操作系统

实验报告

实验名称： 命令解释程序

院系： 计算机学院·网络空间安全学院

班级：

学号：

姓名：

指导老师：匡林爱

日期：2020年11月24日

1. **实验目的和要求：**

* 掌握命令解释程序的设计方法。
* 学习Windows系统调用的使用，了解目录操作、进程控制等相关知识。
* 培养C/C++语言程序设计技能，提高程序设计和文档编写能力。
* 锻炼团队成员的交流与合作能力。

本实验要求实现一个简单的命令解释程序，其设计类似于MS-DOS的Command程序。具体要求如下：

(1) 参考Command命令解释程序，采用控制台命令行输入方式，命令行提示符是当前目录名与提示符“>”，在提示符后输入命令。命令执行结束后，在控制台继续显示提示符，等待输入新的命令。

(2) 实现以下内部命令：

* cd <路径名> 切换当前目录。
* dir [<路径名>] 显示指定目录下的文件、子目录及磁盘空间等相关信息。
* tasklist 显示系统当前进程信息，包括进程标识符pid、该进程包含的线程数、进程名等。
* taskkill <pid> 结束系统中正在运行的进程，须指定进程标识符pid。
* history 显示控制台中曾经输入过的命令。
* help 显示本程序使用的帮助信息。
* exit 退出控制台，结束本命令解释程序。

(3) 对前台进程和后台进程的操作

本实验设计的命令解释程序可以将进程放在前台执行或者后台执行。

启动前台进程的方法是在提示符下输入命令行：

fp <可执行文件名(含路径)>

启动后台进程的方法是在提示符下输入命令行：

bg& <可执行文件名(含路径)>

在前台进程运行期间，解释程序一直等待，直到前台进程运行结束，才再显示提示符；而在后台进程运行期间，解释程序不必等待，会立即显示提示符，允许用户输入下一条命令。

(4) 命令解释程序还需要捕获Ctrl+C组合键的信号来结束**前台正在运行的控制台进程**，并返回用户输入界面(显示提示符)，等待新命令输入。本实验程序利用系统自备功能，来实现此功能。(注：若前台进程是图形界面，则按Ctrl+C并不能使其结束，而是使本实验的命令解释程序结束。)

(5) 其他要求

该命令解释程序应具有相应的出错提示功能。

程序每次接收用户输入的一行命令，在用户输入回车键(Enter)后开始执行命令。

若输入命令时仅输入回车键，则不作任何操作，重新显示提示符，等待用户输入新的命令。

定义空格为分隔符，程序应能处理命令中出现的重复空格符。

1. **实验内容：**

### 背景介绍：

命令解释程序是**用户**和**系统内核**之间的接口程序。对于Windows系统来说，由于已经提供了具有良好交互性的图形用户界面，传统的控制台命令解释程序已经很少被广大用户所了解和使用。但是，对于某些应用，例如使用一条命令删除所有扩展名为tmp的文件，或者删除某些具有特殊名字的病毒文件，在图形用户界面下很难甚至不能完成。这需要通过Windows提供的Command命令接口来完成。Command程序是一个命令解释器，它拥有自己的内部命令集，用户和其他应用程序都可以通过对Command程序的调用完成与系统内核的交互。

### Win32 API相关函数介绍：

**1．GetCurrentDirectory函数**

功能：查找当前进程的当前目录，调用成功，返回装载到lpBuffer的字节数。失败则返回0。

格式：

DWORD GetCurrentDirectory ( //DWORD就是unsigned long

DWORD nBufferLength, // 缓冲区的长度

LPTSTR lpBuffer // 指定一个预定义字串，用于装载当前目录

) //LPSTR：是指向字符串的指针的类型名，即char \*

【注】API中涉及的类型名，请参阅配套文档“Win32 Simple Data Types.doc”。

**2．WaitForSingleObject函数**

功能：等待一个事件信号直至信号出现或者超时。若等到信号则返回WAIT\_OBJECT\_0(即0)，若等待超过dwMiliseconds时间还是无信号，则返回WAIT\_TIMEOUT(即258)。若函数调用失败，则返回WAIT\_FAILED (即-1)。

格式：

DWORD WaitForSingleObject (

HANDLE hHandle, // 事件的句柄

DWORD dwMilliseconds // 最大等待时间，以ms计时。

)

**3．SetCurrentDirectory函数**

功能：设置当前目录。返回非0表示成功，返回0表示失败。

格式：

BOOL SetCurrentDirectory (

LPCTSTR lpPathName // 新设置的当前目录路径

)

**4．FindFirstFile函数**

功能：用于从一个文件夹(包括子文件夹)中查找指定文件，返回找到的文件句柄。若调用失败，则返回INVALID\_HANDLE\_VALUE (即-1)。

格式：

HANDLE FindFirstFile (

LPCTSTR lpFileName, // 文件名字符串(可用通配符)

LPWIN32\_FIND\_DATA lpFindFileData // 指向一个用于保护文件的结构体

)

【注】WIN32\_FIND\_DATA结构的说明请参看MSDN或本上机实验指导的配套文档。

**5．FindNextFile函数**

功能：继续查找FindFirstFile函数搜索后的文件。它返回的文件句柄可以作为参数用于FindNextFile函数。这样就可方便地枚举出与lpFileName参数指定的文件名相匹配的所有文件。调用失败，返回0。

格式：

HANDLE FindNextFile (

HANDLE hFindFile, // 前一个搜素到的文件句柄

LPWIN32\_FIND\_DATA lpFindFileData // 指向一个用于保护文件的结构体

)

**6．GetVolumeInformation函数**

功能：用于获取磁盘相关信息。执行成功返回非0；失败，返回0。

格式：

BOOL GetVolumeInformation (

LPCTSTR lpRootPathName, // 磁盘驱动器代码字符串(具体构成方法参看程序)

