第一章　文件和目录管理

几乎所有的应用程序都要用文件来保存数据和结果，以便以后使用。要有效地使用文件，首先应该理解文件是如何按照目录层次进行组织的以及 C 语言是如何操作文件的。本章简要地介绍 MS－DOS 操作系统内的目录和文件组织、文件和目录的命名以及对文件内容的解释。

另外，Microsoft Visual C＋＋完全实现了负责文件和目录管理任务的函数，这些管理任务包括：

（1）创建和删除目录。

（2）查找一个文件。

（3）检查文件的状态。

这一章也简要介绍了怎样使用这些函数。

通过阅读这一章，可以为使用第二章介绍的流和第三章介绍的低级输入/输出(I/O)函数作好准备，因为目录和文件管理任务通常先于文件 I/O。利用本章介绍的函数也可以查找指定的文件或检查某个文件是否存在。

本章后面的部分是文件和目录管理函数的参考部分。

1.1　MS－DOS 文件系统

与大多数操作系统(如 UNIX 及 DEC 公司的 VMS)一样，MS－DOS 用树状层次来组织文件和目录。硬盘或软盘上的所有文件及目录都出现在根目录(由反斜杠"\"指示)下。根目录是由 MS－DOS 在格式化软盘或硬盘分区(物理硬盘媒介中的分区)时建立的，它不需要由用户自己建立。每个目录都可以包含另外的目录和文件，树状层次就是这样形成的。

1.1.1　路径名

由于文件系统可能在软盘或硬盘上，所以 MS－DOS 要求用户指定一个驱动器符号，以便完全确定文件。实际上，文件的全名(全路径名)标识了驱动器和从根目录开始的所有目录。全路径名以带有冒号(:)的驱动器符号开始，后跟单反斜杠(\)以指明是根目录，然后将子目录名依次列下去，中间以反斜杠隔开，而文件名则出现在路径名的末尾。

在 MS－DOS 中，文件名最多由 8 个字符组成，后面可跟最多由 3 个字符组成的扩展名，中间以点号(.)隔开。Microsoft Visual C＋＋库中包含_fullpath，_makepath，splitpath 等函数，它们可对路径名进行操作。

程序员应注意非标准函数名的下划线前缀。

Microsoft Visual C＋＋在许多函数(如_access，_chdir 和 _chmod)名前加了一个下划线前缀，这是为了与 ANSI 标准 C 语言中的非标准函数命名约定保持一致。在其它许多 C 编译器(包括早期版本的 Microsoft C 编译器)以及 UNIX 与 POSIX 中，这些函数都可通过不带下划

线前缀的函数名来调用。注意，在 Microsoft Visual C++ 中也可使用不带下划线前缀的函数名。不带下划线前缀的函数定义于 oldnames.lib 库中，链接器在缺省方式下会自动查找这个库。

1. 当前驱动器

全路径名对于标识系统中的任意文件而言是必需的，但并不是在所有情况下都要提供全路径名。MS--DOS 标记了当前驱动器和当前工作目录。如果用户不指定驱动器名，那么 MS--DOS 就认为文件位于当前驱动器中。同样，如果不提供目录名，则假定文件位于当前工作目录中。

在 MS—DOS 中，一个驱动器是指一台设备，即一个软盘或硬盘驱动器。驱动器符号的常规含义如下：

(1) 驱动器 A：和 B：分别指第一个第二个软盘驱动器。

(2) 驱动器 C：～Z：标识硬盘驱动器，C：指第一个硬盘驱动器。

要在程序中确定当前驱动器或改变当前驱动器，可分别使用 _getdrive 和 _chdrive 函数。

2. 当前工作目录

正如 MS—DOS 保存着当前驱动器一样，它也对当前工作目录进行跟踪。在 C 程序中，通过调用 _getcwd 函数即可获取当前目录的全路径名。

可以用 DOS 命令 CHDIR 或 CD 来改变当前目录。若要创建新目录，则可使用 MKDIR 或 MD 命令，而 RD 命令则可删除某个空目录。Microsoft Visual C++ 提供了在程序中实现这些功能的 _chdir，_mkdir 及 _rmdir 函数。

