实验五 复制文件

徐恒达 1120151811

目录

1 实验目的

- 通过对 Windows / Linux 编程，进一步熟悉操作系统的基本概念，较好地理解 Windows /Linux 的系统结构和编程特点。
- 深入了解 Windows 和 Linux 系统下的文件操作的相关的系统 API 的功能和作用，并可以用 API 的组合完成一些常见的文件操作。

2 实验内容

完成一个目录复制命令 mycp，包括目录下的文件和子目录。要求分别在 Windows 和 Linux 平台上完成。运行结果如下。

```
beta@bugs.com [~/]# ls -l sem
total 56
drwxr-xr-x 3 beta beta 4096 Dec 19 02:53 ./
drwxr-xr-x 8 beta beta 4096 Nov 27 08:49 ../
-rw-r--r-- 1 beta beta 128 Nov 27 09:31 Makefile
-rwxr-xr-x 1 beta beta 5705 Nov 27 08:50 consumer*
-rw-r--r-- 1 beta beta 349 Nov 27 09:30 consumer.c
drwxr-xr-x 2 beta beta 4096 Dec 19 02:53 subdir/
beta@bugs.com [~/]# mycp sem target
beta@bugs.com [~/]# ls -l target
total 56
drwxr-xr-x 3 beta beta 4096 Dec 19 02:53 ./
drwxr-xr-x 8 beta beta 4096 Nov 27 08:49 ../
-rw-r--r-- 1 beta beta 128 Nov 27 09:31 Makefile
-rwxr-xr-x 1 beta beta 5705 Nov 27 08:50 consumer*
-rw-r--r-- 1 beta beta 349 Nov 27 09:30 consumer.c
drwxr-xr-x 2 beta beta 4096 Dec 19 02:53 subdir/
```

在 Linux 下可使用的系统调用有 mkdir、opendir、readdir、symlink、readlink 等。

在 Windows 下可使用的系统调用有 CreateDirectory、FindFirstFile、FindNextFile 等。

3 实验环境

3.1 硬件配置

处理器：Intel(R) Core(TM) i3-4020Y CPU @ 1.5GHz
内存：4.0GB DDR3

3.2 Windows 配置

Windows 10 Pro Version 1709 64bit

3.3 Linux 配置

Ubuntu 16.04 LTS

4 实验过程

4.1 实现思路

拷贝文件夹的过程就是一个递归遍历目录树的过程，遍历一棵树同时产生另一棵完全相同的树。采用深度优先遍历 DFS 进行目录拷贝的过程如伪代码中所示。

拷贝函数 mycp 接受两个参数，第一个是源路径，表示要拷贝的文件或文件夹路径，第二个是目标路径，表示要拷贝到的位置。

mycp 是一个递归函数，递归遍历整棵目录树，一个文件夹表示树上的一个非叶节点，一个文件代表一个页节点。mycp 函数检查传入的原路径，如果原路径对应的是一个文件，说明到了叶节点，就直接拷贝到目标路径，如果是一个文件夹，就在先目标路径位置创建一个同名文件夹，然后打开源文件夹，遍历其中的每一个元素（文件和文件夹），对每一个元素调用 mycp 函数递归处理，直至所有操所全部完成。

Algorithm 1 Mycp

Require: mycp(源路径，目标路径)

```
1: if 原路径是文件 then
2:     拷贝文件 (源路径，目标路径)
3: else if 源路径是文件夹 then
4:     在目标路径创建文件夹
5:     for all 文件 in 源文件夹 do
6:         mycp(原路径 + 文件名, 目标路径 + 文件名)
7:     end for
8: end if
```

4.2 Windows 实现

在 Windows 中主要使用这几个函数实现操作，FindFirstFile() 和 FindNextFile() 函数用来遍历目录下的所有内容，读取每个文件的属性，也用来读取某个特定文件的属性；CreateDirectory() 创建文件夹，CopyFile() 拷贝文件。下面对这几个函数的用法作简单说明。