LPCTSTR lpVolumeNameBuffer, // 磁盘驱动器卷标名称

DWORD nVolumeNameSize, // 磁盘驱动器卷标名称长度

LPWORD lpVolumeSerialNumber, // 磁盘驱动器卷标序列号

LPWORD lpMaximunComponentLength, //系统允许的最大文件长度

LPWORD lpFileSystemFlags, // 文件系统标识

LPCTSTR lpFileSystemNameBuffer, // 文件系统名称

DWORD nFileSystemNameSize // 文件系统名称长度

)

**7．GetDiskFreeSppaceEx函数**

功能：获取与一个磁盘的组织以及剩余容量有关的信息。调用失败返回0。

格式：

HANDLE GetDiskFreeSppaceEx (

LPCTSTR lpRootPathName, // 不包括卷名的磁盘根路径名

PULARGE\_INTEGER lpFreeBytesAvailableToCaller, // 调用者可用的字节数

PULARGE\_INTEGER lpTotalNumberOfBytes, // 磁盘上的总字节数

PULARGE\_INTEGER lpTotalNumberOfFreeBytes // 磁盘上的可用字节数

)

参数说明：lpRootPathName：根路径名。例如形式为"C:\\"。使用NULL表示函数使用当前目录所在的磁盘。

**8．FileTimeToLocalFileTime函数**

功能：将一个FILETIME结构转换成本地时间。

格式：

BOOL FileTimeToLocalFileTime (

const FILETIME\* lpFileTime, // 指向一个包含了UTC时间信息的结构

LPFILETIME lpLocalFileTime // 用于装载转换过的本地时间的结构体

)

**9．FileTimeToSystemTime函数**

功能：根据一个FILETIME结构的内容，装载一个SYSTENTIME结构。

格式：

BOOL FileTimeToSystemTime (

const FILETIME\* lpFileTime, // 指向一个包含了文件时间信息的结构

LPFILETIME lpSystemTime // 用于装载系统时间的结构体

)

**10．CreateToolhelp32Snapshot函数**

功能：为指定的进程、进程使用的堆(heap)、模块(module)、线程(thread)建立一个快照(snapshot)。快照建立成功则返回快照的句柄，失败则返回INVALID\_HANDL\_VALUE。

格式：

HANDLE WINAPI CreateToolhelp32Snapshot (

DWORD dwFlags, // 指向快照中包含的系统内容

DWORD th32ProcessID // 指定将要快照的进程ID

)

**11．Process32First函数**

功能：是一个进程获取函数，当使用CreateToolhelp32Snapshot()函数获得当前运行进程的快照后，可以使用Process32First ( )函数获得第一个进程的句柄。

格式：

BOOL WINAPI Process32First (

HANDLE hSnapshot, // 快照句柄

LPPROCESSENTRY32 lppe // 指向一个保护进程快照信息的LPPROCESSENTRY32结构

)

**12．Process32Next函数**

功能：获取快照中下一个进程信息。

格式：

BOOL WINAPI Process32Next (

HANDLE hSnapshot, // 由Process32First或Process32Next函数获得的快照句柄

LPPROCESSENTRY32 lppe // 指向一个保护进程快照信息的LPPROCESSENTRY32结构

)

**13．OpenProcess函数**

功能：该函数打开一个已经存在的进程对象，若成功，返回值是指定进程的打开句柄。若失败，则返回空值。

格式：

HANDLE OpenProcesst (

DWORD dwDesiredAccess, // 权限标识（详见MSDN）

BOOL bInheritHandle, // 指出返回的句柄是否能被当前进程创建的新进程继承，

// TRUE表示可继承，FALSE表示不能继承。

DWORD dwProcessID // 进程ID

)

**14．SetConsoleCtrlHandler函数**

功能：添加或删除一个事件钩子(Handler)。

格式：

BOOL SetConsoleCtrlHandler (

PHANDLER\_ROUTINE HandlerRoutine, // 回调函数的指针

BOOL Add // 表示添加或删除

)

**15．CreateProcess函数**

此函数已经在前面介绍，请参阅“上机实验一”的1.2.2节，在此不再赘述。

**16．GetExitCodeProcess函数**

功能：获取一个已中断进程的退出代码。

格式：

BOOL GetExitCodeProcess (

HANDLE hProcess, // 进程句柄

LPDWORD lpExitCode // 指向接受退出码的变量

)

**17．TerminateProcess函数**

功能：以给定的退出码终止进程。

格式：

BOOL TerminateProcee (

HANDLE hProcess, // 进程句柄

UINT uExitCode // 进程的退出码

)

### 重要的数据结构

**1．历史命令循环数组(队列)**

在history命令中，用数组来存放输入过的历史命令。程序中假设该数组的元素个数为20，数组元素的结构定义如下：

typedef struct ENV\_HISTORY {

int start; // 队列的头指针

int end; // 队列的尾指针

char his\_cmd[20][128]; // 队列数组(顺序结构的队列)

} ENV\_HISTORY;

ENV\_HISTORY envhis; // 定义队列变量(为队列分配内存空间)

**2．文件信息链表**

程序中，需要把dir命令取得的文件信息用链表保存，输出这些信息时对链表遍历。

链表结点的定义如下：

struct files\_Content {

FILETIME time; // 文件创建时间

char name[200]; // 文件名

int type; // type=1普通文件, type=0目录

int size; // 文件大小

files\_Content \*next; // 构成链表的链接指针

} ;

1. **实验步骤：**

主程序的流程如图1所示。

**1．解析命令**

解析命令就是分析输入的命令行(input数组)，分离命令行中的命令和参数。命令和参数的分隔是由空格符完成的。将命令存入arg[0]指向的字符串，将参数存入arg[1]指向的字符串中。

**2．命令处理**

命令出路与执行命令的目的有关，其中系统调用是重要的组成部分。

初始化环境

显示提示符

等待用户输入命令

解析命令

是exit?

