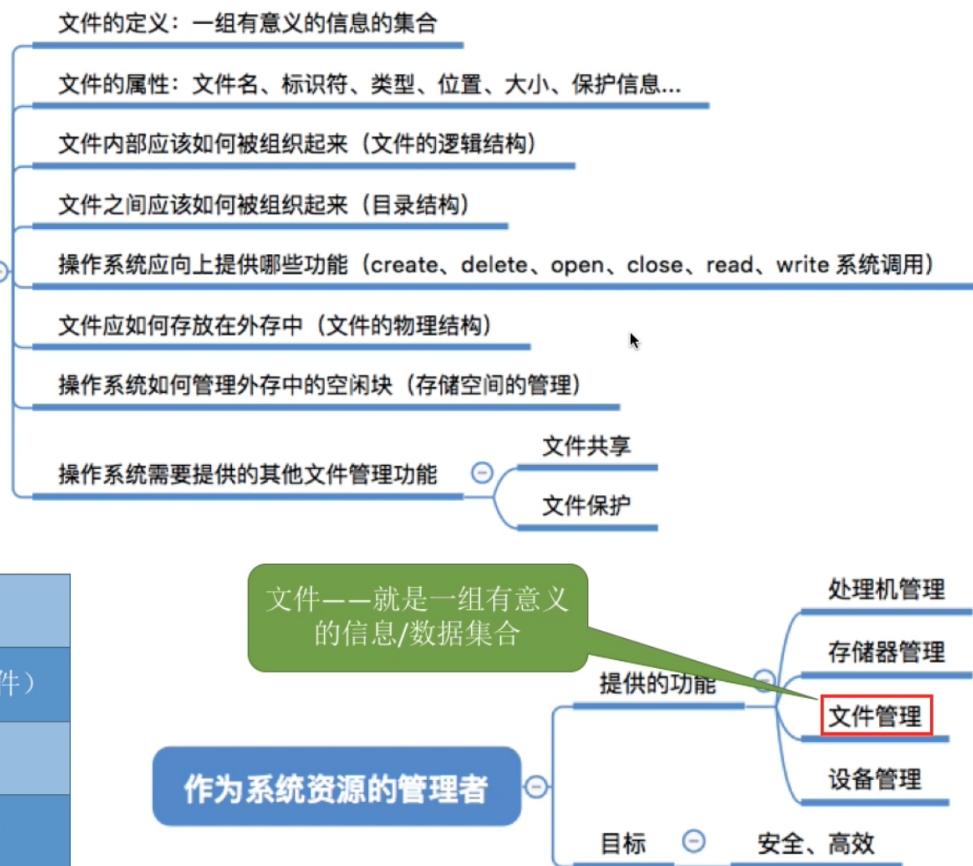
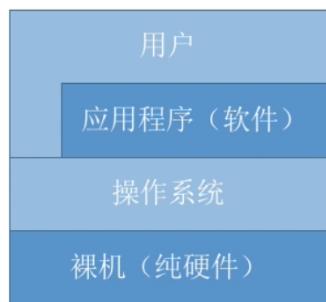


