Windows操作系统

C/C++ 程序实验

首都师范大学 信息工程学院

系统分析与管理实验室

**实验一 Windows 进程观测**

### 一、背景知识

Windows 可以识别的应用程序包括控制台应用程序、GUI应用程序和服务应用程序。控制台应用程序可以创建GUI，GUI应用程序可以作为服务来运行，服务也可以向标准的输出流写入数据。不同类型应用程序间的唯一重要区别是其启动方法。

Windows提供了创建控制台应用程序的能力，使用户可以利用标准的C++工具，如iostream库中的cout和cin对象，来创建小型应用程序。当系统运行时，Windows 的服务通常要向系统用户提供所需功能。

服务应用程序类型需要ServiceMail( ) 函数，由服务控制管理器 (SCM) 加以调用。SCM是操作系统的集成部分，负责响应系统启动以开始服务、指导用户控制或从另一个服务中来的请求。其本身负责使应用程序的行为像一个服务。通常，服务登录到特殊的LocalSystem账号下，此账号具有与开发人员创建的服务不同的权限。

当令C++ 编译器创建可执行程序时，编译器将源代码编译成OBJ文件，然后将其与标准库相链接。产生的EXE文件是装载器指令、机器指令和应用程序的数据的集合。装载器指令告诉系统从哪里装载机器代码。另一个装载器指令告诉系统从哪里开始执行进程的主线程。在进行某些设置后，进入开发者提供的main( ) 、ServiceMain( ) 或WinMain( ) 函数的低级入口点。机器代码中包括有控制逻辑，它所做的事包括跳转到Windows API函数，进行计算或向磁盘写入数据等。

Windows允许开发人员将大型应用程序分为较小的、互相有关系的服务模块，即动态链接库 (DLL) 代码块，在其中包含应用程序所使用的机器代码和应用程序的数据。

### 二、实验目的

通过对Windows编程，进一步熟悉操作系统的基本概念，较好地理解Windows 的结构。

### 三、实验内容与步骤

#### 1. 简单的控制台应用程序

我们先来创建一个名为“Hello，World”的应用程序。

**步骤1**：登录进入Windows 。

**步骤2**：在“开始”菜单中单击“程序”-“附件”-“记事本”命令，将清单1-l中的程序键入记事本中，并把代码保存在D盘根目录下，文件名保存为Hello.cpp。

**清单1-1 一个简单的Windows 控制台应用程序**

// hello项目

# include <iostream>

void main()

{

std::cout << “Hello, Windows !” << std :: endl ;

system(“pause”);

}

**步骤3**：在“开始”菜单中单击“程序”-“附件”-“命令提示符”命令，进入Windows“命令提示符”窗口，并利用简单的标准命令行：

D:\> CL Hello.cpp

来创建可执行的Hello.EXE。

操作能否正常进行？如果不行，则可能的原因是什么？

操作无法正常运行原因为.cpp文件中使用了中文字符

**步骤4**：运行Hello.EXE程序，产生用户键入的一行文字。

运行结果 (如果运行不成功，则可能的原因是什么？) ：

运行成功，输出Hello,Windows!

请按任意键继续. . .

#### 2. GUI应用程序

在下面的实验中，C++ 编译器创建一个GUI应用程序，代码中包括了WinMain( ) 方法，这是GUI类型的应用程序的标准入口点。

**步骤5**：在“开始”菜单中单击“程序”-“附件”-“记事本”命令，将清单1-2中的程序键入记事本中，并把代码保存在D盘根目录下，文件名为1-2.cpp。

**清单1-2 Windows 的GUI应用程序**

// msgbox项目

# include <windows.h> // 标准的include

// 告诉连接器与包括MessageBox API函数的user32库进行连接

# pragma comment(lib, “user32.lib” )

// 这是一个可以弹出信息框然后退出的筒单的应用程序

int APIENTRY WinMain(HINSTANCE /\* hInstance \*/ ,

HINSTANCE /\* hPrevInstance \*/ ,

LPSTR /\* lpCmdLine \*/ ，

int /\* nCmdShow \*/ )

{

:: MessageBox(

NULL, // 没有父窗口

“Hello, Windows !” , // 消息框中的文本

“Greetings”, // 消息框标题

MB\_OK) ; // 其中只有一个OK按钮

// 返回0以便通知系统不进入消息循环

return(0) ;

}

也可以利用任何其他文本编辑器键入程序代码，如果这样，例如使用WORD来键入和编辑程序，则应该注意什么问题？

使用WORD来键入和编辑程序应该注意区分中英文字符（标点符号）。

**步骤6**：在“命令提示符”窗口输入如下命令，产生1-2.EXE 文件，然后运行。

D:\> CL 1-2.cpp

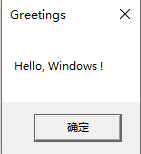
在清单1-2的GUI应用程序中，首先需要Windows.h头文件，以便获得传送给WinMain( ) 和MessageBox( ) API函数的数据类型定义。