4.2.1 FindFirstFile 函数

```
HANDLE WINAPI FindFirstFile(
  _In_ LPCTSTR        lpFileName,
  _Out_ LPWIN32_FIND_DATA lpFindFileData
);
```

参数 lpFileName 是要查找的内容，可是一个确切的字符串路径，也可以是带有通配符的路径，比如如果要查找 folder 文件夹下的所有内容就可以传入"folder*"。

参数 lpFindFileData 是一个指向WIN32_FIND_DATA结构体的指针，函数将找到的一个符合要求的文件或文件夹的属性信息放到结构体中，结构体的定义如下。

```
typedef struct _WIN32_FIND_DATA {
  DWORD   dwFileAttributes;
  FILETIME ftCreationTime;
  FILETIME ftLastAccessTime;
  FILETIME ftLastWriteTime;
  DWORD   nFileSizeHigh;
  DWORD   nFileSizeLow;
  DWORD   dwReserved0;
  DWORD   dwReserved1;
  TCHAR   cFileName[MAX_PATH];
  TCHAR   cAlternateFileName[14];
} WIN32_FIND_DATA;
```

字段 dwFileAttributes 标明了文件的属性，是一个属性位图，不同的二进制位标识不同的属性，同时对不同的属性定义了各自的常量表示符，可以用不同的属性常量与dwFileAttributes作按位与操作来得知文件是否具有此属性。比如，如果要判读找到的是不是一个文件夹，就可以这样判断ffd.dwFileAttributes&FILE_ATTRIBUTE_DIRECTORY。

字段cFileName[MAX_PATH]是一个字符数组，存储了文件名。

4.2.2 FindNextFile 函数

FindFirstFile() 函数可以得到符合查找条件的第一个文件信息，如果是要确切获得一个文件的属性，FindFirstFile() 函数已经足够了，但若是要查找多个文件，比如遍历目录下的所有文件，就需要使用 FindNextFile() 函数获取后面的文件信息。函数声明如下。

```
BOOL WINAPI FindNextFile(
  _In_ HANDLE         hFindFile,
  _Out_ LPWIN32_FIND_DATA lpFindFileData
);
```

传入调用 FindFirstFile() 后返回的 FindFile 句柄，FindNextFile() 函数读取下一个文件信息，写到WIN32_FIND_DATA结构体中，不断调用 FindNextFile() 函数，每次函数都将下一个文件信息写到结构体中并返回 TRUE，全部遍历完毕返回 FALSE。

4.2.3 CreateDirectory 函数

```
BOOL WINAPI CreateDirectory(
  _In_    LPCTSTR            lpPathName,
  _In_opt_ LPSECURITY_ATTRIBUTES lpSecurityAttributes
);
```

函数 CreateDirectory() 在 lpPathName 位置创建一个空文件夹，lpSecurityAttributes 参数标明了文件夹的安全属性，传入 NULL 表示使用默认属性。

4.2.4 CopyFile 函数

```
BOOL WINAPI CopyFile(
  _In_ LPCTSTR lpExistingFileName,
  _In_ LPCTSTR lpNewFileName,
  _In_ BOOL  bFailIfExists
);
```

CopyFile() 函数将 lpExistingFileName 指向的文件拷贝到 lpNewFileName，如果 bFailIfExists 参数为 TRUE，则如果目标位置已经存在文件时拷贝失败，FALSE 表示覆盖已有文件。

4.2.5 完整实现

核心拷贝函数如下。实现时需要注意两个特殊文件，每个文件夹下都会有两个文件，名为. 和..，分别表示当前文件夹和父文件夹引用，对这两个文件跳过不处理。

```
void Copy(const char srcPath[], const char dstPath[])
{
    WIN32_FIND_DATA ffd;
    FindFirstFile(srcPath, &ffd);
    if (!(ffd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)) {
        CopyFile(srcPath, dstPath, FALSE);
        return;
    }

    CreateDirectory(dstPath, NULL);
    char dir[MAX_PATH];
    sprintf(dir, "%s\\*", srcPath);
    HANDLE hFind = FindFirstFile(dir, &ffd);
    for (BOOL f = TRUE; f; f = FindNextFile(hFind, &ffd))
    {
        char *name = ffd.cFileName;
        if (strcmp(name, ".") == 0) continue;
        if (strcmp(name, "..") == 0) continue;
        char srcSubPath[MAX_PATH];
        sprintf(srcSubPath, "%s\\%s", srcPath, ffd.cFileName);
        char dstSubPath[MAX_PATH];
        sprintf(dstSubPath, "%s\\%s", dstPath, ffd.cFileName);
        Copy(srcSubPath, dstSubPath);
    }
}
```

