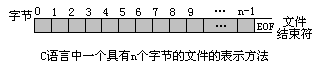
**操作系统第八次实验报告**

1. 文件和流的关系

　　C将每个文件简单地作为顺序字节流(如下图)。每个文件用文件结束符结束，或者在特定字节数的地方结束，这个特定的字节数可以存储在系统维护的管理数据结构中。当打开文件时，就建立了和文件的关系。

　　在开始执行程序的时候，将自动打开3个文件和相关的流：标准输入流、标准输出流和标准错误。流提供了文件和程序的通信通道。例如，标准输入流使得程序可以从键盘读取数据，而标准输出流使得程序可以在屏幕上输出数据。打开一个文件将返回指向FILE结构(在stdio.h中定义)的指针，它包含用于处理文件的信息，也就是说，这个结构包含文件描述符。文件描述符是操作系统数组(打开文件列表的索引)。每个数组元素包含一个文件控制块(FCB, File Control Block)，操作系统用它来管理特定的文件。

　　标准输入、标准输出和标准错误是用文件指针stdin、stdout和stderr来处理的。



2. C语言文件操作的底层实现简介

2.1 FILE结构体

C语言的stdio.h头文件中，定义了用于文件操作的结构体FILE。这样，我们通过fopen返回一个文件指针(指向FILE结构体的指针)来进行文件操作。可以在stdio.h(位于visual studio安装目录下的include文件夹下)头文件中查看FILE结构体的定义，如下：

typedef struct {

short level; /\* fill/empty level of buffer \*/

unsigned flags; /\* File status flags \*/

char fd; /\* File descriptor \*/

unsigned char hold; /\* Ungetc char if no buffer \*/

short bsize; /\* Buffer size \*/

unsigned char \*buffer; /\* Data transfer buffer \*/

unsigned char \*curp; /\* Current active pointer \*/

unsigned istemp; /\* Temporary file indicator \*/

short token; /\* Used for validity checking \*/

} FILE; /\* This is the FILE object \*/

VC6.0中：

#ifndef \_FILE\_DEFINED

struct \_iobuf {

　　　　char \*\_ptr; //文件输入的下一个位置

　　　　int \_cnt; //当前缓冲区的相对位置

　　　　char \*\_base; //指基础位置(即是文件的其始位置)

　　　　int \_flag; //文件标志

　　　　int \_file; //文件的有效性验证

　　　　int \_charbuf; //检查缓冲区状况,如果无缓冲区则不读取

　　　　int \_bufsiz; //???这个什么意思

　　　　char \*\_tmpfname; //临时文件名

};

typedef struct \_iobuf FILE;

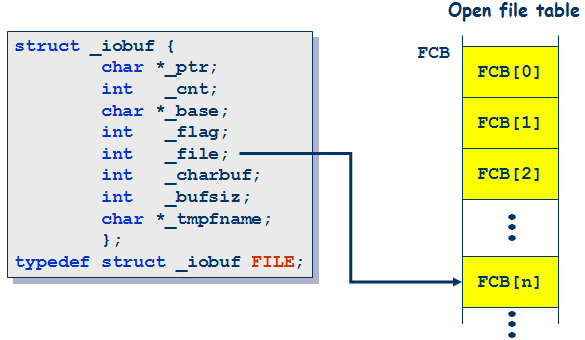
#define \_FILE\_DEFINED

#endif

2.2 C语言文件管理的实现

C程序用不同的FILE结构管理每个文件。程序员可以使用文件，但是不需要知道FILE结构的细节。实际上，FILE结构是间接地操作系统的文件控制块  
(FCB)来实现对文件的操作的，如下图：

上面图中的\_file实际上是一个描述符，作为进入打开文件表索引的整数。



2.3 操作系统文件管理简介

从2.2中的图可以看出，C语言通过FILE结构可以间接操作文件控制块(FCB)。为了加深对这些的理解，这里科普下操作系统对打开文件的管理。

文件是存放在物理磁盘上的，包括文件控制块(FCB)和数据块。文件控制块通常包括文件权限、日期（创建、读取、修改）、拥有者、文件大小、数据块信息。数据块用来存储实际的内容。对于打开的文件，操作系统是这样管理的：

|  |  |
| --- | --- |
| 1 | 系统维护了两张表，一张是系统级打开文件表，一张是进程级打开文件表（每个进程有一个）。 |

系统级打开文件表复制了文件控制块的信息等；进程级打开文件表保存了指向系统级文件表的指针及其他信息。

系统级文件表每一项都保存一个计数器，即该文件打开的次数。我们初次打开一个文件时，系统首先查看该文件是否已在系统级文件表中，如果不在，则创建该项信息，否则，计数器加1。当我们关闭一个文件时，相应的计数也会减1，当减到0时，系统将系统级文件表中的项删除。