接着的pragma指令指示编译器/连接器找到User32.LIB库文件并将其与产生的EXE文件连接起来。这样就可以运行简单的命令行命令CL MsgBox.CPP来创建这一应用程序，如果没有pragma指令，则MessageBox( ) API函数就成为未定义的了。这一指令是Visual Studio C++ 编译器特有的。

接下来是WinMain( ) 方法。其中有四个由实际的低级入口点传递来的参数。hInstance参数用来装入与代码相连的图标或位图一类的资源，无论何时，都可用GetModuleHandle( ) API函数将这些资源提取出来。系统利用实例句柄来指明代码和初始的数据装在内存的何处。句柄的数值实际上是EXE文件映像的基地址，通常为0x00400000。下一个参数hPrevInstance是为向后兼容而设的，现在系统将其设为NULL。应用程序的命令行 (不包括程序的名称) 是lpCmdLine参数。另外，系统利用nCmdShow参数告诉应用程序如何显示它的主窗口 (选项包括最小化、最大化和正常) 。

最后，程序调用MessageBox( ) API函数并退出。如果在进入消息循环之前就结束运行的话，最后必须返回0。

运行结果 (试将其中的信息与清单1-1程序的运行结果进行比较) ：

运行成功，输出

#### 2. 进程对象

操作系统将当前运行的应用程序看作是进程对象。利用系统提供的惟一的称为句柄 (HANDLE) 的号码，就可与进程对象交互。这一号码只对当前进程有效。

本实验表示了一个简单的进程句柄的应用。在系统中运行的任何进程都可调用GetCurrentProcess( ) API函数，此函数可返回标识进程本身的句柄。然后就可在Windows需要该进程的有关情况时，利用这一句柄来提供。

**步骤7**：将清单1-3.cpp程序键入记事本中，并把代码保存在D盘根目录下，文件名为1-3.cpp。

**清单1-3 获得和使用进程的句柄**

// prochandle项目

# include <windows.h>

# include <iostream>

  // 确定自己的优先权的简单应用程序

void main( )

{

// 从当前进程中提取句柄

HANDLE hProcessThis = :: GetCurrentProcess( ) ;

// 请求内核提供该进程所属的优先权类

DWORD dwPriority = :: GetPriorityClass(hProcessThis) ;

  // 发出消息，为用户描述该类

std :: cout << “Current process priority: ” ;

switch(dwPriority)

{

case HIGH\_PRIORITY\_CLASS:

std :: cout << “High” ;

break;

case NORMAL\_PRIORITY\_CLASS:

std :: cout << “Normal” ;

break;

case IDLE\_PRIORITY\_CLASS:

std :: cout << “Idle” ;

break;

case REALTIME\_PRIORITY\_CLASS:

std :: cout << “Realtime” ;

break;

default:

std :: cout << “<unknown>” ;

break;

}

std :: cout << std :: endl;

system(“pause”);

}

清单1-3中列出的是一种获得进程句柄的方法。对于进程句柄可进行的惟一有用的操作是在API调用时，将其作为参数传送给系统，正如清单1-3中对GetPriorityClass( ) API函数的调用那样。在这种情况下，系统向进程对象内“窥视”，以决定其优先级，然后将此优先级返回给应用程序。

OpenProcess( ) 和CreateProcess( ) API函数也可以用于提取进程句柄。前者提取的是已经存在的进程的句柄，而后者创建一个新进程，并将其句柄提供出来。

**步骤8**：在“命令提示符”窗口输入如下命令，产生1-3.EXE 文件，然后运行。

D:\> CL 1-3.cpp

运行结果：Current process priority: Normal

请按任意键继续. . .

**\*选作：**启动“Windows 任务管理器”，在“进程”菜单中选择进程“1-3.exe”，右击进程选择“设置优先级”，可修改进程的优先级。修改程序，显示优先级修改前后的结果对比。

运行结果：修改代码

// prochandle项目

# include <windows.h>

# include <iostream>

// 确定自己的优先权的简单应用程序

void main( )

{

char a;

std::cout<<"Enter Character a To Terminate It..."<<std::endl;

std::cin>>a;

while (a!='a'){

// 从当前进程中提取句柄

HANDLE hProcessThis = :: GetCurrentProcess( ) ;

// 请求内核提供该进程所属的优先权类

DWORD dwPriority = :: GetPriorityClass(hProcessThis) ;

// 发出消息，为用户描述该类

std :: cout << "Current process priority: " ;

switch(dwPriority)