4.3 Linux 实现

在 Linux 下实现文件拷贝操作主要用到这几个函数：lstat，readlink，symlink，mkdir，opendir，readdir。下面分别对它们作简要介绍。

4.3.1 lstat 函数

```
int lstat(const char *path, struct stat *buf);
```

函数获取 path 路径所指向文件的状态，写在 buf 指针所指向的 struct stat 结构体中。如果文件是一个符号链接文件，lstat 函数返回符号链接文件本身的信息，而不是返回所指向的文件的信息。

结构体 stat 的定义如下。

```
struct stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    mode_t    st_mode;    /* protection */
    nlink_t   st_nlink;   /* number of hard links */
    uid_t     st_uid;     /* user ID of owner */
    gid_t     st_gid;     /* group ID of owner */
    dev_t     st_rdev;    /* device ID (if special file) */
    off_t     st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system I/O */
    blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
    time_t    st_atime;   /* time of last access */
    time_t    st_mtime;   /* time of last modification */
    time_t    st_ctime;   /* time of last status change */
};
```

字段st_mode表明了文件的属性信息，同时提供了若干宏定义函数来判断文件是否具有某一属性，比如用S_ISLNK(statbuf.st_mode)判断文件是否是一个符号链接，使用S_ISDIR(statbuf.st_mode)判断是否是一个文件夹等。

4.3.2 readlink 函数

```
ssize_t readlink(const char *path, char *buf, size_t bufsiz);
```

readlinke() 函数读取符号链接文件的内容，将其写在 buf 所指向的内存区域中，bufsiz 是 buf 区域的大小。

4.3.3 symlink 函数

```
int symlink(const char *path1, const char *path2);
```

symlink() 函数创建一个名为 path2 的符号链接文件，并将 path1 指向区域中的符号链接信息写到文件中。

有了这三个函数，现在就可以实现符号链接文件的拷贝过程，代码如下。

```
struct stat statbuf;
lstat(src, &statbuf);
if (S_ISLNK(statbuf.st_mode)) {
  char buf[50];
  readlink(src, buf, sizeof(buf));
  symlink(buf, dst);
}
```

4.3.4 mkdir 函数

```
int mkdir(const char *pathname, mode_t mode);
```

mkdir() 函数在 pathname 位置创建文件夹，mode 参数给出文件夹的权限信息。

4.3.5 opendir 函数

```
DIR *opendir(const char *name);
```

opendir() 函数打开 name 所指向的文件夹，返回一个指向 DIR 目录流类型的指针，只有可以使用这个指针读取目录中的信息。

4.3.6 readdir 函数

```
struct dirent *readdir(DIR *dirp);
```

readdir() 函数的 dirp 参数是要读取的目录的 DIR 目录流指针，每调用一次 readdir()，函数返回一个 dirent 结构体指针，可以从 dirent 结构体中读取目录中每个文件的信息，全部读取完毕返回 NULL。

结构体 dirent 的定义如下。

```
struct dirent {
    ino_t        d_ino;      /* inode number */
    off_t        d_off;      /* offset to the next dirent */
    unsigned short d_reclen; /* length of this record */
    unsigned char d_type;    /* type of file; not supported
```

```
                                by all file system types */
    char            d_name[256]; /* filename */
};
```