1.1.2 作为文件的设备

虽然大多数文件是指保存在磁盘上的文件，但 DOS 中有些文件名却可以引用设备，如打印机或串行通信端口等。下面是一些这样的文件名：

(1) AUX 指第一个串行通信口。

(2) COM1 和 COM2 分别指第一个和第二个串行通信口。

(3) COM3 和 COM4 分别指第三个和第四个串行通信口。由于串行通信口要共享中断，所以在 PC 机上使用多个串行通信口时可能会出现问题。

(4) CON 指控制台，即键盘和屏幕。

(5) PRN 指第一个并行打印机端口。

(6) LPT1，LPT2 和 LPT3 分别指第一、第二和第三个并行打印机端口。

(7) NUL 指空设备，即当将输出信息发送给 NUL 时什么也不产生。

即使提供了扩展名，也不能将这些名字(不管是小写还是大写)用作基于磁盘的程序文件或数据文件的名称。它们只能用于访问设备。

1.1.3 文件属性

除了名字之外，DOS 文件还具有如下重要属性，它们会影响文件 I/O 操作：

(1) “权限设置”确定了文件的类型及其可访问性，即将文件操作限制为只读或读写。权

限设置是 DOS 的一部分。

(2) "转换模式"确定了在 I/O 操作过程中怎样解释文件的内容。转换是 C 语言的一部分。

1. 权限设置

文件的权限设置指出了允许对该文件所执行的操作。在 MS－DOS 中，可以把文件标识为只读文件或读写文件(MS－DOS 不支持只写文件，也不支持只执行文件)。可以用 chmod 函数来设置文件的权限。

函数_access 可以确定文件的访问类型。注意，当用 fopen 之类的函数打开文件时，必须用参数指定访问模式(希望打开文件的模式)。例如，可以用读写权限(只读、只写或读写)来打开文件。

2. 转换模式

当用 fopen 函数(请参阅第二章)打开文件时，除了可指定读写访问权限之外，还可指定怎样解释文件的内容。共有两种选择：

(1) 文本模式；

(2) 二进制模式。

在二进制模式下，文件中的每个字节都不加任何转换地提供给程序。在文本模式下，当程序读写文件时要进行下面的转换和解释：

(1) 当读文件时，一个回车换行对被转换成一个换行符，Ctrl+Z(0x1A)被解释成文件结束符(EOF)。

(2) 当写文件时，一个换行符被扩展成一个回车换行对并被写入文件中。

在打开文件时可以指定转换模式，也可以用_setmode 函数来改变已打开文件的转换模式。

1.1.4 文件句柄

尽管在 DOS 命令中文件名可能是标识文件最自然的方式，但在 C I/O 库中文件是通过其它方式标识的。与将要在第二章中看到的一样，当用 fopen 函数打开某个文件时，它将返回一个指向 FILE 结构的指针，而且以后的 I/O 操作均利用此指针来标识这个文件。类似地，如果用_open，_sopen 或_creat 函数创建或打开了一个文件(细节见第三章)，这些函数将返回一个称作文件句柄的整型文件标识符。后面介绍了几个用文件句柄作为参数的函数，例如_chsize，_filelength，_fstat，_isatty 和_locking 等。

1.2 基本文件和目录管理任务

表 1.1 是按字母顺序排列的，它提供了文件和目录管理函数的概况，而表 1.2 则按函数的功能进行组织。有关更多的信息，请参阅本章后面的部分。

表 1.1　文件和目录管理函数的字母顺序列表

函　数	描　　述
_access	检查文件是否存在及其读写权限设置。
_chdir	改变当前工作目录。
chdrive	改变当前驱动器。
_chmod	改变文件的读写权限设置。
_chsize	改变文件大小。
_filelength	返回文件的字节长度。
_fstat	提供一个用句柄标识的文件的有关信息。
_fullpath	把部分路径名转换成全路径名。
_getcwd	返回当前驱动器上的当前工作目录。
_getdcwd	返回指定驱动器上的当前工作目录。
_getdrive	返回一个标识着当前驱动器的整数(1＝A,2＝B,3＝C)。
_isatty	如果文件句柄指向字符设备,则返回非零值。
_locking	锁定或解锁文件中指定范围内的字节。
_makepath	根据 DOS 路径成分来建立 DOS 路径名。
_mkdir	创建一个新目录。
_mktemp	返回一个唯一的文件名。
remove	删除由路径名标识的文件。
rename	更改文件名。
_rmdir	删除一个目录。
_searchenv	在由环境变量指定的目录清单中查找某个文件。
_setmode	设置已打开文件的转换模式。
_splitpath	把 DOS 路径名分解成各组成部分。
stat	提供一个由文件名标识的文件的有关细节信息。
_umask	设置新打开的文件所用的权限屏蔽位。
_unlink	删除由路径名标识的文件。