进程打开一个文件时，会在进程级文件表中添加一项。每项的信息包括当前文件偏移量（读写文件的位置）、存取权限、和一个指向系统级文件表中对应文件项的指针。系统级文件表中的每一项通过文件描述符（一个非负整数）来标识。

联系2.2和2.3上面的内容，可以发现，应该是这样的：FILE结构体中的\_file成员应该是指向进程级打开文件表，然后，通过进程级打开文件表可以找到系统级打开文件表，进而可以通过FCB操作物理磁盘上面的文件。

2.4 文件操作的例子

#include <stdio.h> void main() {

FILE \* fp1;

FILE \* fp2;

fp1 = fopen("filetest.cpp","r");

fp2 = fopen("filetest.cpp","r");

　　char buffer[256];   
 fscanf(fp1,"%s",buffer);

printf("%s\n",buffer);

　　 fscanf(fp2,"%s",buffer);

printf("%s\n",buffer);

printf("fp1(filetest.cpp):%d\n",fp1->\_file);

printf("fp2(filetest.cpp):%d\n",fp2->\_file);

printf("stdin:%d\n",stdin->\_file);

printf("stdout:%d\n",stdout->\_file);

printf("stderr:%d\n",stderr->\_file);

}

filetest.cpp中的内容如下：

#include<stdio.h> int main()

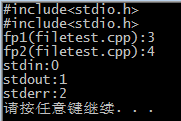
{

printf("Hello World!\n");

return 0;

}

运行结果如下：



通过这个程序可以看出，应该是每打开一次文件，哪怕多次打开的都是同一个文件，进程级打开文件表中应该都会添加一个记录。如果是打开的是同一个文件，这多条记录对应着同一个物理磁盘文件。由于每一次打开文件所进行的操作都是通过进程级打开文件表中不同的记录来实现的，这样，相当于每次打开文件的操作是相对独立的，这就是上面的程序的运行结果中，两次读取文件的结果是一样的(而不是第二次读取从第一次结束的位置进行)。

另外，还可以看出，程序运行的时候，默认三个流是打开的stdin，stdout和stderr，它们的\_file描述符分别是0、1和2。也可以看出，该程序打开的文件描述符依次从3开始递增。

3.顺序访问文件

3.1 顺序写入文件

先看一个例子：

#include <stdio.h> int main()

{

int account;//账号

char name[30];//账号名

double balance;//余额

FILE \*cfPtr;

if ((cfPtr=fopen("clients.dat","w"))==NULL)

{

printf("File could not be opened.\n");

}

else

{

printf("Enter the account, name and the balance:\n");

printf("Enter EOF to end input.\n");

printf("? ");

scanf("%d%s%lf",&account,name,&balance);

while(!feof(stdin))

{

fprintf(cfPtr,"%d %s %.2f\n",account,name,balance);

printf("? ");

scanf("%d%s%lf",&account,name,&balance);

}

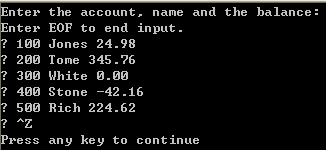
fclose(cfPtr);

}

return 0;

}

运行结果：



从上面的例子中可以看出，写入文件大致需两步：定义文件指针和打开文件。

函数**fopen**有两个参数：文件名和文件打开模式。文件打开模式‘w’说明文件时用于写入的。如果以写入模式打开的文件不存在，则fopen将创建该文件。如果打开现有的文件来写入，则将抛弃文件原有的内容而没有任何警告。在程序中，if语句用于确定文件指针cfPtr是否是NULL(没有成功打开文件时fopen的返回值)。如果是NULL，则将输出错误消息，然后程序终止。否则，处理输入并写入到文件中。