文件的基本概念

初识文件管理



计算机中存放了各种各样的文件，一个文件有哪些属性？

文件内部的数据应该怎样组织起来？

文件之间又应该怎么组织起来？

从下往上看，OS应提供哪些功能，才能方便用户、应用程序使用文件？

从上往下看，文件数据应该怎么存放在外存（磁盘）上？

文件的属性

一个文件有哪些属性？

- 文件名

由创建文件的用户决定文件名，主要是为了方便用户找到文件，同一目录下不允许有重名文件。

- 标识符

一个系统内的各文件标识符唯一，对用户来说毫无可读性，因此标识符只是操作系统用于区分各个文件的一种内部名称。

- 类型

指明文件的类型

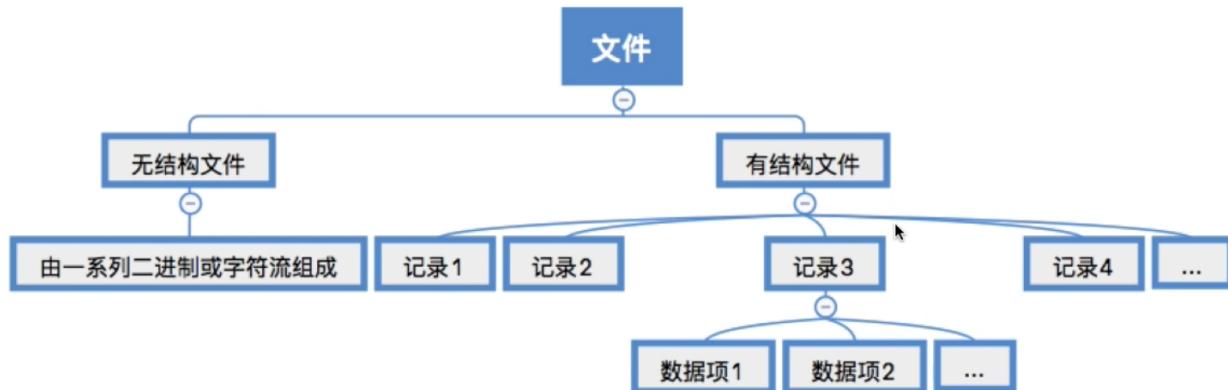
- 位置

文件存放的路径（让用户使用）、在外存中的地址（操作系统使用，对用户不可见）

- 大小
指明文件大小
- 创建时间
- 上次修改时间
- 文件所有者信息
- 保护信息
对文件进行保护的访问控制信息

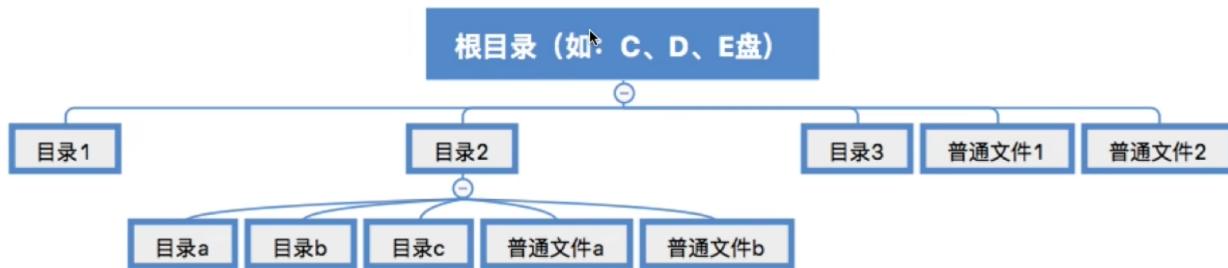
文件内部的数据应该怎样组织起来？

- 无结构文件
如文本文件——由一些二进制或字符流组成，又称“流式文件”
- 有结构文件
如数据库表——由一组相似的记录组成，又称“记录式文件”
记录是一组相关数据项的集合
数据项是文件系统中最基本的数据单位



