
            std :: cout << “Telling the child process to quit. ” << std :: endl;
            :: ReleaseMutex(hMutexSuicide) ;

            // 消除句柄
            :: CloseHandle(hMutexSuicide) ;
        }
}

void Child()
{
        // 打开“自杀”互斥体
        HANDLE hMutexSuicide = :: OpenMutex(
            SYNCHRONIZE,                // 打开用于同步
            FALSE,                      // 不需要向下传递
            g_szMutexName) ;                // 名称
        if (hMutexSuicide != NULL)
        {
            // 报告正在等待指令
            std :: cout << “Child waiting for suicide instructions. ” << std :: endl;
            :: WaitForSingleObject(hMutexSuicide, INFINITE) ;

            // 准备好终止，清除句柄
            std :: cout << “Child quiting. ” << std :: endl;
            :: CloseHandle(hMutexSuicide) ;
        }
}

int main(int argc, char* argv[] )
{
        // 决定其行为是父进程还是子进程
        if (argc > 1 && :: strcmp(argv[1] , “child” ) == 0)
        {
            Child() ;
        }
        else
        {
```

```
            Parent() ;
        }
        return 0;
    }
```

清单 3-7 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“Build”菜单中的“Compile 3-7.cpp”命令，再单击“是”按钮确认。系统对 3-7.cpp 进行编译。

步骤 15：编译完成后，单击“Build”菜单中的“Build 3-7.exe”命令，建立 3-7.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？

_____

_____

步骤 16：在工具栏单击“Execute Program”按钮，执行 3-7.exe 程序。

运行结果：

1) _____

表示：_____

2) _____

表示：_____

步骤 17：在熟悉清单 3-7 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。