其中d_type字段标明了文件类型，d_name[256]字段存储了文件名。

4.3.7 完整实现

有了这些函数，就可以实现递归拷贝目录的功能。实现代码如下。

```
void copy(char *src, char *dst)
{
    struct stat statbuf;
    lstat(src, &statbuf);
    if (S_ISLNK(statbuf.st_mode)) {
        char buf[50];
        readlink(src, buf, sizeof(buf));
        symlink(buf, dst);
        return;
    }
    if (!S_ISDIR(statbuf.st_mode)) {
        copyfile(src, dst);
        return;
    }
    mkdir(dst, statbuf.st_mode);
    DIR *pdir = opendir(src);
    struct dirent *pdirent;
    while ((pdirent = readdir(pdir)) != NULL) {
        char *name = pdirent->d_name;
        if (strcmp(name, ".") == 0) continue;
        if (strcmp(name, "..") == 0) continue;
        char srcsub[MAXNAMLEN];
        sprintf(srcsub, "%s/%s", src, name);
        char dstsub[MAXNAMLEN];
        sprintf(dstsub, "%s/%s", dst, name);
        copy(srcsub, dstsub);
    }
}
```

由于 Linux 中没有直接拷贝文件的系统函数，所以要自己实现拷贝文件功能。过程是打开源文件，创建目标文件，建立一个缓冲区，不断从源文件中读出数据到缓冲区，然后将缓冲区中的数据写到目标文件中，找到所有数据拷贝完成。实现代码如下。

```
void copyfile(char *srcfile, char *dstfile) {
    int srcfd = open(srcfile, O_RDONLY);
    int dstfd = open(dstfile, O_WRONLY | O_CREAT, 0666);
    char buf[1024];
    ssize_t bytes;
    while((bytes = read(srcfd, buf, sizeof(buf))) > 0) {
        write(dstfd, buf, bytes);
    }
    close(srcfd);
    close(dstfd);
}
```

5 实验结果

5.1 编译

Windows 下和 Linux 下编译的 makefile 文件如下所示，编译出的可执行文件名为 mycp。

```
mycp: mycp.c
    gcc mycp.c -o mycp
```

5.2 Windows 实验结果

5.2.1 目录拷贝

拷贝之前使用 tree 命令查看当前路径下的目录结构，可以看到有一个名为 src 的文件夹，其中有一个文件 a.txt 和一个子目录 sub，sub 中有一个文件 b.txt，我们打算拷贝这个文件夹。

```
> tree /f
Folder PATH listing
Volume serial number is B80C-E914
C:.
```

```
|   makefile
|   mycp.c
|   mycp.exe
|
|---src
    |   a.txt
    |
    |---sub
            b.txt
```

运行 mycp 命令，将 src 文件夹拷贝为 dst 文件夹，命令行参数如下。

```
> mycp src dst
```

再次调用 tree 命令，发现已经有了 dst 目录，结构与 src 目录完全相同，拷贝成功。

```
> tree /f
Folder PATH listing
Volume serial number is B80C-E914
C:.
|   makefile
|   mycp.c
|   mycp.exe
|
|---dst
|   |   a.txt
|   |
|   |---sub
|           b.txt
|
|---src
    |   a.txt
    |
    |---sub
            b.txt
```

5.2.2 文件拷贝

如果给定 mycp 的源路径参数是文件时，mycp 就拷贝文件。例如将源程序文件 mycp.c 拷贝到当前目录下并命名为 mycp2.c 时，输入命令行参数如下。

```
> mycp mycp.c mycp2.c
```

查看目录，发现有了 mycp2.c 文件，且内容与 mycp.c 完全相同。

```
> dir
 Volume in drive C has no label.
 Volume Serial Number is B80C-E914

 Directory of C:\Users\DaDa\exp\exp5

03/27/2018 03:35 PM  <DIR>          .
03/27/2018 03:35 PM  <DIR>          ..
03/27/2018 11:42 AM              75 a.txt
03/27/2018 11:39 AM  <DIR>          dst
03/20/2018 11:09 AM              35 makefile
03/20/2018 11:09 AM           1,114 mycp.c
03/27/2018 11:37 AM         391,385 mycp.exe
03/20/2018 11:09 AM           1,114 mycp2.c
03/20/2018 09:44 AM  <DIR>          src
               5 File(s)      393,723 bytes
               4 Dir(s) 12,716,564,480 bytes free
```