在结构文件中，各个记录间应该如何组织的问题——应该顺序存放？还是用索引表来表示记录间的顺序？——这是“文件的逻辑结构”重点要探讨的问题

文件之间应该怎样组织起来？

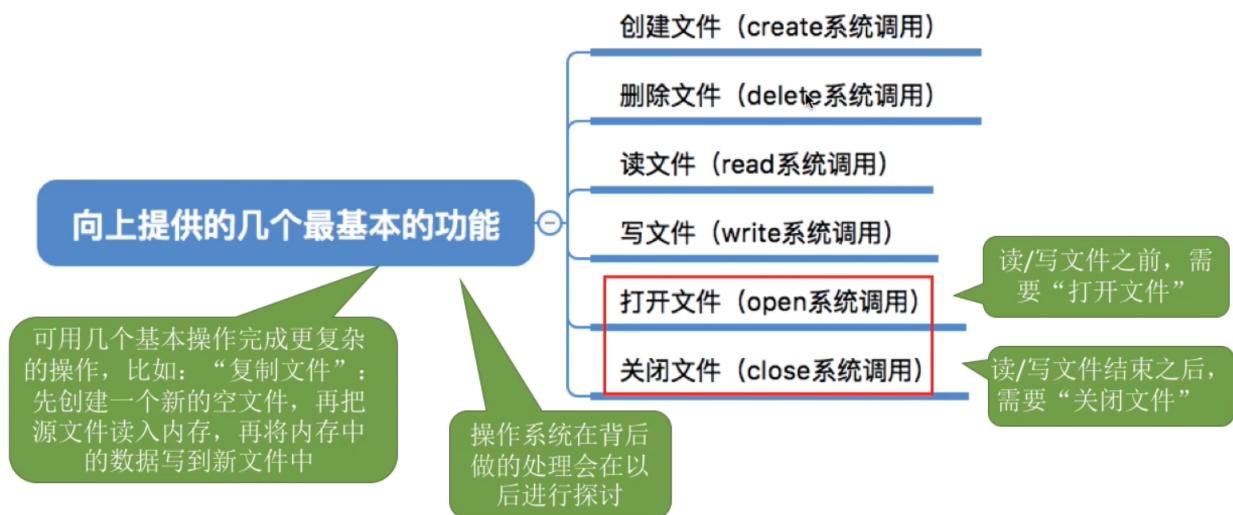


所谓的“目录”起始就是我们熟悉的“文件夹”

用户可以自己创建一层一层的目录，各层目录中存放相应的文件。系统中的各个文件就通过一层一层的目录合理有序的组织起来了

目录其实也是一种特殊的有结构文件（由记录组成），如何实现文件目录是之后会重点探讨的问题

操作系统应该向上提供哪些功能？



可以“创建文件”，（点击新建后，图形化交互进程在背后调用了“create系统调用”）

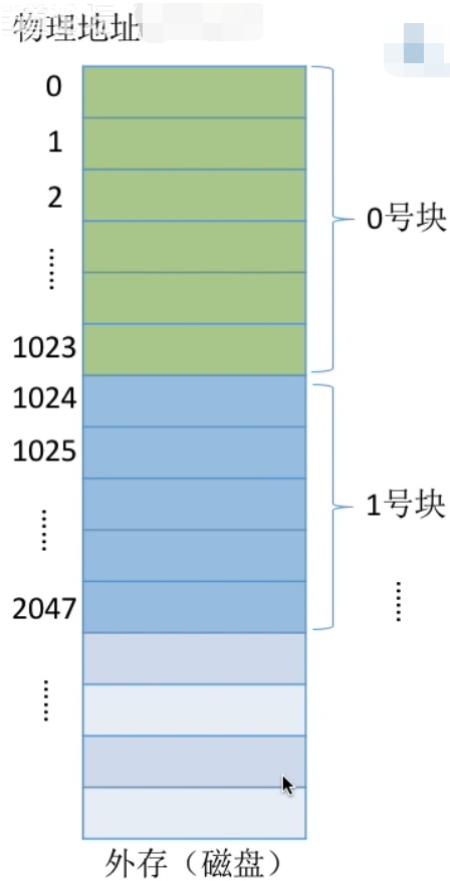
可以“读文件”，将文件数据读入内存，才能让CPU处理（双击后，“记事本”应用程序通过操作系统提供的“读文件”功能，即read系统调用，将文件数据从外存读入内存，并显示在屏幕上）

可以“写文件”，将更改过的文件数据写回外存（我们在“记事本”应用程序中编辑文件内容，点击“保存”后，“记事本”应用程序通过操作系统提供的“写文件”功能，即write系统调用，将文件数据从内存写回外存）