表 1.2　基本的文件和目录管理任务

任　务	函　数
设置或检验文件的访问权限	access, _chmod, _umask
获取或设置当前驱动器	chdrive, _getdrive
管理目录	chdir, _getcwd, _getdcwd, _mkdir, _rmdir
定位文件	searchenv
获取或设置文件属性	chsize, _filelength, _fstat, _isatty, _mktemp, _setmode, stat
删除文件	remove, _unlink
更改文件名	rename
锁定文件中某一范围内的字节	locking
操纵 MS—DOS 路径名	fullpath, _makepath, _splitpath

1.2.1 改变驱动器和目录

有时，用户可能需要从程序中确定当前驱动器和当前工作目录。例如，假定要编写一个菜单驱动的实用程序，以便用户能够改变当前驱动器和当前目录。清单 1.1 给出了这样一个程序(dirutil.c)，它利用函数_getdrive，_getcwd，_chdrive 和 chdir 来完成所要求的任务。菜单驱动的用户界面是利用文本模式下的输出函数(见 graph.doc 文件)生成的。

注意，在大大扩展了的版本中，像 dirutil.c 这样的程序可以提供其它许多功能，如创建目录、删除目录、更改文件名或删除文件等。对于这样的程序，可以通过调用本章描述的函数来实现。

清单 1.1 用来改变当前驱动器或当前目录的菜单驱动程序 dirutil.c

```
//--------------------------------------------------------------
// File: dirutil.c
//
// A small utility program that lets the user change the
// current drive and the current directory
//--------------------------------------------------------------
#include <stdio.h>
#include <graph.h>
#include <direct.h>
#include <process.h>
#include <conio.h>
#include <ctype.h>

static int curdrive;
static char dirname[80],drivename[8];
//--------------------------------------------------------------
void main(void)
{
    int done = 0,c;
    char outbuf[80];

    while (! done)
    {
// Get current drive and directory for display
        curdrive = _getdrive();
        _getcwd(dirname,80);
        sprintf(drivename,"%c",curdrive + 'A' - 1);

        _clearscreen(_GCLEARSCREEN);
        _settextwindow(10,10,19,70);
        _settextposition(1,1);

// Display drive letter
        sprintf(outbuf,"Current Drive:    %s\n",drivename);
```

```
        outtext(outbuf);

// Display current directory
        sprintf(outbuf,"Current Directory:%s\n",dirname);
        outtext(outbuf);

        settextposition(5,1);
        outtext(" 1   Change Drive\n");
        outtext(" 2   Change Directory\n");
        outtext(" 0   Exit");

        settextposition(9,1);
        outtext("Enter selection:");

// Read keypress
        c = getche();

// Process command implied by keypress

        switch(c)
        {
              case '0':
                   exit(0);

              case '1':
// Change drive
                   _settextposition(9,1);
                   _outtext("Enter drive letter:");
                   c = getche();
                   _chdrive(toupper(c) - 'A' + 1);
                   break;

              case '2':
// Change current working directory
                   _settextposition(9,1);
                   _outtext("Enter directory name:");
                   gets(dirname);
                   if(_chdir(dirname) != 0)
                        _outtext("\nError changing directory!");
                   break;
        }
    }
}
```

要建立这个 dirutil 程序，可以用 Visual Workbench 创建一个工程，此工程应该是一个 DOS 可执行程序(不需要包含 MFC 支持，但如果已经建立了 DOS MFC 库，那么包含 MFC 也不会发生错误)。在工程中包含 dirutil.c 程序。必须使此程序为 DOS 可执行程序；如果使此程序为 QuickWin 程序，它也能运行，但是用户将看不到运行结果。

当运行 dirutil 程序时将显示如下画面：

```
Current Drive:    D
Current Directory:D:\MCPB\EX\C\CH1

1  Change Drive
2  Change Directory
0  Exit

Enter selection:0
```

