中国科学技术大学

文件操作与数据持久化

吴锋

Email: wufeng02@ustc.edu.cn

本章内容概述

- 文件操作与数据持久化
- 二进制文件与序列化
- 文件的定位与随机访问
- 文件的错误处理

本章内容概述

- 文件操作与数据持久化
- 二进制文件与序列化
- 文件的定位与随机访问
- 文件的错误处理

文件的基本概念

- 文件是一组相关数据的有序集合，信息最终是以文件形式存储在永久存储设备中（如硬盘、光盘、U盘等）
- 数据持久性：内存（断电即失） vs 磁盘（永久保存）
- 操作系统及应用程序往往通过对文件的输入输出完成某项功能；操作系统将与主机相连的输入输出设备也视为文件，如键盘是输入文件，显示器是输出文件
- 文件按结构形式分为文本文件和二进制文件
  - 文本文件是全部由字符组成（用ASCII码表示）的具有行列结构的文件，又称为ASCII码文件。便于对字符进行处理，可以直接显示
  - 二进制文件是按数据在内存中的存储形式存储到文件中，一般不能直接显示
  - 文本文件的ASCII码也是二进制的，区别在于数值。例：int型整数5在文本文件中保存的是0x35，而在二进制文件中是0x00,0x00,0x00,0x05四个字节

文件的基本概念

- 文件按数据的物理存取方式分为顺序文件和随机文件
  - 顺序文件：数据在存储介质中的实际顺序与它们进入存储器的顺序一致；一切存储在顺序存储器（如磁带）上的文件，都只能是顺序文件（难以在中间定位）

A记录	B记录	C记录

  - 随机文件：数据散列在存储介质中；硬盘上的文件，一般是随机文件（系统工具中的磁盘整理就是尽可能将分散随机存储的文件数据集中放置，以便于就近读写）

A记录		B记录	C记录	

文件的基本概念

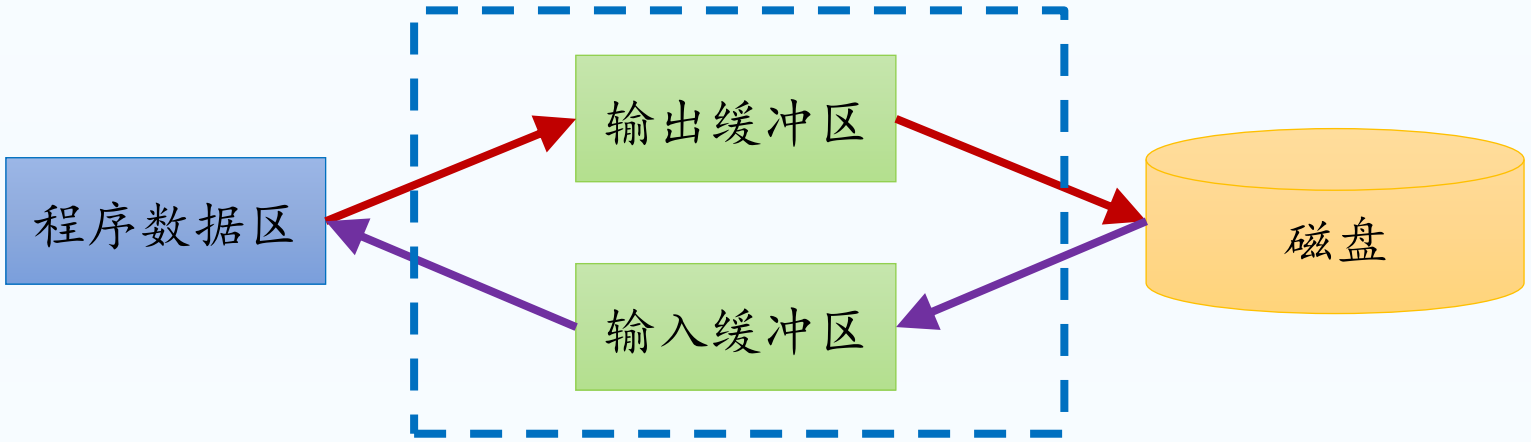
- 文件的读写方式分为顺序存取和随机存取
  - 顺序存取：从头到尾读写文件所有数据，文件被看成一个字符流，称为流式文件
    - 网络中所说的流是指一组有序数据序列，代表传输中的数字序列
    - 向文件顺序写入/读出数据
      - 打开文件并将文件指针置于文件的开头
      - 将内存中的数据顺序写入文件/将数据顺序读入内存
  - 随机存取：随机从文件内指定的位置读写数据
    - 向文件随机写入/读出数据
      - 打开文件并将文件定位指针置于待写入/读出的位置
      - 将内存中的数据写入文件/将数据读入内存
  - 顺序文件只能顺序存取，随机文件两种都可以