可以“删除文件”（点了“删除”之后，图形化交互进程通过操作系统提供的“删除文件”功能，即delete系统调用，将文件数据从外存中删除）

从上往下看，文件应如何存放在外存？

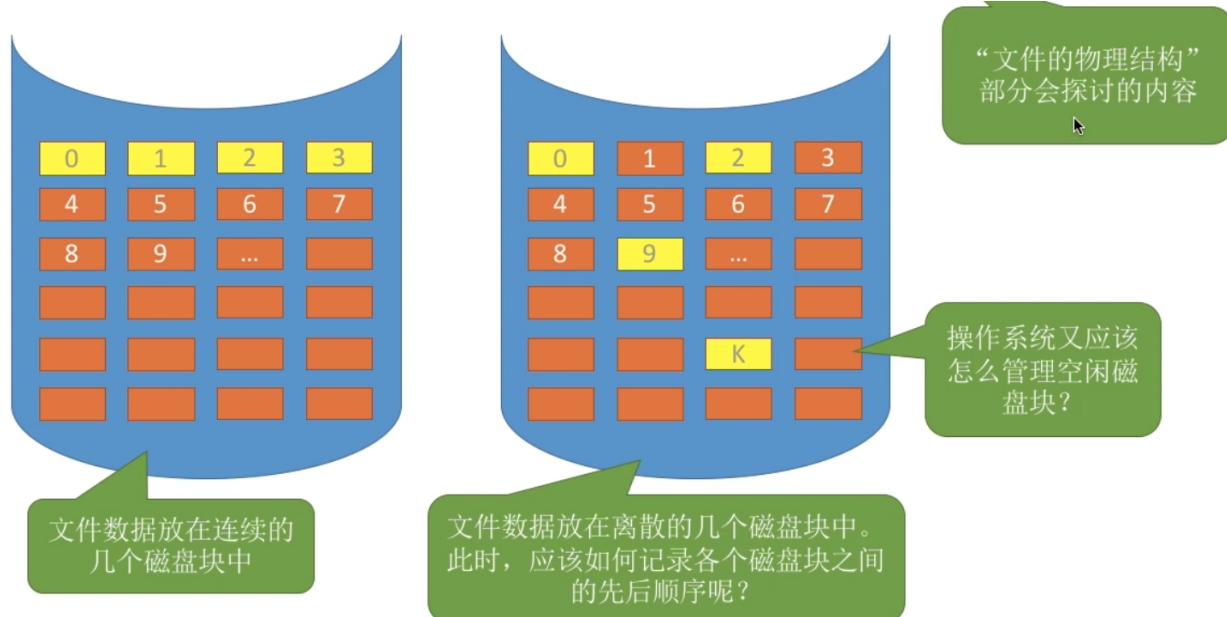
“文件的物理结构”部分会探讨的内容



与内存一样，外存也是由一个个存储单元组成的，每个存储单元可以存储一定量的数据（如1B）。每个存储单元对应一个物理地址

类似于内存分为一个个“内存块”，外存会分为一个个“块/磁盘块/物理块”。每个磁盘块的大小是相等的，每块一般包含2的整数幂个地址（如本例中，一块包含2的10次方个地址，即1KB）。同样类似的是，文件的逻辑地址也可以分为（逻辑块号，块内地址），操作系统同样需要将逻辑地址转换为外存的物理地址（物理块号，块内地址）的形式。块内地址的位数取决于磁盘块的大小

操作系统以“块”为单位为文件分配存储空间，因此即使一个文件大小只有10B，但它依然需要占用1KB的磁盘块。外存中的数据读入内存时同样以块为单位



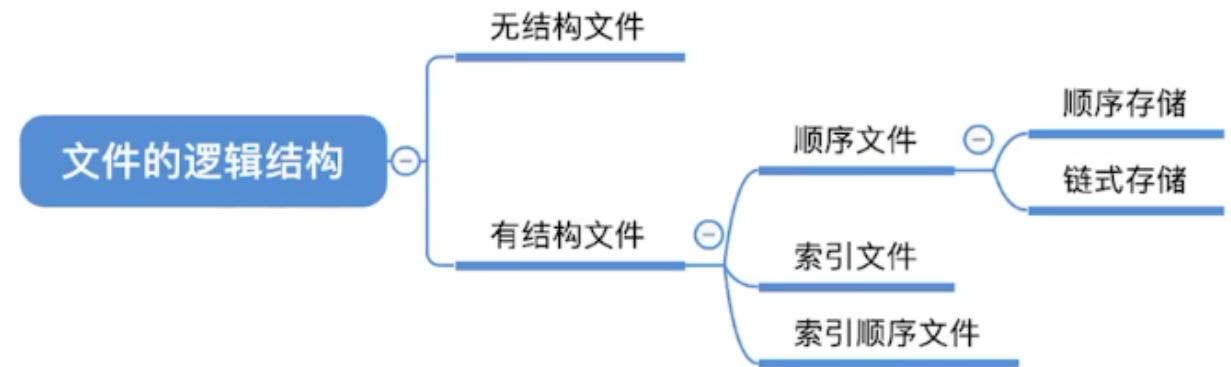
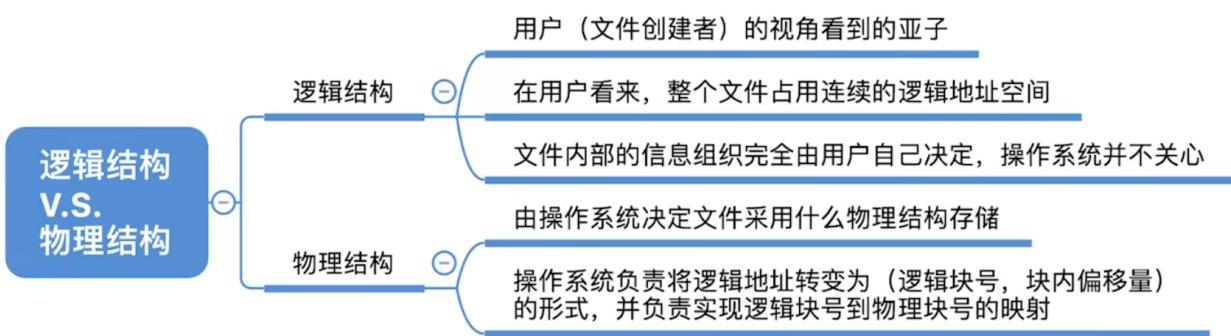
其他需要由操作系统实现的文件管理功能