5.2.3 容错性测试

当用户只输入 mycp 而没有给出参数时，提示缺少参数，并给出正确的调用格式。

```
> mycp
mycp: missing file operand
Usage: mycp <SRC> <DST>
```

当用户只输入源文件夹而没有给出目标文件夹时，提示缺少目标文件夹，并给出正确调用格式。

```
> mycp src
mycp: missing destination file operand after 'src'
```

```
Usage: mycp <SRC> <DST>
```

5.3 Linux 实验结果

5.3.1 目录拷贝

拷贝之前使用 tree 命令查看当前路径下的目录结构，可以看到有一个名为 src 的文件夹，其中有一个文件 a.txt 和一个子目录 sub，sub 中有一个文件 b.txt，我们打算拷贝这个文件夹。

```
$ tree
.
|-- makefile
|-- mycp
|-- mycp.c
|-- src
    |-- a.txt
    |-- sub
        |-- b.txt

2 directories, 5 files
```

运行 mycp 命令，将 src 文件夹拷贝为 dst 文件夹，命令行参数如下。

```
$ ./mycp src dst
```

再次调用 tree 命令，发现已经有了 dst 目录，结构与 src 目录完全相同，拷贝成功。

```
$ tree
.
|-- dst
|   |-- a.txt
|   |-- sub
|       |-- b.txt
|-- makefile
|-- mycp
|-- mycp.c
|-- src
    |-- a.txt
```

```
    |-- sub
        |-- b.txt

4 directories, 7 files
```

5.3.2 文件拷贝

如果给定 mycp 的源路径参数是文件时，mycp 就拷贝文件。例如将源程序文件 mycp.c 拷贝到当前目录下并命名为 mycp2.c 时，输入命令行参数如下。

```
$ ./mycp mycp.c mycp2.c
```

查看目录，发现有了 mycp2.c 文件，且内容与 mycp.c 完全相同。

```
$ ls
total 44
drwxrwxr-x 4 dada dada 4096 Mar 27 15:45 ./
drwxrwxr-x 5 dada dada 4096 Mar 27 15:45 ../
drwxrwxr-x 3 dada dada 4096 Mar 27 15:09 dst/
-rw-rw-r-- 1 dada dada 33 Mar 20 13:42 makefile
-rwxrwxr-x 1 dada dada 13536 Mar 27 15:06 mycp*
-rw-rw-r-- 1 dada dada 1463 Mar 27 15:45 mycp2.c
-rw-rw-r-- 1 dada dada 1463 Mar 20 13:36 mycp.c
drwxrwxr-x 3 dada dada 4096 Mar 20 13:37 src/
```

5.3.3 容错性测试

当用户只输入 mycp 而没有给出参数时，提示缺少参数，并给出正确的调用格式。

```
$ ./mycp
mycp: missing file operand
Usage: mycp <SRC> <DST>
```

当用户只输入源文件夹而没有给出目标文件夹时，提示缺少目标文件夹，并给出正确调用格式。

```
$ ./mycp src
mycp: missing destination file operand after 'src'
Usage: mycp <SRC> <DST>
```

6 心得体会

经过本次实验我了解了 Windows 和 Linux 系统下的基本文件操作的 API，例如创建目录、复制文件、复制文件夹等等。了解了文件系统的基本结构和组成，例如硬链接、软链接等等方面的知识。

并且在实验过程中，通过调试各个 API，逐渐了解到系统预定义的很多结构体以及函数调用是真的非常方便，在使用的时候，需要注意路径的标准书写，并且需要注意文件的权限问题。这些从操作系统的表面是看不到的，但正是这些非常底层的东西才是我们真正应当理解并掌握的。

经过本次实验，我进一步加强了对于文件的权限问题的了解，并且以后在使用文件系统时，也会理解看似非常简单的一个复制操作，不是简简单单的“Ctrl + c”和“Ctrl + v”，而是需要非常多的系统调用在底层快速运算实现的。这也进一步说明了，我们的程序员之路才刚刚开始，任重而道远。