{

case HIGH\_PRIORITY\_CLASS:

std :: cout << "High" ;

break;

case NORMAL\_PRIORITY\_CLASS:

std :: cout << "Normal" ;

break;

case IDLE\_PRIORITY\_CLASS:

std :: cout <<"Idle" ;

break;

case REALTIME\_PRIORITY\_CLASS:

std :: cout << "Realtime";

break;

default:

std :: cout << "<unknown>" ;

break;

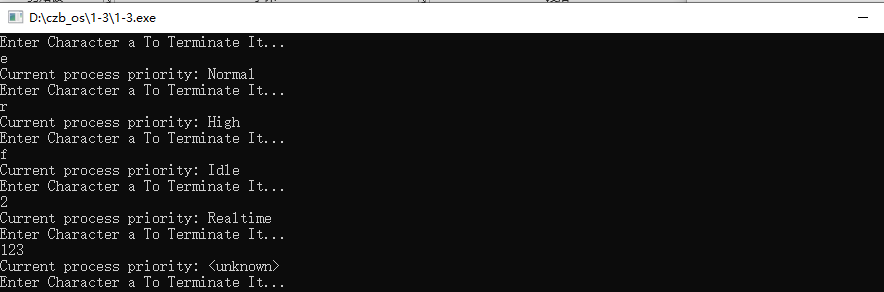
}

std::cout<<std::endl<<"Enter Character a To Terminate It..."<<std::endl;

std::cin>>a;

}

system("pause");

} 

**步骤9**：将清单1-4.cpp程序键入记事本中，并把代码保存在D盘根目录下，文件名为1-4.cpp。

清单1-4显示如何找出系统中正在运行的所有进程，如何利用OpenProcess( ) API函数来获得每一个访问进程的进一步信息。

**清单1-4 利用句柄查出进程的详细信息**

// proclist项目

# include <windows.h>

# include <tlhelp32.h>

# include <iostream>

  // 当在用户模式机内核模式下都提供所耗时间时，在内核模式下进行所耗时间的64位计算的帮助方法

DWORD GetKernelModePercentage(const FILETIME & ftKernel,

const FILETIME & ftUser)

{

// 将FILETIME结构转化为64位整数

ULONGLONG qwKernel =

( ( (ULONGLONG) ftKernel.dwHighDateTime) << 32) +

ftKernel.dwLowDateTime;

ULONGLONG qwUser =

( ( (ULONGLONG) ftUser.dwHighDateTime) << 32) +

ftUser.dwLowDateTime;

  // 将消耗时间相加，然后计算消耗在内核模式下的时间百分比

ULONGLONG qwTotal = qwKernel + qwUser;

DWORD dwPct;

if (qwTotal == 0) {

dwPct = 0;

}

else {

dwPct =

(DWORD)(((ULONGLONG)100 \* qwKernel) / qwTotal);

}

  return(dwPct) ;

}

  // 以下是将当前运行进程名和消耗在内核模式下的时间百分数都显示出来的应用程序

void main( )

{

// 对当前系统中运行的进程拍取“快照”

HANDLE hSnapshot = :: CreateToolhelp32Snapshot(

TH32CS\_SNAPPROCESS, // 提取当前进程

0) ; // 如果是当前进程，就将其忽略

  // 初始化进程入口

PROCESSENTRY32 pe;

:: ZeroMemory(&pe, sizeof(pe) ) ;

pe.dwSize = sizeof(pe) ;

  // 按所有进程循环

BOOL bMore = :: Process32First(hSnapshot, &pe) ;

while(bMore)

{

// 打开用于读取的进程

HANDLE hProcess = :: OpenProcess(

PROCESS\_QUERY\_INFORMATION, // 指明要得到信息

FALSE, // 不必继承这一句柄

pe.th32ProcessID) ; // 要打开的进程

if (hProcess != NULL)

{

// 找出进程的时间

FILETIME ftCreation, ftExit, ftKernelMode, ftUserMode;

:: GetProcessTimes(

hProcess, // 所感兴趣的进程

&ftCreation, // 进程的启动时间 (绝对的)

&ftExit, // 结束时间 (如果有的话)

&ftKernelMode, // 在内核模式下消耗的时间

&ftUserMode) ; // 在用户模式下消耗的时间

  // 计算内核模式消耗的时间百分比

DWORD dwPctKernel = :: GetKernelModePercentage(

ftKernelMode, // 在内核模式上消耗的时间

ftUserMode ) ; // 在用户模式下消耗的时间

  // 向用户显示进程的某些信息

std :: cout << “Process ID: ” << pe.th32ProcessID

<< “, EXE file: ” << pe.szExeFile

<< “, % in kernel mode: ” << dwPctKernel

<< std :: endl;

// 消除句柄

:: CloseHandle(hProcess) ;

}

// 转向下一个进程

bMore = :: Process32Next(hSnapshot, &pe) ;

}

system(“pause”);