文件共享：使多个用户可以共享使用一个文件

文件保护：如何保证不同的用户对文件有不同的操作权限

会结合Windows操作系统的实际应用进行探讨

逻辑结构VS物理结构

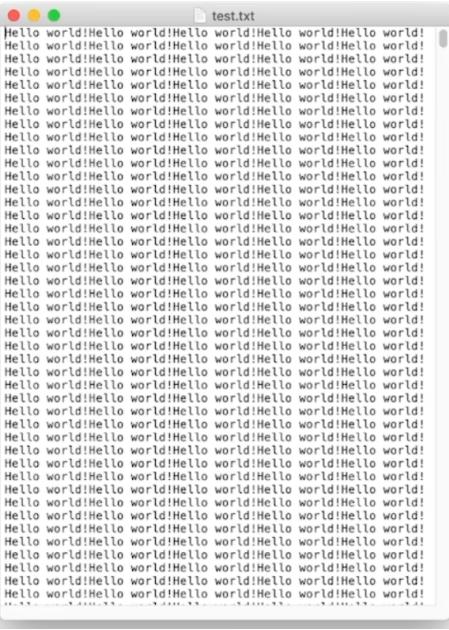


C语言创建无结构文件

```

FILE *fp = fopen("test.txt", "w"); //打开文件
if( fp == NULL ){
    printf("打开文件失败!");
    exit(0);
}
//写入1w个Hello world
for (int i=0; i<10000; i++)
    fputs("Hello world!", fp);
fclose(fp); //关闭文件

```



逻辑结构 (从用户视角看)

每个字符1B。在用户看来，整个文件占用一片连续的逻辑地址空间



Eg: 你要找到第16个字符（编号从0开始）

```

FILE *fp = fopen("test.txt", "r"); //以"读"方式打开文件
if( fp == NULL ){
    puts("Fail to open file!");
    exit(0);
}
fseek(fp, 16, SEEK_SET); //读写指针指向16
char c = fgetc(fp); //从读写指针所指位置读出1个字符
printf("字符 '%c'", c); //打印从文件读出的字符
fclose(fp); //关闭文件

```

用户用逻辑地址访问文件

1000

2000



物理结构 (从操作系统视角看)

Hello world! Hello world! Hello world!

操作系统视角：反正就是一堆二进制数据，每个磁盘块可存储1KB，拆就完了！

1KB #0	1KB #1	1KB #2	1KB #3	1KB #4	1KB #5
-----------	-----------	-----------	-----------	-----------	-----------	-------

