1.标准IO

1.1IO错误码问题

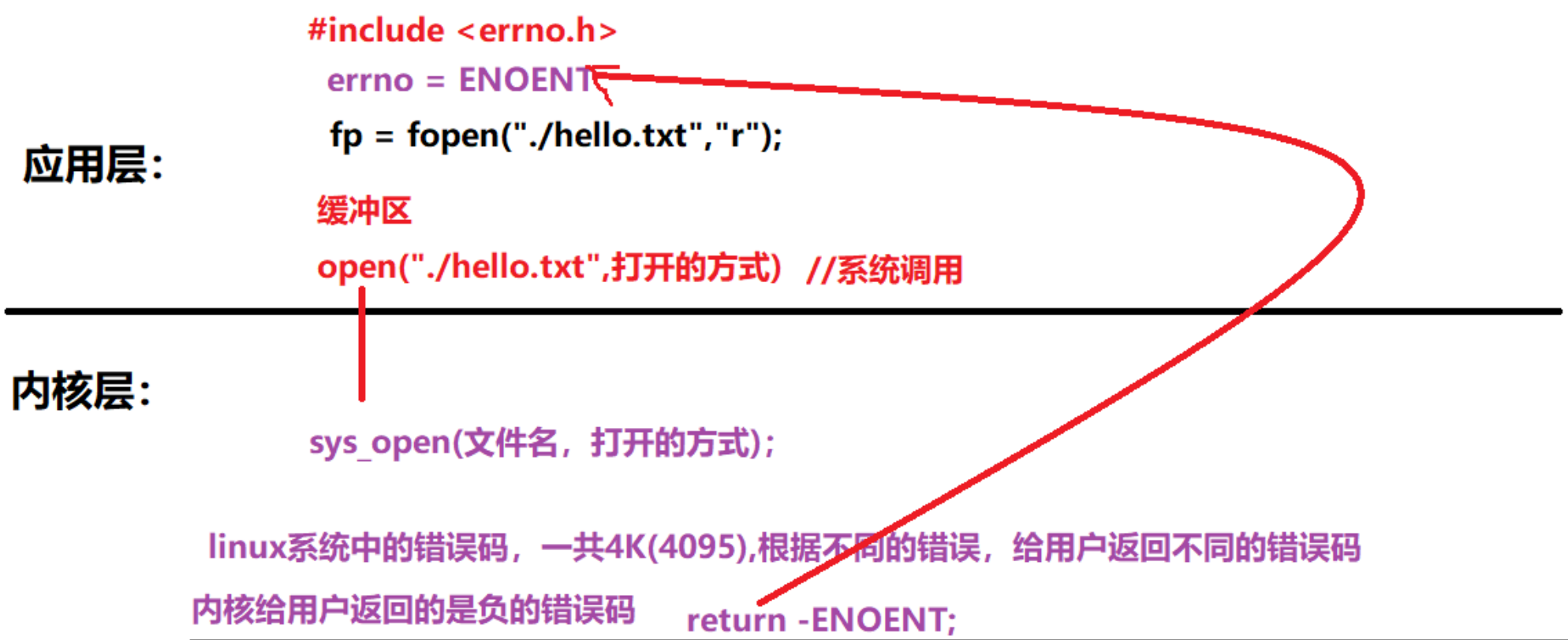
1.1.1错误码工作原理

当用户调用fopen库函数时，会调用open这个系统调用，进入内核空间，在内核空间

中sys_open就会对文件的名字，打开方式等进行检查，如果有问题就返回对应的错误码。

用户如果想要接收错误码就必须包含#include <errno.h>头文件，在这个头文件中会有

一个errno的变量，这个变量记录的就是错误码的值。




```
#define EPERM        1  /* Operation not permitted */
#define ENOENT       2  /* No such file or directory */
#define ESRCH        3  /* No such process */
#define EINTR        4  /* Interrupted system call */
#define EIO      5  /* I/O error */
#define ENXIO        6  /* No such device or address */
#define E2BIG        7  /* Argument list too long */
#define ENOEXEC      8  /* Exec format error */
#define EBADF        9  /* Bad file number */
#define ECHILD      10  /* No child processes */
#define EAGAIN      11  /* Try again */
#define ENOMEM      12  /* Out of memory */
#define EACCES      13  /* Permission denied */
```

1.1.2错误码数值获取实例

```c
#include <stdio.h>
#include <errno.h>
int main(int argc,const char * argv[])
{
    FILE *fp;
    // 因为./hello.txt不存在，所以以只读的方式打开文件的时候会错误
    // 此时fp接收的是NULL,errno的结果是2,
    //2对应的错误信息就是：没有这个文件或目录
    if((fp = fopen("./hello.txt","r"))==NULL){
        printf("fopen error,errno = %d\n",errno);
        return -1;
    }
    return 0;
}
```

1.1.3错误码转错误信息函数

```c
#include <string.h>

char *strerror(int errnum);
功能：将错误码，转换为错误信息（字符串）
参数：
    @errnum:错误码
返回值：成功返回的是错误信息，如果错误码没有置位（0）success，错误位置的错误信息
```

