IO 进程线程

第一天：IO概念，标准IO,

第二天：标准IO，文件IO

第三天：文件IO，库

第四天：进程

第五天：线程

第六天：线程的同步互斥

第七、八天：进程间的通信

如何学：记函数的功能以及函数的名字，将白天的练习还有代码，晚上敲两遍

1. IO

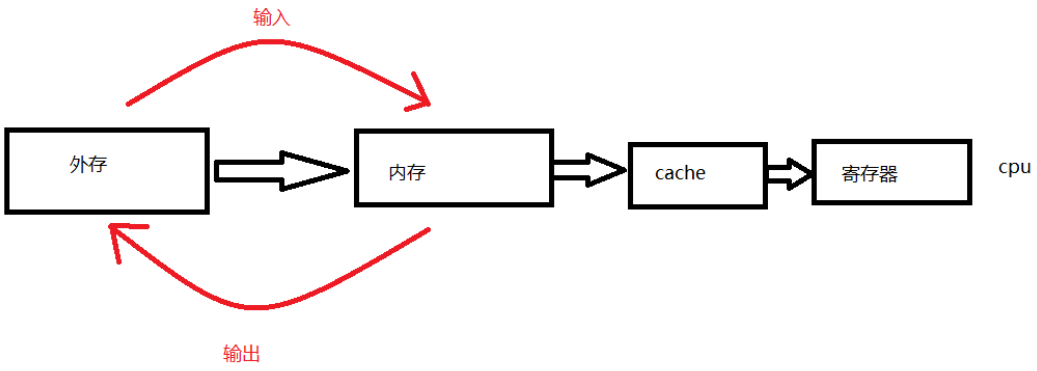
【1】什么是IO

i：input　输入　　从外存设备输入到内存中

o：output 输出　　从内存输出到外存设备中

外部存储设备：硬盘、磁盘

内存：DDR4



总结：IO就是数据从硬盘到内存，内存到硬盘的流动。

【2】IO函数分类

1. 文件IO

文件IO是由操作系统提供的基本IO函数，与操作系统绑定，又称之为系统调用.

例子：

windows	Linux
file_read	read

注意：

1. 文件IO是与操作系统绑定的，所以不同的操作系统有不同的文件IO函数。所以文件IO的移植性更低
2. 文件IO涉及到用户空间到内核空间的切换，cpu模式的切换，C代码调用汇编指令等等操作，是一种**耗时操作**。

2. 标准IO

   根据ANSI标准，是对文件IO的二次封装，printf scanf

   标准IO是对文件IO的二次封装，所以最终标准IO还是会去调用文件IO

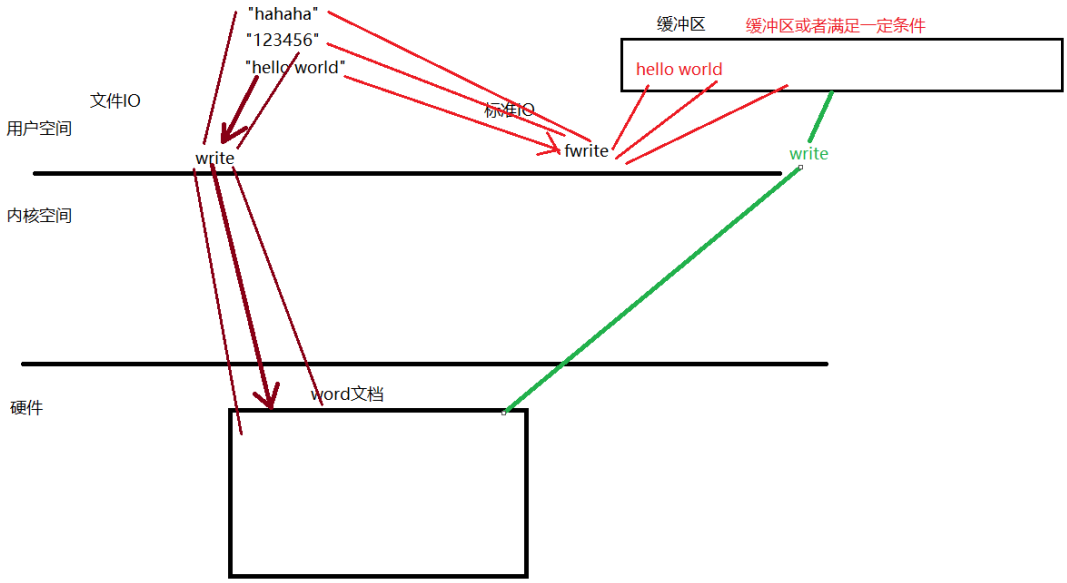
```
scanf:

if(OS == windows)
{
    file_read();    //windows的文件IO
}
else if(OS == Linux)
{
    read();         //Linux的文件IO
}
```