**foef(stdin)**用来确定用户是否从标准输入输入了文件结束符。文件结束符通知程序没有其他数据可以处理了。foef的参数是指向测试是否为文件结束符的FILE指针。一旦输入了文件结束符，函数将返回一个非零值；否则，函数返回0。当没有输入文件结束符时，程序继续执行while循环。

fprintf(cfPtr,"%d %s %.2f\n",account,name,balance);向文件clients.dat中写入数据。稍后通过用于读取文件的程序，就可以提取数据。函数fprintf和printf等价，只是fprintf还需要一个指向文件的指针，所有数据都写入到这个文件中。

在用户输入文件结束之后，程序用fclose关闭clients.dat文件，并结束运行。函数fclose也接收文件指针作为参数。如果没有明确地调用函数fclose，则操作系统通常在程序执行结束的稍后关闭文件。这是操作系统“内务管理”的一个示例，但是，这样可能会带来一些难以预料的问题，所以一定要注意在使用结束之后关闭文件。

3.2 文件打开模式

|  |  |
| --- | --- |
| 模式 | 说明 |
| r | 打开文件，进行读取。 |
| w | 创建文件，以进行写入。如果文件已经存在，则删除当前内容。 |
| a | 追加，打开或创建文件以在文件尾部写入。 |
| r+ | 打开文件以进行更新(读取和写入)。 |
| w+ | 创建文件以进行更新。如果文件已经存在，则删除当前内容。 |
| a+ | 追加，打开或者创建文件以进行更新，在文件尾部写入。 |

3.3 顺序读取文件

下面的例子读取的是上一个例子中写入数据生成的文件。

#include <stdio.h> int main()

{

int account;//账号

char name[30];//账号名

double balance;//余额

FILE \*cfPtr;

if ((cfPtr=fopen("clients.dat","r"))==NULL)

{

printf("File could not be opened.\n");

}

else

{

printf("%-10s%-13s%s\n","Account","Name","Balance");

fscanf(cfPtr,"%d%s%lf",&account,name,&balance);

while(!feof(cfPtr))

{

printf("%-10d%-13s%lf\n",account,name,balance);

fscanf(cfPtr,"%d%s%lf",&account,&name,&balance);

}

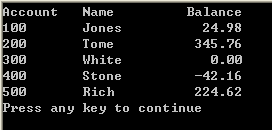
fclose(cfPtr);

}

return 0;

}

运行结果：

  
  
上面的例子中，只需将第一个例子中的文件打开模式从w变为r，就可以打开文件读取数据。

同样地，fscanf(cfPtr,"%d%s%lf",&account,name,&balance);函数从文件中读取一条记录。函数fscanf和函数scanf等价看，只是fscanf接收将从中读取数据的文件指针作为参数。在第一次执行前面的语句时，account的值为100，name的值是Jones，而balance等于24.98。每次执行第二条fscanf语句时，将从文件中读取另一条记录，而account，name和balance将有新值。当到达文件结束位置时，关闭文件，而程序终止。

要从文件中顺序检索数据，程序通常从文件的开始来读取，而且连续读取所有数据，直至找到期望的数据。在程序执行过程中，有可能会多次处理文件中的数据(重新从文件的开头处理数据)。这时候就要用到函数rewind(cfPtr);，它可以使程序的文件位置指针(表示文件中将要读取或者写入的下一个字节的位置)重新设置到文件的开头(也就是偏移量为0的字节)。注意，文件位置指针并不是指针，它是指定文件中将进行下一次读取或者写入的位置的整数值，有时候也称其为文件偏移量，它是FILE结构的成员。

4.随机访问文件

文件中用格式化输入函数fprintf所创建的记录的长度并不是完全一致的。然而，在随机访问文件中，单个记录的长度通常是固定的，而且可以直接访问(这样速度更快)而无需通过其他记录来查找。这使得随机文件访问适合飞机订票系统，银行系统，销售点系统和其他需要快速访问特定数据的事务处理系统。我们可以有很多方法来实现随机访问文件，但是这里我们将把讨论的范围限制在使用固定长度记录的简单方法上。

函数fwrite把从内存中特定位置开始的指定数量的字节写入到文件位置指针指定的文件位置，函数fread从文件位置指针指定的文件位置处把指定数量的字节复制到指定的内存位置。fwrite和fread可以从磁盘上读取数据数组，以及向磁盘上写入数据数组。fread和fwrite的第三个参数是从磁盘中读取或者写入到磁盘上的数组元素的个数。