可以通过按“0”键退出此程序。选项 1 和选项 2 可以改变当前驱动器和当前目录。当任何一项被改变后，程序将更新在菜单上显示的当前驱动器和目录名。

1.2.2 改变文件属性

作为使用文件和目录管理函数的另一个例子，考虑如何使文件变成只读文件以防意外删除或覆盖。清单 1.2 给出了程序 protect.c，此程序利用_chmod 函数使文件变为只读文件。一旦编译并建立了 PROTECT.EXE 文件，就可以用下列命令使某个文件(此例中是 protect.obj)变为只读文件：

```
protect   protect.obj
```

清单 1.2　使文件变成只读文件以防意外删除的程序 protect.c

```
//-----------------------------------------------------------
// File:protect.c
//
// Mark a file read-only so that it cannot be erase or overwritten
//
// Use it as:protect filename
//-----------------------------------------------------------
#include <stdio.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
//-----------------------------------------------------------
void main(int argc,char * * argv)
{
    if(argc < 2)
    {
        printf("Usage:%s <pathname>\n",argv[0]);
    }
    else
    {
        if(_chmod(argv[1],S_IREAD) == -1)
            perror("Error in chmod");
```

```
        else
            printf("%s protected\n",argv[1]);
    }
;
```

一旦某个文件被 PROTECT.EXE“保护”起来，只要用户想删除此文件，就会看到以下错误信息：

```
Access denied
```

要恢复保护前的状态，需要另外一个叫作 unprot 的应用程序，它允许重新对文件执行读写操作。清单 1.3 给出了完成此任务的程序 unprot.c。

清单 1.3　允许对文件进行读写的程序 unprot.c

```
//-----------------------------------------------
// File:unprot.c
//
// Mark a file as read/write to reverse the effects of "protect"
//
// Use it as:unprot filename
//-----------------------------------------------
#include <stdio.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
//-----------------------------------------------
void main(int argc,char * *argv)
{
    if(argc < 2)
    {
        printf("Usage:%s <pathname>\n",argv[0]);
    }
    else
    {
        if(_chmod(argv[1],S_IREAD|S_IWRITE) == -1)
            perror("Error in chmod");
        else
            printf("%s unprotected\n",argv[1]);
    }
}
```

1.3 函数参考

1.3.1 _access

MSC6	MSC7	VC++	QC2.5	QC—WIN	TC2	TC++	BC++2	BC++3	ANSI	POSIX	UNIX V	DOS	QWIN	WIN	WINDLL
×	×	×	×	×	×	×	×	×		×	×	×	×	×	×

关于函数名开头的下划线，见本章前面的内容。

概要

```
#include <io.h>
int access(
    const char  *name,      // 要检查的文件的路径名
    int         mode);      // 文件是否可用此模式访问
```

说明

检查某个文件(由 name 标识)是否存在，是否可用指定访问模式进行访问。mode 参数可以为下列值之一：

00　检查文件是否存在。

02　检查是否允许写文件。

04　检查是否允许读文件。

06　检查是否允许读写文件。

返回值

若返回 0，则所要求的访问是允许的。

若返回 −1，则指示一个错误。设置 errno 为 EACCES 以指示访问被拒绝。如果指定的文件不存在，则将 errno 设置成 ENOENT。

调用实例

```
if(_access("invoice.def",0) == 0) puts("File exists");
```

参见

chmod，_fstat，_stat

1.3.2 _chdir

MSC6	MSC7	VC++	QC2.5	QC—WIN	TC2	TC++	BC++2	BC++3	ANSI	POSIX	UNIX V	DOS	QWIN	WIN	WINDLL
×	×	×	×	×	×	×	×	×		×	×	×	×	×	×

关于函数名开头的下划线，见本章前面的内容。

概要

```
#include <direct.h>
```

```
int chdir (const char * new_path);      // 新目录名
```

说明

函数_chdir 可将当前驱动器上的当前工作目录改为由 new_path 指定的目录。

调用实例

```
// 改变到目录 D:\MSVC\MFC\DOC
if (_chdir("D:\\MSVC\\MFC\\DOC") != 0)
    perror("_chdir");
```