注意：

1. 标准IO的移植性更高。
2. 提高输入输出的效率；

   设置了一个缓冲区，缓冲区满或者满足一定条件后，调用文件IO，陷入内核空间，由内核完成对硬件的操作，大大减少了对文件IO的调用。



2. 标准IO

【1】流和流指针

流（stream）:将数据一个一个的移入缓冲区，或者移出缓冲区的形式叫做字节流。

流指针（FILE*）：每打开一个文件，都会在用户空间中申请一片空间（缓冲区）。

管理这片空间的变量都存储在FILE结构体中，该结构体由系统定义好的，我们直接使用即可。

1. 查看FILE结构体

> 1）sudo apt-get install ctags
>
> 2）cd /usr/include/
>
> 3）sudo ctags -R
>
> 4）在家目录下有个隐藏文件：vim .vimrc 打开隐藏文件
>
> 在结尾或者开头添加 set tags+=/usr/include/tags
>
> 追代码：
>
> 左键选中要追的代码
>
> ctrl +]
>
> ctrl + 鼠标左键
>
> 返回：
>
> ctrl + t
>
> ctrl + 鼠标右键

```
vi -t FILE          查看系统定义好的数据类型，宏
 typedef struct _IO_FILE FILE;

鼠标移动到_IO_FILE，按下ctrl + ]
```

```
struct _IO_FILE {
  char* _IO_buf_base;    /* Start of reserve area. */ 缓冲区的起始地址
  char* _IO_buf_end;     /* End of reserve area. */    缓冲区结束地址

  int _fileno;           文件描述符，在文件IO的时候讲解.
}

int* ptr = 0x10;
ptr+1 == 0x14

short int* ptr = 0x10;
ptr+1 == 0x12;
```

```
int* ptr = 0x14;
ptr-1 == 0x10
```

2. man手册 -->函数原型

查看：函数的功能参数返回值

```
       1   可执行程序或 shell 命令
       2   系统调用(内核提供的函数)     文件IO函数
       3   库调用(程序库中的函数)       库函数，标准IO函数

man 3 printf
```

3 .特殊的流指针

在main函数运行之前，系统会自动帮我们打开三个流指针

```
FILE* stdin   标准输入流指针    从终端文件读取数据时候使用
FILE* stdout  标准输出流指针    将数据打印到终端文件时候使用的流指针
FILE* stderr  标准错误输出流指针
```

【2】标准IO函数

```
fopen   /  fclose
fprintf /  fscanf
fputc   /  fgetc
fputs   /  fgets
fwrite  /  fread
fseek
```

1）fopen

```
功能：打开一个文件：
头文件：
       #include <stdio.h>

       FILE *fopen(const char *pathname, const char *mode);
参数：
    char *pathname：需要打开的文件的路径以及文件名；
    char *mode：打开方式
       r       以读的方式打开文件，
                 如果文件不存在，则函数运行失败；
```

```
                如果文件存在，则打开成功；

        r+      以读写的方式打开文件，
                如果文件不存在，则函数运行失败；
                如果文件存在，则打开成功

        w       以写的方式打开文件，
                如果文件不存在，则创建文件；
                如果文件存在，则清空文件；

        w+      以读写的方式打开文件，
                如果文件不存在，则创建文件；
                如果文件存在，则清空文件；

        a       以追加的方式写；
                如果文件不存在，则创建文件；
                如果文件存在，则以追加的方式写入；

        a+      以读以及追加写的方式打开文件；
                如果文件不存在，则创建文件；
                如果文件存在，则以追加的方式写入;或者从文件开头读取；

返回值：
    成功，返回流指针；
    失败，返回NULL，同时更新错误码errno；

 不同的错误，代码会设置不同的错误编号errno是不同的；
```

2）perror

```
功能：通过errno打印对应的错误信息；
头文件：
        #include <stdio.h>

        void perror(const char *s);
参数：
    char *s：用于提示的字符串；

        #include <errno.h>
        int errno;       /* Not really declared this way; see errno(3) */

vim /usr/include/asm-generic/errno-base.h
vim /usr/include/asm-generic/errno.h
```