文件的基本概念

- 按存储的外部设备分为磁盘文件（普通文件）和设备文件
  - 磁盘文件是指存放在磁盘或其它外部介质上的有序数据集
  - 操作系统把与设备间的输入输出等同于对磁盘文件的读写，称其为设备文件，包括与主机相连的各种外部设备，如显示器、键盘等
  - 通常把显示器定义为标准输出文件(stdout)，向stdout文件输出，即为在显示器上显示
  - 类似的，把键盘定义为标准输入文件(stdin)，从stdin文件读入，即为从键盘读入

文件的基本概念

- 缓冲文件系统和非缓冲文件系统
  - 缓冲文件系统是指系统自动地在内存中为每一个正在使用的文件都开辟一个缓冲区
  - 从内存向磁盘输出数据时，缓冲文件系统先将数据送到内存中的缓冲区，待缓冲区满时才将整个缓冲区的数据一起保存到磁盘文件中（或强制写入）
  - 从磁盘向内存读入数据时，一般是从磁盘文件中一次读入一批数据到缓冲区，程序从缓冲区获取数据，缓冲区空时再次读入一批数据



文件的基本概念

- 缓冲文件系统和非缓冲文件系统
  - 非缓冲文件系统是指系统不自动开辟缓冲区，而由使用文件的程序直接操作文件数据，自行开辟和维护缓冲区
  - 现代操作系统和运行环境都采用缓冲文件系统
    - 写入时不要遗漏缓冲区内尚未存入的数据
    - 读取时可能不是“即时”输入的数据
  - 常见现象：为什么 printf() 后屏幕可能没立即显示？写入文件后立即打开该文件却可能是空的？
  - 核心思想：磁盘 I/O 极慢，CPU 极快；数据先堆积在内存缓冲区，暂缓写入磁盘（减少慢速的 I/O 次数）

文件类型与文件指针

- 操作系统读写文件时需要的信息
  - 文件当前的读写位置
  - 与文件对应的内存缓冲区地址
  - 文件的操作方式，等
- 以上信息都存放在“文件信息区”中，一个由系统定义的结构体类型变量，该结构体类型在stdio.h中定义，标识符是FILE，具体定义与编译系统有关
- C语言标准函数中，系统自动打开和关闭三个标准文件指针
  - 标准输入（默认绑定键盘）stdin
  - 标准输出（默认绑定显示器）stdout
  - 标准出错输出（默认绑定显示器，且无缓冲）stderr

标准输入输出命令行重定向

- 输入重定向（将stdin / 0 重定向，从文件读取内容代替键盘输入）：
  - 方法： ./my_program < input.txt
  - 效果：scanf() 等会从 input.txt 文件中读取输入，无需从键盘输入
- 输出重定向（将stdout / 1 重定向，原本打印在屏幕的内容写入文件）
  - 方法： ./my_program > output.txt （文件如果有内容会被清空）
  - 追加：./my_program >> output.txt （输出内容补充在文件末尾）
  - 效果：printf() 等输出结果会保存在 output.txt 文件中，屏幕无显示
- 错误重定向（将 stderr / 2 重定向，仅将错误信息写入文件）：
  - 方法： ./my_program 2> error.log
- 所有输入输出都重定向到文件（上述重定向可以随意组合）：
  - 方法：./my_program < input.txt > output.txt 2 > error.log
- 一个程序的标准输出作为另一个程序的标准输入：random | sum

文件类型与文件指针

- FILE结构体示例（stdio.h）

```
struct _iobuf {
    int _file;              // 文件描述符
    int _flag;              // 文件状态标志
    char *_ptr;             // 指向当前读取/写入位置的指针
    int _bufsiz;            // 缓冲区大小
    int _cnt;               // 缓冲区中的字节数
    int _charbuf;           // 用于存储单个字符的缓冲区
    char *_base;            // 指向缓冲区起始位置的指针
    char *_tmpfname;        // 临时文件名（如果有）
    … …                     // 可能还会有其他字段
};
typedef struct _iobuf FILE;
```