返回值

0　　成功地改变了当前工作目录。

-1　　指示出错，并将 errno 置为 ENOENT。

参见

mkdir，_rmdir

1.3.3 _chdirve

MSC6	MSC7	VC++	QC2.5	QC-WIN	TC2	TC++	BC++2	BC++3	ANSI	POSIX	UNIX V	DOS	QWIN	WIN	WINDLL
×	×	×	×	×				×				×	×	×	×

概要

```
#include <direct.h>

int chdirve(
        int drive_num);        // 驱动器，1=A，2=B，3=C，4=D . . .
```

说明

将当前驱动器改为由整型变量 drive_num 指定的驱动器。

返回值

0　　成功地改变了驱动器。

-1　　指示出错。

调用实例

```
// 改变为 E:驱动器
int drive_letter = 'e';
// 使 drive_letter 为大写
drive_letter = toupper(drive_letter);
if (_chdrive(drive_letter - 'A' + 1) != 0)
{
    fprintf(stderr,"Error changing drive! \n");
}
```

参见

dos_setdrive(第十二章)，_getdrive

1.3.4 _chmod

MSC6	MSC7	VC++	QC2.5	QC—WIN	TC2	TC++	BC++2	BC++3	ANSI	POSIX	UNIX V	DOS	QWIN	WIN	WINDLL
√	√	√	√	√	√	√	√	√		√	√	√	√	√	√

关于函数名开头的下划线，参见本章前面的内容。

概要

```
#include <io.h>                    // 函数原型
#include <sys\types.h>             // 下一个头文件需要此文件
#include <sys\stat.h>              // 权限设置(S_IREAD 等)

int _chmod(
    const char  *path,     // 将此文件的权限设置
    int         mode);     // 改为此模式
```

说明

将文件的权限设置成由 mode 指定的值。mode 可为下列值之一：

_S_IWRITE	读写均可(因为 MS—DOS 不允许只写文件)。
_S_IREAD	只读。
_S_IREAD | S_IWRITE	读写均可。

返回值

0	成功地改变了权限。
—1	指示出错。将 errno 置为 ENOENT 以指示文件不存在。

调用实例

```
// 使 index 文件成为只读文件
chmod("index.dat",_S_IREAD);
```

参见

access，_fstat，stat

1.3.5 _chsize

MSC6	MSC7	VC++	QC2.5	QC—WIN	TC2	TC++	BC++2	BC++3	ANSI	POSIX	UNIX V	DOS	QWIN	WIN	WINDLL
√	√	√	√	√	√	√	√	√			√	√	√	√	√

关于此函数名前下划线的含义，请参阅本章前面的有关注释。

概要

```
#include <io.h>
int chsize (
    int handle,          // 需要改变长度的已打开文件的句柄
    long size );         // 文件的新字节长度
```

说明

将文件的新长度置为 size 个字节(此文件由句柄标识)。如果文件被扩展,则空字符被加到文件尾。

返回值

若返回 0,则表示文件长度修改成功。

若返回 -1,则表明出现错误,并将 errno 置为 EACCES,EBADF 或 ENOSPC,分别表示无此访问权限、文件句柄不合法、磁盘空间不够等错误。

调用实例

```
// 把一个已打开的文件截为零字节长度
if (_chsize(fhandle,0L) == -1)
    perror("_chsize");
```

参见

fileno(第二章),_close(第三章),_creat(第三章),_open(第三章)

1.3.6 filelength

MSC6	MSC7	VC++	QC2.5	QC-WIN	TC2	TC++	BC++2	BC++3	ANSI	POSIX	UNIX V	DOS	QWIN	WIN	WINDLL
×	×	×	×	×	×	×	×	×				×	×	×	×

关于此函数名前下划线的含义,请参阅本章前面的有关注释。

概要

```
#include <io.h>
long filelength(int handle);   // 已打开文件的句柄
```

说明

返回 handle 所指的已打开文件的长度。

返回值

若该函数调用成功,则得到一个关于文件字节长度的长整型值。

调用实例

```
// 获得文件长度
if ((filesize = filelength(fhandle)) == -1)
    perror("_filelength");
```

参见

chsize，_fileno，_fstat，_stat

1.3.7 _fstat

MSC6	MSC7	VC++	QC2.5	QC—WIN	TC2	TC++	BC++2	BC++3	ANSI	POSIX	UNIX V	DOS	QWIN	WIN	WINDLL
×	×	×	×	• ×	×	×	×	×		×	×	×	×	×	×

关于此函数名前下划线的含义，请参阅本章前面的有关注释。

概要

```
#include <sys\types.h>                    // 由<sys\stat.h>使用
#include <sys\stat.h>                     // 函数原型

int fstat(
    int    handle,                        // 返回有关该文件的信息
    struct _stat *info);                  // 信息返回处
```