寻找执行命令的函数

执行命令

结束

N

Y

图1 程序流程图

void cd\_cmd(char \*route)

{

if (!SetCurrentDirectory(route))

{ // 设置当前目录，若失败则输出出错信息

cout<<" SetCurrentDirectory failed ";

cout<<GetLastError()<<endl;

}

}

以上是cd命令的处理函数，涉及的Windows API为

SetCurrentDirectory( )函数，它的作用是设置当前目录为指定路径，若失败则返回出错信息。其中出错号从另一个API函数GetLastError( )获得。

1. **实验源程序**

**2．WinShell.h**

**#define BUFSIZE MAX\_PATH**

**#define HISNUM 20 //最多可以保存20个历史命令**

**char buf[BUFSIZE];**

**//保存历史命令的结构**

**typedef struct ENV\_HISTORY {**

**int start; // 队列的头指针**

**int end; // 队列的尾指针**

**char his\_cmd[20][128]; // 队列数组(顺序结构的队列)**

**} ENV\_HISTORY;**

**ENV\_HISTORY envhis; // 定义队列变量(为队列分配内存空间)**

**//说明：因envhis是全局变量(属静态变量)，故其成员star,end有初值0**

**//保存文件或目录相关信息的结构**

**struct files\_Content {**

**FILETIME time; // 文件创建时间**

**char name[200]; // 文件名**

**int type; // type=1普通文件, type=0目录**

**int size; // 文件大小**

**files\_Content \*next; // 构成链表的链接指针**

**};**

**2．WinShell.cpp**

**#define \_Win32\_WINNT 0x0501**

**#include <stdlib.h> //atoi()等**

**#include <iostream>**

**#include <windows.h> //DWORD;HANDLE...其中还有许多头文件**

**#include <tlhelp32.h> //CreateToolhelp32Snapshot()**

**#include <string.h>**

**#include "WinShell.h"**

**using namespace std;**

**// 以下两个函数在主函数开头声明后放在主函数后面不行，故将它们移**

**// 至main()的前面，原因可能是它们的参数类型分别是FILETIME和DWORD**

**// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 时间处理函数 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**void ftime(FILETIME filetime)**

**{**

**SYSTEMTIME systemtime;**

**if (filetime.dwLowDateTime == -1) // Win32时间的低32位**

**cout << "Never Expires\n";**

**else**

**{**

**//将UTC(Universal Time Coordinated)文件时间转换成本地文件时间**

**if (FileTimeToLocalFileTime(&filetime, &filetime) != 0)**

**{**

**//将64位时间转化成系统时间**

**if (FileTimeToSystemTime(&filetime, &systemtime) != 0)**

**{**

**//以一定能格式输出时间**

**cout.fill('0'); //不足指定宽度是用0填充**

**cout << dec << systemtime.wYear << '-';**

**cout.width(2); cout << systemtime.wMonth << '-'; //月份用2位显示，下类似**

**cout.width(2); cout << systemtime.wDay << " ";**

**cout.width(2); cout << systemtime.wHour << ':';**

**cout.width(2); cout << systemtime.wMinute;**

**}**

**else**

**cout << "FileTimeToSystemTime failed\n";**

**}**

**else**

**cout << "FileTimeToLocalFileTime failed\n";**

**}**

**cout.fill(' '); //恢复空格填充**

**}**

**// \*\*\*\*\*\*\*\*\*\*\*\* 回调函数 \*\*\*\*\*\*\*\*\*\*\*\***

**BOOL WINAPI ConsoleHandler(DWORD CEvent)**

**{ // 此函数不做什么，由系统处理事件，包括按下Ctrl+C等**

**switch (CEvent)**

**{**

**case CTRL\_C\_EVENT:**

**break;**

**case CTRL\_BREAK\_EVENT:**

**break;**

**case CTRL\_CLOSE\_EVENT:**

**break;**

**case CTRL\_LOGOFF\_EVENT:**

**break;**

**case CTRL\_SHUTDOWN\_EVENT:**

**break;**

**}**

**return TRUE;**

**}**

**// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 主函数 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**int main()**

