彻底实现文件共享

杜小平 （西南石油学院）

一、问题的提出

DOS 有两条用于磁盘文件共享的命令。其中 PATH 内部命令用于可执行文件，APPEND 外部命令用于不可执行文件。当将共享文件的路径加在这两条命令的命令行上执行时，就可建立起相应的文件搜索路径，使得用户可在各自的目录下执行位于其它目录下的共享文件。但是，上述方法无法对任何文件做到共享。例如，某一可执行文件运行时需调入覆盖文件、帮助文件、数据文件等形式的其它文件，而这些被调用的文件又只有在当前目录位于特定目录情况下才能被调用时，则在其它目录下执行这一可执行文件时，将由于找不到正确的路径而导致执行失败。针对这一情况，一些报刊杂志发表了许多方法加以解决，大致有这样几种：第一种，在欲共享的可执行文件中修改被调用文件的路径；第二种，将共享文件需调用的那些文件的目录项映射到其他目录中；第三种为每一个共享文件建立一个批处理文件，该批处理文件具有进入共享文件子目录并执行共享文件的功能。方法一、二的缺点是：需借助 pctools、DEBUG 等工具手工操作，一般用户不易掌握；方法三由于对每一共享文件都要建立批处理文件，因而工作量大，占用额外磁盘空间，且执行完共享文件后无法返回原有的用户工作目录。为此，笔者利用 Turbo C 开发了一个实用程序 EXEC，只需提供可执行文件名，就可自动遍历 DOS 磁盘目录树，进入该文件子目录并执行，彻底实现了磁盘文件的共享。

二、编程原理

EXEC 的执行过程是：保存当前用户工作目录，搜索给定的可执行文件，进入该文件所在子目录，执行该文件，退回用户工作目录。其中进入执行文件所在目录是彻底实现文件共享的关键。

为找到匹配的可执行文件，必须以一定的策略搜索 DOS 目录树。首先应确定适当的数据结构。由于磁盘上目录数事先未知，故采用动态链表的数据结构，其形式为：

```
struct node{
char dirbuf[MAXDIR];
struct node * next;
}
```

其中 dirbuf 字段用于存放目录路径名，next 为链表指针。还需为链表设置表头指针 head。

搜索从根目录开始。若在某一子目录 A 下查到下一级子目录，就动态分配一个链表结点，将该子目录名存放到结点的 dirbuf 字段中，并将该结点插入表头结点之后。若子目录 A 下无匹配的文件且其下的所有子目录已记录完毕，则进入表头结点后的第一个结点所对应的子目录（即子目录廿的最后一个子目录），在进入前还应将该结点从链表中删除，回收该节点所占内存空间。如此递归处理直至匹配文件找到为止。一旦找到匹配文件，就首先释放所有节点占据的内存空间，然后装载该文件并执行。由于这时系统已进入该文件所在目录，因此就避免了前面提到的问题。程序运行完后，又回到用户工作目录。

EXEC 用法是：当只键入 EXEC 时，就显示使用方法。当要执行某一可执行文件（可是 EXE、COM 和 BAT 文件中的一种）时，则键入 EXEC filename,filename 为可执行文件名。如果将 EXEC 的路径加到 PATH 的搜索路径中，则可在盘上的任何地方执行任何可执行文件，从而彻底地实现了文件的共享。

三、程序清单

```
/ * Filename:EXE.C
  Fining and executing specified executive file.
  compiled by turbo C 2.0 or borland C++ 2.0,3.0 * /
```

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <dir.h>
#include <string.h>
#include <process.h>
void serch();
struct node{4char dirbuf[MAXDIR];
    struct node * next;
    }node;
struct node * head;
char file[15],olddir[MAXDIR],middir[MAXDIR],curdir [MAX-
DIR];
main(int argc,char * argv[])
    {
    if(argc!=2)
      {
      printf("\aUSAGE:EXEC filename\n");
      exit(0);
      }
    head=(struct node * )malloc(sizeof(node));
    head->next=NULL;
    strcpy(file,argv[1]);
    strupr(file);
    getcwd(olddir,MAXDIR);
    chdir("\\");
    serch();
    }
void serch()
    {
    struct ffblk f;
    struct node * p;
    int done,length;
    char filename[MAXFIEL],ext[MAXEXT],drive[MAXDRIV
E],dir [MAX- DIR];
    done=findfirst("*.*",&f,FA_DIREC);
    while(!done)
      {
      fnsplit(f.ff_name,drive,dir,filename,ext);
      if(! strcmp(file,filename)& & (! strcmp(ext,".EXE")| | !
strcmp(ext,". COM")
      ||!strcmp(ext,".BAT")))
        {
        while(head->next)
          {
          p=head->next;
          head->next=p->next;
          free(p);
          }
        free(head);
        if(system(file)==-1)
          {
          printf("\aExecuting %s fails\n",file);
          exit(0);
          }
        chdir(olddir);
        exit(0);
        }
      if(f.ff_attrib==FA_DIREC& & strcmp(f.ff_name,".")
&&strcmp(f.ff_name,".."))
        {
        getcwd(curdir,MAXDIR);
        length=strlen(curdir);
        p=(struct node * )malloc(sizeof(node));
        if(curdir[length-1]!='\\')strcat(curdir,"\\");
        strcpy(p->dirbuf,curdir);
        strcat(p->dirbuf,f.ff_name);
        p->next=head->next;
        head->next=p;
        }
      done=findnext(&f);
      }
if(!head->next)
      {
      free(head);
      chdir(olddir);
      printf("NO %s.EXE,%s.COM OR%s.BAT EXISTING
\n", file,file,file);
      exit(0);
      }
strcpy(middir,(head->next)->dirbuf);
head->next=p->next;
free(p);
p=head->next;
chdir("\\");
chdir(middir);
serch();
}
```