说明

将由 handle 所标识的文件的有关信息拷贝到_stat 结构类型的变量 info 中。结构类型_stat 定义于头文件 sys\stat.h 中，描述如下：

```
struct stat                               // 在 UNIX 和 POSIX 中定义结构 stat
{
    dev_t          st_dev;                // 驱动器号或句柄
    ino_t          st_ino;                // 不能在 MS—DOS 中使用
    unigned short  st_mode;               // 文件模式
    short          st_nlink;              // MS—DOS 环境下置为 1
    short          st_uid;                // 不能在 MS—DOS 中使用
    short          st_gid;                // 不能在 MS—DOS 中使用
    dev_t          st_rdev;               // 驱动器号或句柄
    off_t          st_size;               // 文件的字节长度
    time_t         st_atime;              // 上一次访问的时间
    time_t         st_mtime;              // 上一次修改的时间
    time_t         st_ctime;              // 建立的时间
};
```

注意，这个结构类型在其它编译器和 UNIX 及 POSIX 中被定义成 struct stat(无前导下划线)。

返回值

0　　信息拷贝成功。

—1　　表明出现错误，且 errno 被置为 EBADF，表示文件句柄不合法。

调用实例

```
struct _stat info;
```

```
// 得到关于文件流 stdout 的信息
if (_fstat(fileno(stdout),&info) != 0)   perror("_fstat");
```

参见

access, _stat

1.3.8 _fullpath

MSC6	MSC7	VC++	QC2.5	QC—WIN	TC2	TC++	BC++2	BC++3	ANSI	POSIX	UNIX V	DOS	QWIN	WIN	WINDLL
×	×	×	×	×				×				×	×	×	×

概要

```
#include <stdlib.h>
char * fullpath (
    char          * buf,          // 接收全路径名的缓冲区
    const char    * path,         // 将被转换成全路径的路径名
    size_t        nbytes );       // 缓冲区 buf 的长度
```

说明

把一个相对路径名转换成相应的全路径名(包括驱动器和目录名),并把结果拷贝到一个指定的缓冲区 buf 中。参数 nbytes 指定了缓冲区的字节长度。该函数能处理包含".\"和"..\"的路径名,而_makepath 函数则不能处理这样的路径名。

如果 buf 为 NULL,则_fullpath 函数将自动分配一个长度为_MAX_PATH 的缓冲区来存放结果路径名,且用户在不再使用时应释放此缓冲区。

返回值

当函数调用成功时,返回值是一个指向包含全路径名的缓冲区的指针。

NULL 表示函数调用失败。

调用实例

```
char   fpname[_MAX_PATH];
// 将路径名..\include 扩展成全路径名
if (_fullpath(fpname,"..\\include",_MAX_PATH) != NULL)
    printf("Full pathname is:%s\n",fpname);
```

参见

getcwd, _makepath, _splitpath

1.3.9 _getcwd

MSC6	MSC7	VC++	QC2.5	QC—WIN	TC2	TC++	BC++2	BC++3	ANSI	POSIX	UNIX V	DOS	QWIN	WIN	WINDLL
×	×	×	×	×	×	×	×	×		×	×	×	×	×	×

关于此函数名前下划线的含义,请参阅本章前面的有关注释。

概要

```
#include  <direct.h>
char  *getcwd(
    char  *pbuf,              // 返回当前目录名
    int   maxchar);           // 缓冲区长度
```

说明

将当前工作的目录路径名放入 pbuf 中。如果 pbuf 为 NULL，则_getcwd 函数将通过调用 malloc 函数来分配一个长度为 maxchar 的缓冲区，以存放目录路径名。

返回值

当该函数调用成功时，返回值为指向存储结果路径名的缓冲区的指针。如果 pbuf 为 NULL，则返回值为指向_getcwd 所分配的缓冲区的指针。通过调用 free 函数可以释放这块内存。