}

**步骤10**：在“命令提示符”窗口输入如下命令，产生1-4.EXE 文件，然后运行。

D:\> CL 1-4.cpp

运行结果：Process ID: 1204, EXE file: sihost.exe, % in kernel mode: 53

Process ID: 3948, EXE file: svchost.exe, % in kernel mode: 21

Process ID: 3984, EXE file: svchost.exe, % in kernel mode: 39

Process ID: 2660, EXE file: taskhostw.exe, % in kernel mode: 65

Process ID: 6140, EXE file: ctfmon.exe, % in kernel mode: 66

Process ID: 6436, EXE file: explorer.exe, % in kernel mode: 64

Process ID: 6604, EXE file: QQPCTray.exe, % in kernel mode: 37

Process ID: 6696, EXE file: ChsIME.exe, % in kernel mode: 0

Process ID: 6984, EXE file: svchost.exe, % in kernel mode: 50

Process ID: 7444, EXE file: StartMenuExperienceHost.exe, % in kernel mode: 42

Process ID: 7728, EXE file: RuntimeBroker.exe, % in kernel mode: 70

Process ID: 8084, EXE file: TextInputHost.exe, % in kernel mode: 44

Process ID: 7484, EXE file: SearchApp.exe, % in kernel mode: 38

Process ID: 7264, EXE file: RuntimeBroker.exe, % in kernel mode: 65

Process ID: 8612, EXE file: RuntimeBroker.exe, % in kernel mode: 84

Process ID: 9076, EXE file: StudentMain.exe, % in kernel mode: 66

Process ID: 8868, EXE file: EzMonitor.exe, % in kernel mode: 95

Process ID: 8592, EXE file: secmon.exe, % in kernel mode: 100

Process ID: 6156, EXE file: dllhost.exe, % in kernel mode: 100

Process ID: 8564, EXE file: SystemSettings.exe, % in kernel mode: 47

Process ID: 3144, EXE file: ApplicationFrameHost.exe, % in kernel mode: 100

Process ID: 8720, EXE file: UserOOBEBroker.exe, % in kernel mode: 100

Process ID: 8008, EXE file: svchost.exe, % in kernel mode: 100

Process ID: 3120, EXE file: ShellExperienceHost.exe, % in kernel mode: 67

Process ID: 1644, EXE file: RuntimeBroker.exe, % in kernel mode: 80

Process ID: 8356, EXE file: taskhostw.exe, % in kernel mode: 39

Process ID: 2088, EXE file: WINWORD.EXE, % in kernel mode: 41

Process ID: 5132, EXE file: WINWORD.EXE, % in kernel mode: 23

Process ID: 5692, EXE file: cmd.exe, % in kernel mode: 77

Process ID: 6416, EXE file: conhost.exe, % in kernel mode: 82

Process ID: 8588, EXE file: svchost.exe, % in kernel mode: 50

Process ID: 8316, EXE file: Code.exe, % in kernel mode: 36

Process ID: 4368, EXE file: Code.exe, % in kernel mode: 100

Process ID: 2552, EXE file: Code.exe, % in kernel mode: 68

Process ID: 1944, EXE file: Code.exe, % in kernel mode: 50

Process ID: 5520, EXE file: Code.exe, % in kernel mode: 24

Process ID: 4776, EXE file: Code.exe, % in kernel mode: 38

Process ID: 1932, EXE file: Code.exe, % in kernel mode: 44

Process ID: 8576, EXE file: Code.exe, % in kernel mode: 19

Process ID: 4136, EXE file: smartscreen.exe, % in kernel mode: 100

Process ID: 8804, EXE file: 1-4.exe, % in kernel mode: 50

Process ID: 8980, EXE file: conhost.exe, % in kernel mode: 66

Process ID: 4484, EXE file: cmd.exe, % in kernel mode: 0

Process ID: 5408, EXE file: 1-4.exe, % in kernel mode: 100

Process ID: 4744, EXE file: conhost.exe, % in kernel mode: 50

Process ID: 1280, EXE file: cmd.exe, % in kernel mode: 0

Process ID: 5956, EXE file: 1-4.exe, % in kernel mode: 100

Process ID: 9112, EXE file: conhost.exe, % in kernel mode: 88

Process ID: 4432, EXE file: cmd.exe, % in kernel mode: 100

Process ID: 1572, EXE file: 1-4.exe, % in kernel mode: 50

Process ID: 4264, EXE file: conhost.exe, % in kernel mode: 16

Process ID: 7580, EXE file: cmd.exe, % in kernel mode: 0

Process ID: 8624, EXE file: 1-4.exe, % in kernel mode: 0

Process ID: 3468, EXE file: conhost.exe, % in kernel mode: 33

程序的功能描述

程序通过对所有正在运行的进程进行循环遍历将当前运行进程名和消耗在内核模式下的时间百分数都显示出来。尝试多开1-4.exe后可以观察到不同的kernel mode百分比占比。