**{**

**//声明程序中用到的函数**

**void cd\_cmd(char \*dir); // cd命令处理函数**

**void dir\_cmd(char \*dir); // dir命令处理函数**

**void GetProcessList(); // 获得系统当前进程列表**

**void history\_cmd(); // 获得最近输入的命令**

**void add\_history(char \*); // 将输入命令行添加到命令历史中**

**HANDLE process(int, char[]); // 创建进程**

**BOOL killProcess(char \*); // kill进程**

**void help(); // 显示帮助信息**

**char c, \*input, \*arg[2], path[BUFSIZE];**

**int input\_len = 0, is\_bg = 0, i, j, k;**

**HANDLE hprocess; // 进程执行结束，返回进程句柄**

**DWORD dwRet;**

**while (true)//显示提示符，等待用户输入命令是个无限循环过程**

**{**

**// 将指向输入命令的指针数组初始化**

**for (i = 0; i < 2; i++)**

**arg[i] = NULL;**

**// 获得当前目录并存入path中，BUFSIZE是最多能够保存的路径名长度**

**dwRet = GetCurrentDirectory(BUFSIZE, path);//返回目录数据实际长度存于dwRet**

**if (dwRet == 0) // 返回当前目录失败，输出出错信息**

**cout << "GetCurrentDirectory failed " << GetLastError() << endl;**

**else if (dwRet > BUFSIZE)// BUFSIZE长度小于返回目录数据的长度，输出出错信息**

**cout << "GetCurrentDirectory failed (buffer too small; need " << dwRet << "bytes)\n";**

**else**

**cout << path << '>'; // 显示提示符(当前目录名+'>')**

**// \*\*\*\*\*\*\*\*\*\*\* 键盘输入 \*\*\*\*\*\*\*\*\*\*\*\***

**input\_len = 0;**

**// 将命令开头的无用字符过滤掉**

**while ((c = cin.get()) == ' ' || c == '\t' || c == EOF);**

**if (c == '\n') //输入为空命令(仅输入回车符)时**

**continue; //结束本次循环，回到循环开头，重新显示提示符**

**while (c != '\n')**

**{**

**buf[input\_len++] = c;**

**c = cin.get();**

**}**

**buf[input\_len++] = '\0'; // 加上串结束符**

**// 分配动态存储空间，将命令从缓存复制到input中**

**input = new char[input\_len];**

**strcpy(input, buf); //为了便于后边的处理，将命令行复制到input中**

**// \*\*\*\*\*\*\*\*\*\*\* 解析命令 \*\*\*\*\*\*\*\*\*\*\*\***

**for (i = 0, j = 0, k = 0; i < input\_len && k < 2; i++)//k<2是限制只处理1个命令参数**

**{ //即arg[0]为命令，arg[1]为参数**

**if (input[i] == ' ' || input[i] == '\0')**

**{**

**if (j == 0) // 去掉连在一起的多个空格**

**continue;**

**else**

**{**

**buf[j++] = '\0';**

**arg[k] = new char[sizeof(char)\*(j + 1)];**

**strcpy(arg[k++], buf); // 将命令或参数复制到arg中**

**j = 0; // 准备取下一个参数**

**}**

**}**

**else//不是' '和'\0'字符，则存入buf[]中**

**buf[j++] = input[i];**

**}**

**add\_history(input); // 将输入命令添加到历史命令队列中**

**// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 命令处理 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**if (strcmp(arg[0], "cd") == 0) // \*\*\*\* cd命令 \*\*\*\***

**{**

**if (arg[1] != NULL)**

**{**

**cd\_cmd(arg[1]);**

**delete[]arg[1];**

**}**

**else**

**cout << "cd命令必须指定路径名!\n";**

**delete[]input;**

**delete[]arg[0];**

**continue; //返回循环开头，重新显示提示符**

**}**

**if (strcmp(arg[0], "dir") == 0) // \*\*\*\* dir命令 \*\*\*\***

**{**

**char \*route;**

**if (arg[1] == NULL) // 若dir命令无参数，则对当前目录操作**

**{**

**route = path; // 取当前目录**

**dir\_cmd(route);**

**}**

**else**

**{**

**dir\_cmd(arg[1]);**

**delete[]arg[1];**

**}**

**delete[]input; // 释放堆空间**

**delete[]arg[0];**

**continue;**

**}**

**if (strcmp(arg[0], "tasklist") == 0) // \*\*\*\* tasklist命令 \*\*\*\***

**{**

**GetProcessList();// 该函数通过调用若干API函数，获取系统当前进程列表**

**delete[]input;**

**delete[]arg[0];**

**if (arg[1] != NULL)//防止用户误输入命令参数**

**delete arg[1];**

**continue;**

**}**

**if (strcmp(arg[0], "fp") == 0) // \*\*\* fp命令(前台进程) \*\*\***

**{**

**if (arg[1] == NULL)**

**{**

**cout << "没有指定可执行文件\n";**

**delete[]input;**

**delete[]arg[0];**

**continue;**

**}**

**is\_bg = 0; // 后台标志置0(不是后台进程)**

**hprocess = process(is\_bg, arg[1]); //创建进程，返回新进程的句柄**

**// 等待新进程执行完毕(INFINTE表示等待无限制)**

**if (WaitForSingleObject(hprocess, INFINITE) == WAIT\_OBJECT\_0)**

**{**

**//如果进程执行完毕，释放控制台**

**delete[]input;**

**delete[]arg[0];**

**delete[]arg[1];**

**}**

**continue;**

**}**

**if (strcmp(arg[0], "bg&") == 0) // \*\*\* bg&命令(后台进程) \*\*\***

**{**

**if (arg[1] == NULL)**

**{**

**cout << "没有指定可执行文件\n";**

**delete[]input;**

**delete[]arg[0];**

**continue;**

**}**

**is\_bg = 1; // 后台标志置1(真)**

**process(is\_bg, arg[1]); //为可执行文件arg[1]创建后台进程**

**delete[]input;**

**delete[]arg[0];**

**delete[]arg[1];**

**continue;**

**}**

**if (strcmp(arg[0], "taskkill") == 0) // \*\*\*\*\* kill进程 \*\*\*\*\***

**{**

**BOOL success;**

**if (arg[1] != NULL)**

**{**

**success = killProcess(arg[1]); // arg[1]指向进程ID**

**if (!success) // 若撤销进程失败，则显示出错信息**

**cout << "kill process failed!\n";**

**delete[]arg[1];**

**}**

**else**

**cout << "taskkill命令必须指定进程ID!" << endl;**

**delete[]input;**

**delete[]arg[0];**

**if (arg[1] != NULL)//防止用户误输入命令参数**

**delete arg[1];**

**continue;**

**}**

**if (strcmp(arg[0], "history") == 0) // \*\*\*\* 显示历史命令 \*\*\*\***

**{**

**history\_cmd();**

**delete[]input;**

**delete[]arg[0];**

**if (arg[1] != NULL)//防止用户误输入命令参数**

**delete arg[1];**

**continue;**

**}**

**if (strcmp(arg[0], "help") == 0) // \*\*\*\* help命令 \*\*\*\***

**{**

**help();**

**delete[]input;**

**delete[]arg[0];**

**if (arg[1] != NULL)//防止用户误输入命令参数**

**delete arg[1];**

**continue;**

**}**

**if (strcmp(arg[0], "exit") == 0) // \*\*\*\* exit命令 \*\*\*\***

**{**

**cout << "\nBye bye!\n\n";**

**delete[]input;**

**delete[]arg[0];**

**if (arg[1] != NULL)//防止用户误输入命令参数**

**delete arg[1];**

**break; // 退出死循环，结束程序**

**}**

**else // 输入命令不正确，给出出错信息**

**{**

**cout << "please input correct commmand!\n";**

**delete[]input;**

**if (arg[0])**

**delete[]arg[0];**

**if (arg[1])**

**delete[]arg[1];**

**continue;**

**}**

**}**

**} // \*\*\*\* 主函数结束 \*\*\*\***

**// \*\*\*\*\*\*\*\*\*\*\*\*\* 相关命令出路函数 \*\*\*\*\*\*\*\*\*\*\*\*\*\***

**void cd\_cmd(char \*route) // \*\*\*\* cd命令实现函数 \*\*\*\***

**{**

**if (!SetCurrentDirectory(route)) //设置当前目录，若失败则返回出错信息**

**cout << "SetCurrentDirectory failed " << GetLastError() << endl;**

**}**

**// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* dir命令实现函数 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**void dir\_cmd(char \*route)**