本次实验所获得的系统底层的知识是通过其他课程所不能了解的，而我相信这或许就是我们科班出身的程序员的核心竞争力。

A 附录

A.1 Windows 版本源代码

A.1.1 mycp.c

```
#include <stdio.h>
#include <windows.h>

void Copy(const char srcPath[], const char dstPath[])
{
    WIN32_FIND_DATA ffd;
    FindFirstFile(srcPath, &ffd);
    if (!(ffd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
    {
        CopyFile(srcPath, dstPath, FALSE);
        return;
    }

    CreateDirectory(dstPath, NULL);
    char dir[MAX_PATH];
```

```
    sprintf(dir, "%s\\*", srcPath);
    HANDLE hFind = FindFirstFile(dir, &ffd);
    for (BOOL f = TRUE; f; f = FindNextFile(hFind, &ffd))
    {
        char *name = ffd.cFileName;
        if (strcmp(name, ".") == 0) continue;
        if (strcmp(name, "..") == 0) continue;
        char srcSubPath[MAX_PATH];
        sprintf(srcSubPath, "%s\\%s", srcPath, ffd.cFileName);
        char dstSubPath[MAX_PATH];
        sprintf(dstSubPath, "%s\\%s", dstPath, ffd.cFileName);
        Copy(srcSubPath, dstSubPath);
    }
}

int main(int argc, char *argv[])
{
    if (argc == 1)
    {
        printf("mycp: missing file operand\n");
        printf("Usage: mycp <SRC> <DST>\n");
        return 1;
    }
    if (argc == 2)
    {
        printf("mycp: missing destination file ");
        printf("operand after '%s'\n", argv[1]);
        printf("Usage: mycp <SRC> <DST>\n");
        return 1;
    }
    Copy(argv[1], argv[2]);
    return 0;
}
```

A.1.2 makefile

```
mycp: mycp.c
    gcc mycp.c -o mycp
```

A.2 Linux 版本源代码

A.2.1 mycp.c

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <dirent.h>
#include <sys/stat.h>

void copyfile(char *srcfile, char *dstfile)
{
    int srcfd = open(srcfile, O_RDONLY);
    int dstfd = open(dstfile, O_WRONLY | O_CREAT, 0666);
    char buf[1024];
    ssize_t bytes;
    while((bytes = read(srcfd, buf, sizeof(buf))) > 0)
    {
        write(dstfd, buf, bytes);
    }
    close(srcfd);
    close(dstfd);
}

void copy(char *src, char *dst)
{
    struct stat statbuf;
    lstat(src, &statbuf);

    if (S_ISLNK(statbuf.st_mode))
    {
        char buf[50];
```

```
        readlink(src, buf, sizeof(buf));
        symlink(buf, dst);
        return;
    }

    if (!S_ISDIR(statbuf.st_mode))
    {
        copyfile(src, dst);
        return;
    }

    mkdir(dst, statbuf.st_mode);
    DIR *pdir = opendir(src);
    struct dirent *pdirent;
    while ((pdirent = readdir(pdir)) != NULL)
    {
        char *name = pdirent->d_name;
        if (strcmp(name, ".") == 0) continue;
        if (strcmp(name, "..") == 0) continue;

        char srcsub[MAXNAMLEN];
        sprintf(srcsub, "%s/%s", src, name);
        char dstsub[MAXNAMLEN];
        sprintf(dstsub, "%s/%s", dst, name);

        copy(srcsub, dstsub);
    }
}

int main(int argc, char *argv[])
{
    if (argc == 1)
    {
        printf("mycp: missing file operand\n");
        printf("Usage: mycp <SRC> <DST>\n");
        return 1;
```

```
    }
    if (argc == 2)
    {
        printf("mycp: missing destination file ");
        printf("operand after '%s'\n", argv[1]);
        printf("Usage: mycp <SRC> <DST>\n");
        return 1;
    }
    copy(argv[1], argv[2]);
    return 0;
}
```

A.2.2 makefile

```
mycp: mycp.c
    gcc mycp.c -o mycp
```