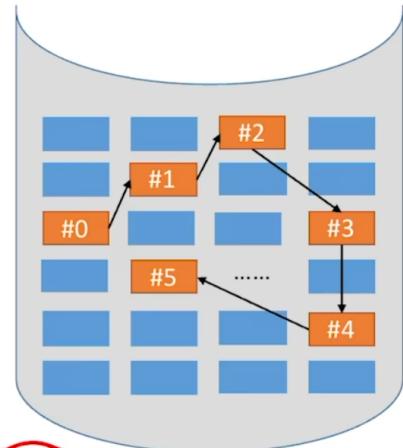
被操作系统拆分为若干个块，逻辑块号相邻

用户：

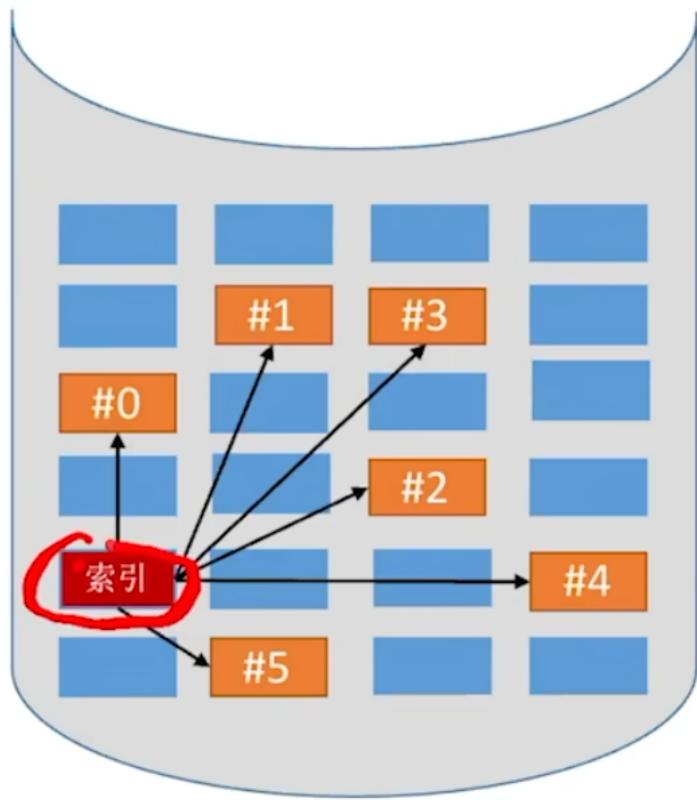
使用 C 语言库函数 `fseek`，将文件读写指针指向位置 n
使用 C 语言库函数 `fgetc`，从读写指针所指位置读出 1B 内容

`fgetc` 底层使用了 Read 系统调用，
操作系统将（逻辑块号，块内偏移量）
转换为（物理块号，块内偏移量）

指明逻辑地址



链接分配：逻辑上相邻的块在物理上用链接指针表示先后关系



索引分配：操作系统为每个文件维护一张索引表，其中记录了逻辑块号 → 物理块号 的映射关系

C 语言创建顺序文件

```

typedef struct {
    int number;           //学号
    char name[30];        //姓名
    char major[30];       //专业
} Student_info;

//以"写"方式打开文件
FILE *fp = fopen("students.info", "w");
if(fp == NULL) {
    printf("打开文件失败!");
    exit(0);
}
Student_info student[N];      //用数组保存N个学生信息
for(int i = 0; i<N; i++) {   //生成 N 个学生信息
    student[i].number=i;
    student[i].name[0]='?';
    student[i].major[0]='?';
}

//将 N 个学生的信息写入文件
fwrite(student, sizeof(Student_info), N, fp);
fclose(fp);

```

用户视角：
每个学生记录占 64B
sizeof(Student_info)



```

//以"读"方式打开文件
FILE *fp = fopen("students.info", "r");
if(fp == NULL) {
    printf("打开文件失败!");
    exit(0);
}
//文件读写指针指向编号为5的学生记录
fseek(fp, 5*sizeof(Student_info), SEEK_SET);
Student_info stu;
//从文件读出1条记录，记录大小为 sizeof(Student_info)
fread(&stu, sizeof(Student_info), 1, fp);
printf("学生编号: %d\n", stu.number);
fclose(fp);