**{**

**WIN32\_FIND\_DATA FindFileData; //将找到的文件或目录以WIN32\_FIND\_DATA结构返回**

**files\_Content head, \*p, \*q; //定义指定文件结构体的头结点和指针**

**HANDLE hFind = INVALID\_HANDLE\_VALUE; // 句柄变量初值为“非法句柄值”**

**DWORD dwError; // 定义32位整数**

**char volume\_name[256], str[22];**

**int file = 0, dir = 0; //文件数和目录数初始值为0**

**\_\_int64 sum\_file = 0; //总文件大小为0字节，其值较大保存为64位整数**

**\_\_int64 l\_user, l\_sum, l\_idle; //调用者可用空间，总容量，磁盘总可用空间**

**unsigned long volume\_number; //卷序列号**

**char \*DirSpec[4];**

**head.next = NULL;**

**DirSpec[0] = new char[2];**

**strncpy(DirSpec[0], route, 1);**

**DirSpec[0][1] = '\0'; //DirSpec[0]为驱动器名**

**DirSpec[1] = new char[4];**

**strcpy(DirSpec[1], DirSpec[0]);**

**strncat(DirSpec[1], ":\\", 3); //DirSpec[1]用于获取驱动器信息**

**DirSpec[2] = new char[strlen(route) + 2];**

**DirSpec[3] = new char[strlen(route) + 5];**

**strcpy(DirSpec[2], route); //DirSpec[2]为dir命令的目录名**

**strcpy(DirSpec[3], route);**

**int len = strlen(route);**

**if (route[len - 1] != '\\')**

**strncat(DirSpec[2], "\\", 2);**

**strncat(DirSpec[3], "\\\*.\*", 5); //DirSpec[3]用于查找目录中的所有文件**

**//搜素DirSpec[3]指定的文件，文件信息存于FindFileData变量中，返回找到的文件句柄**

**hFind = FindFirstFile(DirSpec[3], &FindFileData);**

**if (hFind == INVALID\_HANDLE\_VALUE) //查找句柄返回为无效值，查找失败**

**cout << "Invalid file handle, Error is " << GetLastError() << endl;**

**else**

**{**

**//获取卷的卷名(存于volume\_name)，卷序列号(存于volume\_number)**

**GetVolumeInformation(DirSpec[1], volume\_name, 50, &volume\_number, NULL, NULL, NULL, 10);**

**if (strlen(volume\_name) == 0)**

**cout << "\n\n驱动器" << DirSpec[0] << "中的卷没有标签。" << endl;**

**else**

**cout << "\n\n驱动器" << DirSpec[0] << "中的卷是 " << volume\_name << endl;**

**cout << "卷的序列号是 " << hex << volume\_number << dec << endl << endl;;**

**cout << DirSpec[2] << " 的目录\n\n";**

**head.time = FindFileData.ftCreationTime;//获得的文件创建时间，存入文件结构体head中**

**strcpy(head.name, FindFileData.cFileName);//获得的文件名，存入文件结构体head中**

**// 若数据属性是目录，则置type为0**

**if (FindFileData.dwFileAttributes == FILE\_ATTRIBUTE\_DIRECTORY)**

**{**

**head.type = 0;**

**dir++;**

**}**

**else**

**{**

**//如果数据属性是文件，type位为1**

**head.type = 1;**

**head.size = FindFileData.nFileSizeLow; //将文件大小存入结构体head中**

**file++; //文件数增1**

**sum\_file += FindFileData.nFileSizeLow; //将文件大小(字节数)累加**

**}**

**p = &head; // p指向头结点head**

**//如果还有下一个数据，继续查找**

**while (FindNextFile(hFind, &FindFileData) != 0)**

**{ // 第二个结点开始，分配动态空间**

**q = new files\_Content[sizeof(files\_Content)];**

**q->next = NULL;**

**q->time = FindFileData.ftCreationTime; // 保存文件创建时间**

**strcpy(q->name, FindFileData.cFileName); // 保存文件名**

**if (FindFileData.dwFileAttributes == FILE\_ATTRIBUTE\_DIRECTORY)**

**{**

**q->type = 0; // 找到的是目录**

**dir++; // 目录数增1**

**}**

**else // 否则，找到的是文件**

**{**

**//如果数据属性是文件，type位为1**

**q->type = 1;**

**q->size = FindFileData.nFileSizeLow; //将文件大小存入结构体**

**file++; //文件数增1**

**sum\_file += FindFileData.nFileSizeLow; //将文件大小累加**

**}**

**p->next = q; // 构成单链表**

**p = q; // p指向新结点**

**}**

**p->next = NULL; // 链表尾结点的next指针须置为NULL**

**//将结构体中数据的创建时间、类型、大小、名称等信息依次输出**

**p = &head; // 从链表头结点开始**

**while (p != NULL)**

**{**

**ftime(p->time); // 按规定格式显示文件创建时间**

**if (p->type == 0) // 若是目录，则显示“<DOR>”**

**cout << "\t<DIR>\t\t";**

**else**

**{ // 若是文件，则按宽度为9的格式显示文件大小(字节数)**

**cout << "\t\t"; cout.width(9);**

**cout << dec << (unsigned)p->size;**

**}**

**cout << '\t' << p->name << endl; // 显示文件名**

**p = p->next; // 准备显示下一个目录项(文件或目录)**

**}**

**//显示文件和目录总数以及磁盘空间相关信息**

**cout.width(15);**

**cout << file << " 个文件\t\t\t";**

**//printf()使用格式符“%I64d”可以输出64位整数，但“cout<<”只支持32位整数**

**//故此处先将64位整数sum\_file转换成以10进制形式的字符串后再输出**

**\_i64toa(sum\_file, str, 10); //64位整数sum\_file转换成10进制字符串存于str中**

**cout << str << " 字节" << endl;**

**GetDiskFreeSpaceEx(DirSpec[1], (PULARGE\_INTEGER)&l\_user,**

**(PULARGE\_INTEGER)&l\_sum, (PULARGE\_INTEGER)&l\_idle);**

**cout.width(15);**

**cout << dir << " 个目录\t\t\t";**

**\_i64toa(l\_idle, str, 10); //64位整数l\_idle转换成10进制字符串存于str中**

**cout << str << " 可用字节\n";**

**cout.width(15);**

**\_i64toa(l\_sum, str, 10); //64位整数l\_sum转换成10进制字符串存于str中**

**cout << str << " 磁盘总字节\n\n" << endl;**

**dwError = GetLastError();**

**FindClose(hFind);**

**// 若出现其他异常情况，则输出出错信息**