3）fclose

功能：关闭文件；
头文件：

```
#include <stdio.h>

int fclose(FILE *stream);
```

参数：
FILE *stream：指定要关闭的文件的流指针；
返回值：
成功，返回0；
失败，返回EOF(其实就是-1)，更新errno；

4）fprintf

功能：将数据格式化输出到文件中；
头文件：

```
#include <stdio.h>

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
```

参数：
FILE *stream：指定要输出到哪个文件中，则填对应的流指针.
const char *format：标准格式化；
...:不定参数，不定数据类型不定数据个数；
返回值：
成功，返回被打印的字节个数，其实就是大于0；
失败，返回负数，其实就是小于0；

5）fscanf

功能：从指定文件中格式化输入数据到内存中；
头文件：

```
#include <stdio.h>

int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
```

参数：
FILE *stream：指定要从哪个文件中读取数据，则填对应的流指针.
const char *format：标准格式化；
...:不定参数，不定数据类型不定数据个数；
返回值：
成功，返回成功读取到的数据个数，注意不是字节数；
失败，或者读取到文件结尾，返回EOF，同时更新errno；

练习

> 自行写一个usr.txt文件，文件中每一行都存储一个用户名和密码，中间用空格隔开，要求：
>
> 1. 从终端获取用户名和密码
> 2. 与文件存储的用户名密码比较
> 3. 如果用户名不存在，则输出用户不存在

> 4. 如果密码错误，则输出密码错误
> 5. 如果均正确，则输出登录成功

```c
#include <stdio.h>
#include <string.h>

int main(int argc, const char *argv[])
{
    FILE* fp = fopen("./usr.txt", "r");
    if(NULL == fp)
    {
        perror("fopen");
        return -1;
    }

    char name[10] = "";
    char passwd[10] = "";
    printf("请输入用户名和密码>>>");
    scanf("%s %s", name, passwd);

    char f_name[10] = "";
    char f_passwd[10] = "";
    int ret = 0;

    while(1)
    {
        ret =fscanf(fp, "%s %s", f_name, f_passwd);
        if(ret < 0)
        {
            printf("用户名不存在\n");
            break;
        }
        //  printf("f_name = %s f_passwd = %s\n", f_name, f_passwd);

        //比较用户名
        if(strcmp(name, f_name) != 0)
            continue;       //如果用户名不相等，则直接读取下一行，不需要比较密码

        //能运行到当前位置，则说明用户名相同
        //则需要去比较密码
        if(strcmp(passwd, f_passwd) == 0)
        {
            printf("登录成功\n");
        }
        else
        {
            printf("密码输入错误\n");
        }

        break;
    }

    fclose(fp);
    return 0;
}
```

6）fputc

功能：向指定文件中输出单个字符；
头文件：

```
#include <stdio.h>

int fputc(int c, FILE *stream);
int putchar(int c);  putchar('a'); putchar(10); putchar(97);
```

参数：
int c：指定要输出的字符对应的字符形式，或者整型，例如‘a’,或者97
FILE *stream：指定要输出到哪个文件中，填对应的流指针；
返回值：
成功，返回成功输出的字符对应的整型形式；
失败，返回EOF(-1);

7）fgetc

功能：从文件中获取单个字符；
头文件：

```
#include <stdio.h>

int fgetc(FILE *stream);
int getchar(void);       char c = getchar();
```

参数：
FILE *stream：指定要从哪个文件中读取数据，则填对应的流指针.
返回值：
成功，返回成功读取到的字符对应的整型；
失败或者读取到文件结尾，返回EOF;

```
#include <stdio.h>

int main(int argc, const char *argv[])
{
    //打开一个文件，以读的方式打开文件
    FILE* fp = fopen("./01_fopen.c", "r");
    if(NULL == fp)
    {
        perror("fopen");
        return -1;
    }

    char c ;
    while(1)
    {
        c = fgetc(fp);
        if(EOF == c)
            break;

        printf("%c", c);
    }

    fclose(fp);

    return 0;
```

```
}
```