- 缓冲文件系统中，关键的概念是“文件指针”，定义文件指针的形式：

```
FILE  *fp1,*fp2;
```

- 通过文件指针可以找到存放该文件信息的结构变量，然后按结构变量提供的信息找到文件并进行操作

文件的打开

- 文件需要打开才能操作，打开文件的实质是在用户程序和操作系统间建立起联系，通过文件指针共享文件信息

- C语言使用标准库函数 fopen() 打开或新建文件，函数原型：

  FILE *fopen(const char*fname, const char*mode);

  - fname 是将要访问的文件名，路径必须有效，否则可能找不到文件
  - mode 是文件模式，用以规定文件可以操作的方式，是一个字符串
  - FILE * 是函数返回类型，注意这里返回的是一个文件指针，文件的实质结构体变量由系统产生和维护

文件的绝对路径与相对路径

- 绝对路径：从根目录（如 C:\ 或 / ）开始的完整路径
  - Windows系统（分隔符 \ ）： C:\Users\Admin\Desktop\data.txt
  - Linux系统（分隔符 / ）： /home/user/project/data.txt
- 相对路径：从当前（可执行程序）所在目录开始的路径
  - 当前目录（可以省略）： ./data.txt 或 data.txt
  - 上级目录（可以叠加）： ../config.ini ， ../../config.ini
- 注意：在C语言字符串中，需要用 “\\” 表示 \
  - 例如：fopen(“C:\\Users\\Admin\\Desktop\\data.txt”, ”r”)

文件的操作模式

- 文件操作模式 mode

文件操作模式	意义
“r”	以只读方式打开文本文件，该文件应该存在，不存在则报错
“w”	以只写方式建立文本文件，若文件存在则清空文件内容；若文件不存在则建立该文件
“a”	以追加的方式打开文本文件。若文件不存在，则会建立该文件；如果文件存在，则从文件尾开始写（不会清空文件）

- rb、wb、ab 与 r、w、a 相似，但对象是二进制文件
- r+、w+、a+ 以读写的操作模式打开文本文件，可切换读写模式
- rb+、wb+、ab+ 以读写的操作模式打开二进制文件，可切换读写模式

文件的操作模式（续）

- 读写模式（+）的作用

基础模式	读写模式 (+)	核心区别与应用场景
"r" (只读) 文件必须存在	"r+" (读写) 文件必须存在	多了【修改】的能力。 r: 只能读，不能写。 r+: 可以修改文件中间的任意字节。 场景：修改游戏存档、更新数据库记录。
"w" (只写) 清空/创建	"w+" (读写) 清空/创建	多了【回看】能力。 会清空原有内容，只能读取刚刚写入的数据。 场景：临时文件写完，立刻需要读出来处理。
"a" (追加) 创建/尾部	"a+" (读写) 创建/尾部	多了【读取】能力。 a: 只能向后写。 a+: 可以读取整个文件的历史记录，但写入时强制在末尾添加内容。 场景：聊天软件 (读取历史消息 + 发送新消息)。

思考：修改文件中间的某个字符（原地修改），如果使用 w+ 文件会怎么样?

文件的打开（例）

```
FILE *fp1, *fp2, *fp3;
char filename[]="file3.dat";

// 以文本只读方式打开file1
if (!(fp1=fopen("file1", "r"))) {
  printf("Cannot Open This File!\n");
  exit(0); // 退出程序
}

// 以二进制读写方式打开FILE2.TXT
fp2=fopen("C:\\HOME\\FILE2.TXT", "rb+");
// 以二进制读写方式打开file3.dat
fp3=fopen(filename, "a+b");
```

注意路径中的反斜线需要用字符转换

顺序并不严格

文件的关闭

- 打开文件后，系统会分配文件的缓冲区，文件使用完后应及时关闭文件以释放缓冲区
- C语言使用库函数 fclose() 关闭文件，原型：

  int fclose(FILE *stream);

  - stream是打开文件时获得的文件指针（流指针）
  - 文件正常关闭后返回值为0，否则返回 EOF 即-1
  - 示例：fclose(fp);
- 关闭文件还有一个重要作用是操作系统此时将文件缓冲区的剩余内容写入文件，正常关闭文件才能确保文件内容的正确

文件的关闭（例）

```c
// 典型代码范式
FILE *fp = fopen("demo.txt", "w");
if (fp) {
  fprintf(fp, "Hello Persistence");
  // 此时数据可能还在缓冲区
  fclose(fp);
  // 现在数据一定在磁盘上了
  fprintf(fp, "Hello Again"); // 错误：关闭文件后不能再操作文件
  fp = NULL; // 好习惯：关闭文件后将文件指针置为 NULL
}
```