**if (dwError != ERROR\_NO\_MORE\_FILES)**

**cout << "FindNextFile error. Error is " << dwError << endl;**

**//释放files\_Content结构体占用的动态空间**

**p = &head;**

**p = p->next; // head占用的不是动态空间，跳过head**

**while (p != NULL)**

**{**

**q = p->next;**

**delete p; // 依次释放files\_Content链表的后续结点**

**p = q;**

**}**

**}**

**}**

**// \*\*\*\*\*\*\*\*\*\*\*\*\*\* tasklist命令 \*\*\*\*\*\*\*\*\*\*\*\*\*\***

**void GetProcessList() // 函数功能：获取系统当前运行进程列表的命令**

**{**

**HANDLE hProcessSnap = NULL;**

**PROCESSENTRY32 pe32 = { 0 };**

**int pn = 0; // 用于累计进程数**

**// 对系统中的进程进行拍照**

**hProcessSnap = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);//pid为0表示任意进出**

**if (hProcessSnap == INVALID\_HANDLE\_VALUE)**

**cout << "\nCtreateToolhelp32Snapshot() failed:" << GetLastError();**

**//使用前要填充结构大小**

**pe32.dwSize = sizeof(PROCESSENTRY32);**

**// 列出进程**

**if (Process32First(hProcessSnap, &pe32))**

**{**

**DWORD dwPriorityClass;**

**cout << "\n优先级\t\t进程ID\t\t线程\t\t进程名\n";**

**do {**

**HANDLE hProcess;**

**hProcess = OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, pe32.th32DefaultHeapID);**

**dwPriorityClass = GetPriorityClass(hProcess);**

**CloseHandle(hProcess);**

**//输出结果**

**cout << pe32.pcPriClassBase << "\t\t"; //输出该进程的优先级**

**cout << pe32.th32ProcessID << "\t\t"; //输出该进程的pid**

**cout << pe32.cntThreads << "\t\t"; //输出该进程的线程数**

**cout << pe32.szExeFile << endl; //输出进程名(或可执行文件名)**

**pn++; //进程数增1**

**} while (Process32Next(hProcessSnap, &pe32));//当快照中有下一个进程，继续循环**

**cout << pn << " 个进程\n" << endl;**

**}**

**else**

**cout << "\nProcess32First() failed:" << GetLastError();**

**CloseHandle(hProcessSnap); // 释放快照句柄**

**}**

**void add\_history(char \*inputcmd)// \*\*\*\*\* 将命令行插入历史队列中 \*\*\*\*\***

**{**

**envhis.end = (envhis.end + 1) % HISNUM;// end加1，准备将新的命令行插入队列**

**// end和start指向同一数组**

**if (envhis.end == envhis.start)//若队列满，则允许新插入的命令行覆盖旧命令行**

**envhis.start = (envhis.start + 1) % HISNUM; //调整队列头指针start(移一位)**

**// 将命令存入end指向的数组中**

**strcpy(envhis.his\_cmd[envhis.end], inputcmd);**

**}**

**// \*\*\*\*\*\*\*\*\*\*\*\*\*\* history 命令 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**void history\_cmd() // 显示历史命令**

**{**

**int i, j = 1;**

**if (envhis.end == envhis.start)**

**cout << "无历史命令\n"; //循环数组为空**

**else if (envhis.start < envhis.end)**

**{**

**//显示history命令数组中start+1到end的命令**

**for (i = envhis.start + 1; i <= envhis.end; i++)**

**{**

**cout << j << '\t' << envhis.his\_cmd[i] << endl;**

**j++;**

**}**

**}**

**else//否则，应分两段处理**

**{**

**// 显示history命令数组中start+1到HISNUM-1的命令**

**for (i = envhis.start + 1; i < HISNUM; i++)**

**{**

**cout << j << '\t' << envhis.his\_cmd[i] << endl;**

**j++;**

**}**

**// 显示history命令数组中0到end+1的命令**

**for (i = 0; i <= envhis.end + 1; i++)**

**{**

**cout << j << '\t' << envhis.his\_cmd[i] << endl;**

**j++;**

**}**

**}**

**cout << endl << endl;**

**}**

**// \*\*\*\*\*\*\*\*\*\*\*\*\*\* 创建进程命令 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**HANDLE process(int bg, char appName[]) //fp和bg&命令调用此函数**

**{**

**// 初始化进程相关信息**

**STARTUPINFO si; // 关于STARTUPINFO结构的成员的介绍，参看MSDN**

**PROCESS\_INFORMATION pi; //关于PROCESS\_INFORMATION结构的成员的介绍，参看MSDN**

**si.cb = sizeof(si); // si.cb的值应是STARTUPINFO结构体的大小(字节数)**

**GetStartupInfo(&si);//获得STARTUPINFO结构体，存于si中**

**ZeroMemory(&pi, sizeof(pi)); // 擦去pi的内容(其内存空间清零)**

**if (bg == 0) //前台进程**

**{**

**//设置钩子，捕捉组合键Ctrl+C命令，收到即结束进程**

**if (SetConsoleCtrlHandler((PHANDLER\_ROUTINE)ConsoleHandler, TRUE) == FALSE)**

**{**

**cout << "Unable to install handler!\n";**

**return NULL;**

**}**

**//用可执行文件appName创建前台进程。此函数各个参数的介绍请参看实验指导书**

**CreateProcess(NULL, appName, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi);**

**return pi.hProcess;**

**}**

**else//创建后台进程的要点是：①新进程另开窗口；②其窗口是隐藏的**

**{**

**//设置进程窗口选项**

**si.dwFlags = STARTF\_USESHOWWINDOW;**

**//若不指定此值，则将忽略下一语句的wShowWindow**

**si.wShowWindow = SW\_HIDE; //隐藏窗口**

**//创建后台进程，执行可执行文件appName**

**CreateProcess(NULL, appName, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi);**

**return NULL;**

**}**

**}**

**// \*\*\*\*\*\*\*\*\*\*\*\*\*\* taskkill命令(撤销进程) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**BOOL killProcess(char \*pid)**