练习

> 1. 要求用fputc和fgetc拷贝一个文件，例如将01.c的内容拷贝到02.c中

```
#include <stdio.h>

#define ERR_MSG(msg)  do{\
    printf("line:%d\n", __LINE__); \
    perror(msg);\
}while(0)


int main(int argc, const char *argv[])
{
    if(argc < 2)
    {
        printf("请外部传参输入要拷贝的文件名\n");
        return -1;
    }

    //打开一个文件，以读的方式打开文件
    FILE* fp = fopen(argv[1], "r");
    if(NULL == fp)
    {
        //printf("%d %s %s\n", __LINE__, __func__, __FILE__);
        ERR_MSG("fopen");
        return -1;
    }

    //以写的方式打开目标文件
    FILE* fp_w = fopen("./copy.c", "w");
    if(NULL == fp_w)
    {
        ERR_MSG("fopen");
        return -1;
    }

    char c ;

    while((c=fgetc(fp)) != EOF)
    {
        fputc(c, fp_w);
    }
    printf("拷贝完毕\n");

    fclose(fp);

    return 0;
}
```

> Linux操作系统，用编辑器打开文件保存退出后，会自动补上一个'\n'
>
> windows操作系统，用编辑器打开文件保存退出后，会自动补上一个'\r'
>
> unix操作系统，用编辑器打开文件保存退出后，会自动补上一个'\r\n'

作业

> 1. 用fgetc实现，计算一个文件有几行，要求封装成函数，用命令行传参，

8）缓冲区

注意只有标准IO才有缓冲区，文件IO是没有的。

i.全缓冲

操作对象

对普通文件进行从操作：用fopen函数手动打开的文件，创建的缓冲区全部都是全缓冲。

大小：

4096byte = 4k

```
#include <stdio.h>

int main(int argc, const char *argv[])
{
    FILE* fp = fopen("fullBuf.txt", "w");
    if(NULL == fp)
    {
        perror("fopen");
        return -1;
    }

    fputc('a', fp);
    //由于操作系统优化，只申请不操作的话不会真正的把缓冲区申请出来。
    printf("%ld\n", fp->_IO_buf_end - fp->_IO_buf_base);

    fclose(fp);
    return 0;
}
```

缓冲区的刷新机制

1. 缓冲区满
2. 用fflush函数强制刷新

```
        #include <stdio.h>

        int fflush(FILE *stream);

fflush(fp);
```

3. 调用fclose关闭流指针
4. 主函数调用return退出程序
5. 调用exit函数退出程序

```
功能：退出进程；
        #include <stdlib.h>

        void exit(int status);
参数：
    int status：进程退出状态值，目前随便填一个整型数即可；

exit(0);
```

ps：

1. 读写转换。

ii.行缓冲

操作对象：

标准输入流指针（stdin）　标准输出流指针(stdout)

大小：

1024byte = 1K

```
#include <stdio.h>

int main(int argc, const char *argv[])
{
    //fprintf(stdout, "hello world\n");
    printf("hello world\n");
    printf("%ld\n", stdout->_IO_buf_end - stdout->_IO_buf_base);
    return 0;
}
```

缓冲区的刷新机制

1. 缓冲区满
2. 用fflush函数强制刷新

```
        #include <stdio.h>

        int fflush(FILE *stream);

fflush(stdout);
```

3. 调用fclose关闭流指针
4. 主函数调用return退出程序
5. 调用exit函数退出程序

```
功能：退出进程；
        #include <stdlib.h>

        void exit(int status);
参数：
    int status：进程退出状态值，目前随便填一个整型数即可；

exit(0);
```

6. 遇到'\n'字符刷新缓冲区;

ps：

1. 读写转换。

iii.无缓冲

操作对象：

标准错误输出流指针(stderr)

大小：0

9）fputs

```
功能：将字符串输出到指定的文件中；
头文件：
        #include <stdio.h>

        int fputs(const char *s, FILE *stream);
参数：
    char *s：指定要输出的字符串首地址；
    FILE *stream：；
返回值：
    成功，返回非负数；
    失败，返回EOF；
```

10）fgets

```
功能：从指定的文件中获取字符串；
头文件：
        #include <stdio.h>

        char *fgets(char *s, int size, FILE *stream);
参数：
    char *s：该指针指向的内存空间会存储获取到的字符串；
```

```
    int size：指定要获取多少个字节的数据;最多会读取size-1个字节;
                期间遇到新的一行或者文件结尾会停止读取;在有效字符串的结尾会自动补上'\0';
    FILE *stream:;
返回值：
    成功，返回字符串首地址，其实就是s参数;
    失败或者文件读取完毕，返回NULL;
```

1. 用fgets和fputs实现文件的拷贝
2. 用fgets实现计算一个文件有几行。