若返回 NULL，则表示函数调用出错，errno 被置为 ENOMEM 或 ERANGE，分别表示用来分配缓冲区的内存不够或者路径名太长等错误。

调用实例

```
char buffer[_MAX_DIR];          // MAX_DIR 在 stdlib.h 中定义
// ...
if (_getcwd(buffer,_MAX_DIR) == NULL)  perror("_getcwd");
```

参见

dos_getdrive(第十二章)，_chdir，_getdcwd

1.3.10 _getdcwd

MSC6	MSC7	VC++	QC2.5	QC—WIN	TC2	TC++	BC++2	BC++3	ANSI	POSIX	UNIX V	DOS	QWIN	WIN	WINDLL
×	×	×	×	×				×				×	×	×	×

概要

```
#include <direct.h>

char  *_getdcwd (
    int   drive_num,          // 驱动器：1=A，2=B，3=C 等
    char  *cwd_name,          // 子目录名返回的缓冲区
    int   maxlen );           // 缓冲区 cdw_name 的长度
```

说明

将指定驱动器上的当前工作路径名拷贝到缓冲区 cwd_name 中。如果 cwd_name 为 NULL，则_getdcwd 函数通过调用 malloc 函数来分配长度为 maxlen 个字符的缓冲区，以存放此路径名。

返回值

当该函数调用成功时，返回一个指向存储当前目录的缓冲区的指针。如果 cwd_name 为 NULL，则指针将指向_getdcwd 函数所分配的缓冲区。通过调用 free 函数可以释放这块内存。

若返回 NULL，则表示调用出错，errno 被置为 ENOMEM 或 ERANGE，分别表示用来分配新缓冲区的内存不够或者路径名太长等错误。

调用实例

```
char buffer[_MAX_PATH]      // MAX_PATH 在 stdlib.h 中定义

// 取得 C:驱动器上的当前工作目录
if (_getdcwd(3,buffer,_MAX_PATH) == NULL)  perror("_getdcwd");
```

参见

dos_getdrive(第十二章)，_chdir，_getcwd，_getdrive

1.3.11 _getdrive

MSC6	MSC7	VC++	QC2.5	QC-WIN	TC2	TC++	BC++2	BC++3	ANSI	POSIX	UNIX V	DOS	QWIN	WIN	WINDLL
×	×	×	×	×			×	×				×	×	×	×

概要

```
#include <direct.h>

int getdrive(void);
```

说明

返回当前磁盘驱动器的标识符。

返回值

成功时返回标识当前磁盘驱动器的整数值：

1　　标识 A：驱动器。

2　　标识 B：驱动器。

...

调用实例

```
// 在更换驱动器之前保存当前驱动器
saved_drive_num = getdrive();
```

参见

dos_getdrive(第十二章)，_dos_setdrive(第十二章)，_chdrive，_getcwd，_getdcwd

1.3.12 _isatty

MSC6	MSC7	VC++	QC2.5	QC—WIN	TC2	TC++	BC++2	BC++3	ANSI	POSIX	UNIX V	DOS	QWIN	WIN	WINDLL
×	×	×	×	×	×	×	×	×		×	×	×	×	×	×

关于此函数名前下划线的含义，请参阅本章前面的有关注释。

概要

```
#include <io.h>

int _isatty(int handle);      // 文件是否为字符设备
```

说明

确定 handle 所指的文件是否为一个字符设备，如控制台、打印机或串行端口等。

返回值

非零　　表明文件是字符设备。

零　　表明不是字符设备。

调用实例

```
// 检查 stdout 并不定向到某个文件
if (_isatty(fileno(stdout)))
    puts("Interactive mode (stdout tied to console)");
```

参见

fstat，_stat

1.3.13 _locking

MSC6	MSC7	VC++	QC2.5	QC—WIN	TC2	TC++	BC++2	BC++3	ANSI	POSIX	UNIX V	DOS	QWIN	WIN	WINDLL
×	×	×	×	×				×			×	×	×	×	×

关于此函数名前下划线的含义，请参阅本章前面的有关注释。

注意，在 Borland 编译器的早期版本中，其等价函数为 lock 和 unlock。

概要

```
#include <sys\locking.h>      // LK_constants 的定义
#include <io.h>

int locking (
    int    handle,            // 已打开文件的句柄
    int    lmode,             // 锁定模式
    long   nbytes);           // 锁定或解锁的字节数
```