**{**

**int id, i;**

**DWORD dwExitStatus;**

**HANDLE hprocess;**

**id = atoi(pid); // 将进程ID转换成整数**

**hprocess = OpenProcess(PROCESS\_TERMINATE, FALSE, id);//打开进程ID为id的进程，获得其句柄**

**GetExitCodeProcess(hprocess, &dwExitStatus);//根据刚获得的句柄，获取其退出码存于dwExitStatus**

**if (i = TerminateProcess(hprocess, dwExitStatus))//终止该进程**

**return TRUE; //若终止进程成功，返回TRUE**

**else**

**return FALSE;**

**}**

**// \*\*\*\*\*\*\*\*\*\*\*\*\* 显示帮助 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**void help()**

**{**

**cout << "\ncd：切换当前目录。\n";**

**cout << "输入形式：cd ..\n\t cd [drive:] [path] (cd c:\\temp)\n";**

**cout << "注：cd命令以空格为分隔符区分命令和参数。\n\n";**

**cout << "dir：显示目录中的文件和子目录列表。\n";**

**cout << "输入形式：dir\n\t dir [drive:] [path] (dir c:\\temp)\n";**

**cout << "注：dir命令以空格为分隔符区分命令和参数。\n\n";**

**cout << "tasklist：显示系统中当前进程的信息。\n";**

**cout << "输入形式：tasklist\n\n";**

**cout << "fp：创建进程并在前台执行。\n";**

**cout << "输入形式：fp\n\n";**

**cout << "bg&：创建进程并在后台执行。\n";**

**cout << "输入形式：bg&\n\n";**

**cout << "taskkill：终止进程。\n";**

**cout << "输入形式：taskkill [pid]\n";**

**cout << "注：taskkill命令以空格为分隔符，pid是进程id。\n\n";**

**cout << "history：显示历史命令。\n";**

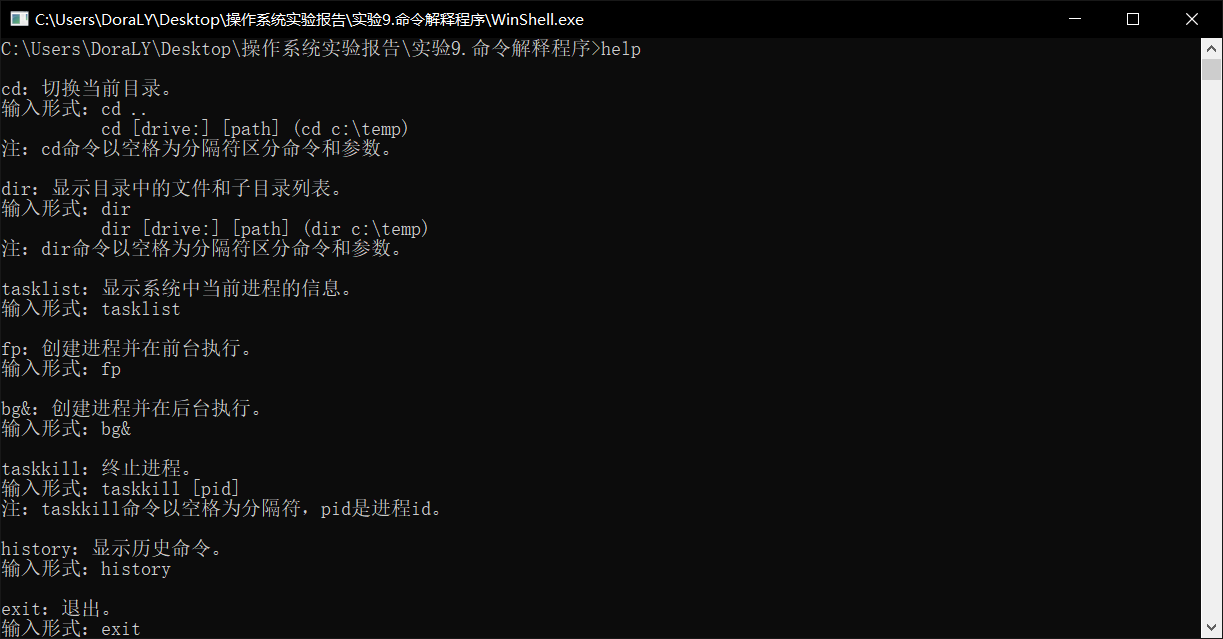