文件处理程序很少向文件中写入字段。通常情况下，它们一次写入一个struct。

4.1 创建随机访问的文件

#include<stdio.h> struct clientData

{

int acctNum;

char lastName[15];

char firstName[10];

double balance;

}; int main()

{

int i;

struct clientData blankClient={0,"","",0.0};

FILE \*cfPtr;

if ((cfPtr = fopen("credit.dat","wb"))== NULL)

{

printf("File could not be opened.\n");

}

else

{

for (i=1;i<=100;i++)

{

fwrite(&blankClient,sizeof(struct clientData),1,cfPtr);

}

fclose(cfPtr);

}

return 0;

}

fwrite(&blankClient,sizeof(struct clientData),1,cfPtr);用于向文件中写入一个数据块，其会在cfPtr指向的文件中写入大小为sizeof(struct clientData)的结构blankClient。当然，也可以写入对象数组的多个元素，只需把数组名传给第一个参数，把要写入的元素个数写入第三个参数即可。

4.2 随机向随机访问文件中写入数据

#include<stdio.h> struct clientData

{

int acctNum;

char lastName[15];

char firstName[10];

double balance;

}; int main()

{

int i;

struct clientData client={0,"","",0.0};

FILE \*cfPtr;

if ((cfPtr = fopen("credit.dat","rb+"))== NULL)

{

printf("File could not be opened.\n");

}

else

{

printf("Enter account number(1 to 100, 0 to end input\):\n");

scanf("%d",&client.acctNum);

while (client.acctNum!=0)

{

printf("Enter lastname, firstname, balance\n");

fscanf(stdin,"%s%s%lf",client.lastName,client.firstName,&client.balance);

//在文件中定位用户指定的记录

fseek(cfPtr,(client.acctNum-1)\*sizeof(struct clientData),SEEK\_SET);

//将用户指定的信息写入文件

fwrite(&client,sizeof(struct clientData),1,cfPtr);

//输入下一个账号

printf("Enter account number:\n");

scanf("%d",&client.acctNum);

}

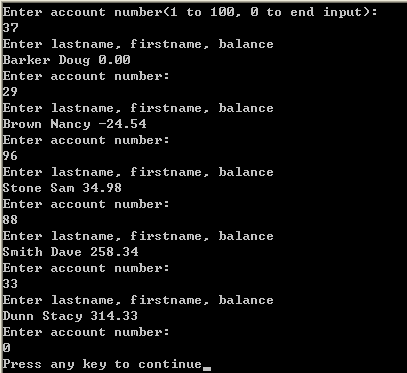
fclose(cfPtr);

}

return 0;

}

运行结果：

  
  
fseek(cfPtr,(client.acctNum-1)\*sizeof(struct clientData),SEEK\_SET);将cfPtr所引用文件的位置指针移动到由(client.acctNum-1)\*sizeof(struct clientData)计算所得到的字节位置处，这个表达式的值称为偏移量或者位移。负号常量SEEK\_SET说明，文件位置指针指向的位置是相对于文件开头的偏移量。

ANSI标准制定了fseek的函数原型为int fseek(FILE \*stream, long int offset, int whence);其中offset是stream指向的文件中从位置whence开始的字节数。参数whence可以有三个值：SEEK\_SET, SEEKCUR或者SEEK\_END，分别对应文件的开头当前位置和结尾。

4.2 从随机访问文件中读取数据

#include<stdio.h> struct clientData

{

int acctNum;

char lastName[15];

char firstName[10];

double balance;

}; int main()

{

struct clientData client={0,"","",0.0};

FILE \*cfPtr;

if ((cfPtr = fopen("credit.dat","rb"))== NULL)

{

printf("File could not be opened.\n");

}

else

{

printf("%-6s%-16s%-11s%10s\n","Acct","Last name","First name","Balance");

while(!feof(cfPtr))

{

fread(&client,sizeof(struct clientData),1,cfPtr);

if (client.acctNum!=0)

{

printf("%-6d%-16s%-11s%10.2f\n",client.acctNum,client.lastName,client.firstName,client.balance);

}

}

fclose(cfPtr);

}

return 0;

}

运行结果：