```

用户用逻辑地
址访问文件

顺序文件采用顺序存储/链式存储

顺序文件：各个记录可以顺序存储或链式存储。

顺序存储，各条记录
相邻这存放



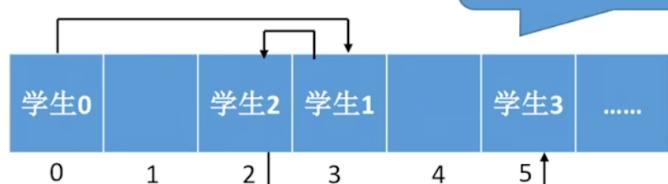
支持随机访问：指可
以直接确定第i条记录
的逻辑地址

```

typedef struct {
    int number;           //学号
    char name[30];        //姓名
    char major[30];       //专业
} Student_info;

```

链式存储，各条记录离散着
存放，用指针表示先后关系

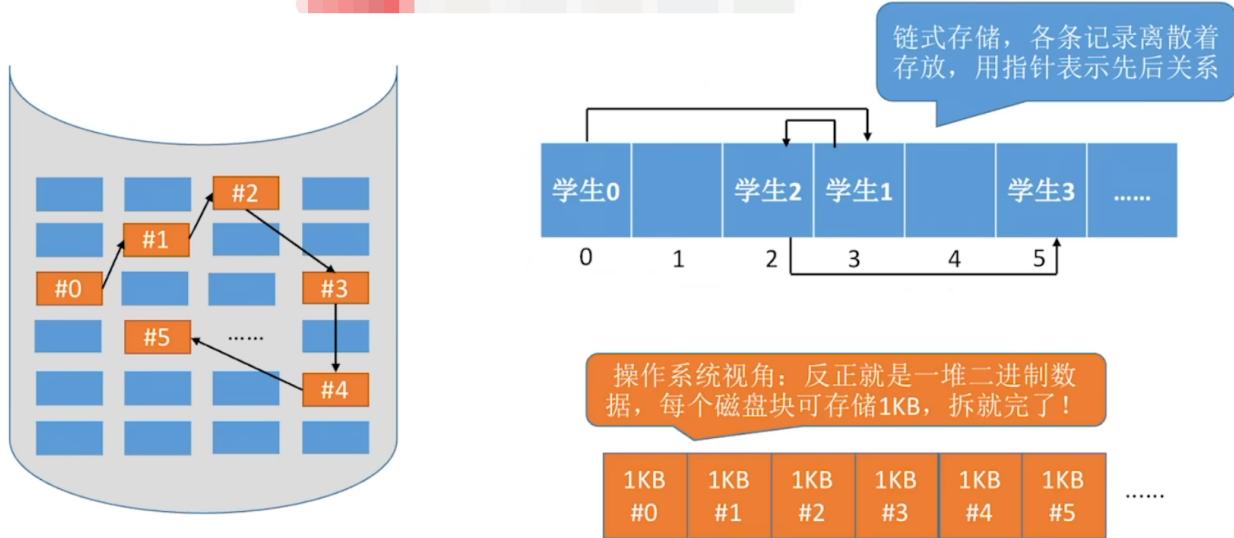


```

typedef struct {
    int number;           //学号
    char name[30];        //姓名
    char major[30];       //专业
    int next;             //下一个学生记录的存放位置
} Student_info;

```

链式存储的顺序文件采用连续存储

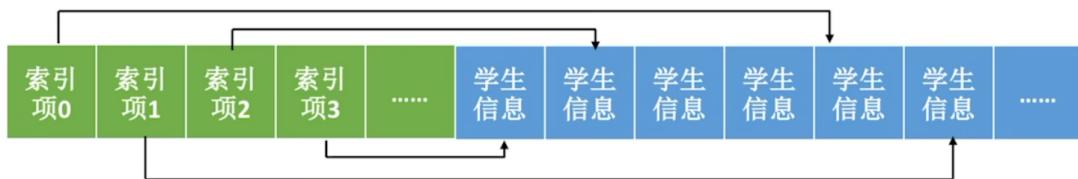


文件内部各条记录链式存储：由创建文件的用户自己设计的
文件整体用链接分配：由操作系统决定

逻辑结构：索引文件

```
typedef struct {
    int number;      //学号
    int addr;        //学生记录的逻辑地址
} IndexTable;

typedef struct {
    char name[30];   //姓名
    char major[30];  //专业
    //还可添加其他各种各样的学生信息
} Student_info;
```



索引文件：从用户视角来看，整个文件依然是连续存放的。如：前1MB存放索引项，后续部分存放记录。

索引文件采用索引分配