**cout << "输入形式：history\n\n";**

**cout << "exit：退出。\n";**

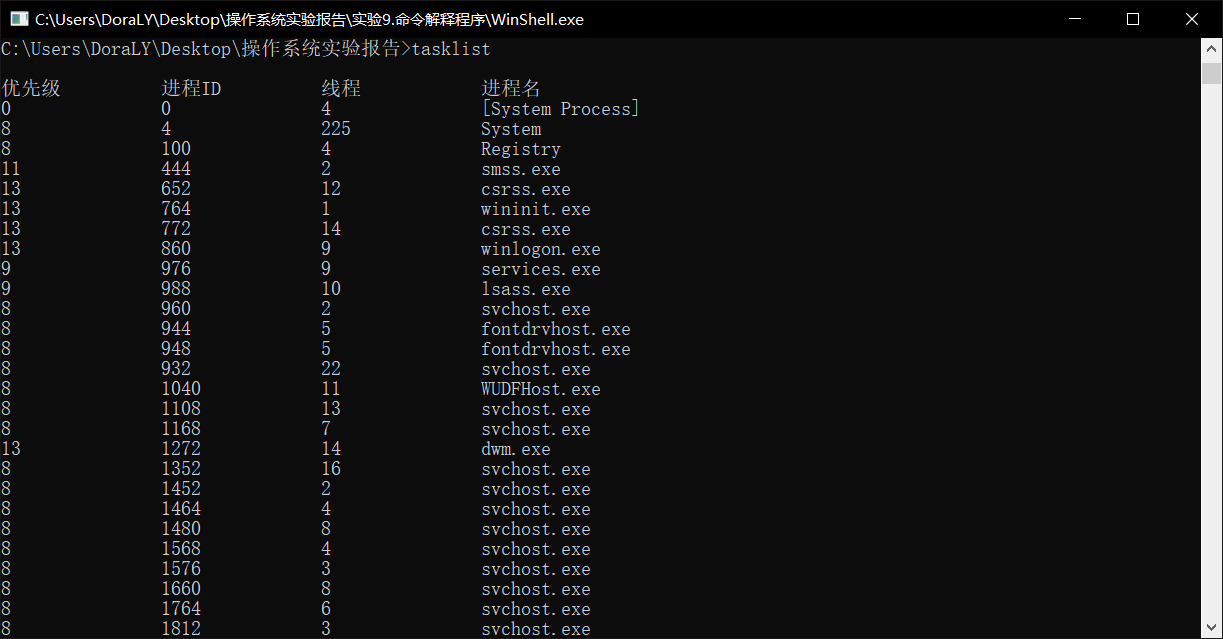
**cout << "输入形式：exit\n\n\n\n";**

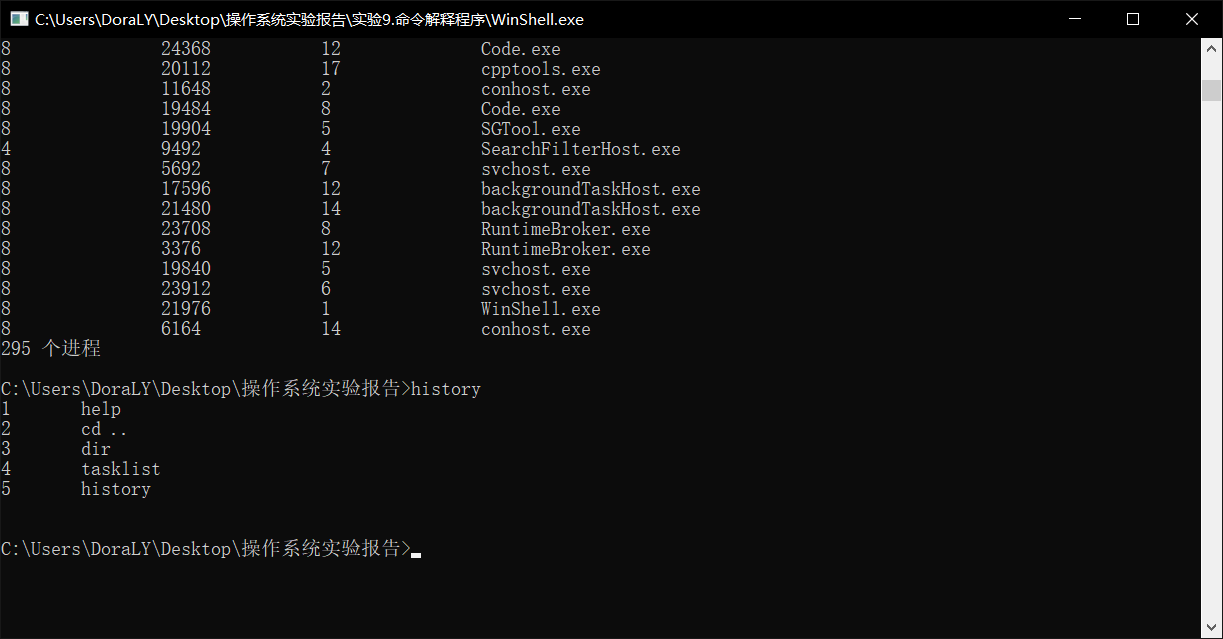
**}#pragma once**

1. **实验结果**









1. **实验总结：**

**本次实验主要是要求我们掌握命令解释程序的设计方法。通过调用相关的WIN32 API函数，我们就可以仿照MS-DOS的Command程序设计过程，来设计实验要求的命令解释程序。同时这次实验还要求我们了解目录操作和进程控制等相关知识。在这次实验中，我通过查询实验指导书，以及网上查找资料，理解了命令解释程序的相关重要知识，如命令解释程序与操作系统内核的关系以及对前台和后台进程的相关操作。和以往一样，在这次实验中，我也学习并掌握了不少新的API函数，比如GetCurrentDirectory函数、WaitForSingleObject函数和SetCurrentDirectory函数等等，这些函数功能不一，涉及到了目录操作、信号操作和文件操作等，由于指导书上有着对这些函数清晰的定义和解释，我能够更加轻松地看懂实验代码。在代码的修改和程序工程文件的创建过程中，我也渐渐地理解清楚了实验代码的大致核心要点，比如两个重要的数据结构，其一是一个循环数组，它通过指针的移动用来存放我们输入过的历史命令，其二则是一个信息链表，通过链式结构保存文件信息，我觉得这两个数据结构实在是太重要了，我也很庆幸能够通过实验来接触到它们，这不仅大大加深了我对指针知识的理解，还给我举例示范了链式结构的一种使用方法，对我的帮助很大。在实验过程中，出现了代码错误不能运行的情况，我通过询问同学和上网搜索报错原因，发现了问题所在并成功解决了问题，如在程序运行中strcpy要改成strcpy\_s这种问题。通过自己的努力发现问题并解决问题，我相信这对于我以后的代码生涯也是很宝贵的经历。总而言之，我在这次实验过程中学到了很多。**