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

int main(int argc,const char * argv[])
{
    FILE *fp;
```

```
8
9       if((fp = fopen("./hello.txt","r"))==NULL){
10          printf("fopen error,errno = %d\n",errno);
11          //strerror(errno)；这个函数会将错误码，转换为错误信息
12          printf("errmsg:%s\n",strerror(errno));
13          return -1;
14      }
15      return 0;
16  }
```

1.1.4perror将错误信息打印到终端

```
1  #include <stdio.h>
2
3  void perror(const char *s);
4  功能：将错误信息打印到终端上（stderr）
5  参数：
6      @s:用户附加的信息
7  返回值：无
```

```
1  #include <stdio.h>
2
3  int main(int argc,const char * argv[])
4  {
5      FILE *fp;
6
7      if((fp = fopen("./hello.txt","r"))==NULL){
8          perror("fopen error");
9          return -1;
10     }
11     return 0;
12 }
```

注：将上述的打印封装成宏，放在head.h中，如何使用head.h

1.将head.h文件放到/usr/include/

2.修改/usr/include/head.h的用户名和组名

sudo chown linux\:linux head.h

3.在自己的.c中包含 #include <head.h>

4.修改代码片段补全(new)

设置->配置用户代码片段->c.json(c)->将原来的stdio.h改成head.h

```
1  #include <head.h>
2
3  int main(int argc, const char* argv[])
4  {
5      FILE* fp;
6      if ((fp = fopen("./hello.txt", "r")) == NULL)
7          PRINT_ERR("fopen error");
8
9      return 0;
10 }
```

1.2标准IO缓冲区问题

1.2.1缓冲区的种类

缓冲区一共有三种：全缓存，行缓存，无缓存。

全缓存：和文件相关的缓冲区都是全缓存。

行缓存：和终端相关的缓冲区都是行缓存。

无缓存：标准出错stderr无缓存

1.2.2缓冲区的大小

全缓存：4K(4096)

行缓存：1k(1024)

无缓存：0

```
1  #include <head.h>
2
3  int main(int argc, const char* argv[])
4  {
5      //  1.行缓存-stdin (1K)
6      // 缓冲区在验证的时候，必须使用，只有使用了才会分配缓冲区，行缓存的大小是1k(1024)
7      fgetc(stdin);
```

```c
    printf("stdin size  = %ld\n", stdin->_IO_buf_end - stdin->_IO_buf_base);

    //  2.行缓存-stdout(1K)
    printf("stdout size  = %ld\n", stdout->_IO_buf_end - stdout->_IO_buf_base);

    //  3.全缓存-fp(4K)
    FILE* fp;
    if ((fp = fopen("./hello.txt", "w")) == NULL)
        PRINT_ERR("fopen error");
    fputc('t', fp);
    printf("fp size  = %ld\n", fp->_IO_buf_end - fp->_IO_buf_base);
    fclose(fp);

    // 4.无缓存-stderr
    //  stderr在打印大小的时候结果是1，但是你向这个空间中写一个字符的时候，
    // 它认为写满了直接就会输出这个字符，所以认为这个缓冲区的大小是0
    fputc('t', stderr); // 向标准出错中写一个字符
    printf("stderr size  = %ld\n", stderr->_IO_buf_end - stderr->_IO_buf_base);
    return 0;
}
```

1.2.3缓冲区的刷新方式

行缓存刷新方式：

```c
#include <head.h>

int main(int argc,const char * argv[])
{
    // 1.行缓存遇到换行符会刷新缓冲区
    // printf("111111\n");
    // while(1);

    // 2.当程序结束的时候会刷新行缓存
    // printf("111111");

    // 3.当关闭文件指针的时候会刷新行缓存
    // printf("11111111");
    // fclose(stdout);
    // while(1);

    // 4.当输入和输出发生切换的时候会刷新行缓存
    // printf("11111111");
    // int num;
    // scanf("%d",&num);
    // while(1);

    // 5.行缓存满1k也会刷新缓冲区
    // for(int i=0;i<1025;i++){
    //     fputc('h',stdout);
    // }
    // while(1);

    // 6.主动刷新缓冲区
    printf("1111111");
    fflush(stdout); //主动刷新缓冲区
    while(1);
    return 0;
}
```

全缓存刷新方式：

```c
#include <head.h>

int main(int argc,const char * argv[])
{
    FILE *fp;
    if((fp = fopen("./hello.txt","w+"))==NULL)
        PRINT_ERR("fopen error");

    // 1.全缓存遇到换行符不会刷新缓冲区
    // fputc('t',fp);
    // fputc('\n',fp);
    // while(1);

    // 2.当程序结束的时候会刷新全缓存
    // fputc('G',fp);

    // 3.当关闭文件指针的时候会刷新全缓存
    // fputc('G',fp);
    // fclose(fp);
    // while(1);

```

```
    // 4.当输入和输出发生切换的时候会刷新全缓存
    // fputc('L',fp);
    // fgetc(fp);
    // while(1);

    // 5.全缓存满4k也会刷新缓冲区
    // for(int i=0;i<4097;i++){
    //     fputc('b',fp);
    // }
    // while(1);

    // 6.主动刷新缓冲区
    fputc('s',fp);
    fflush(fp); //主动刷新缓冲区
    while(1);
    return 0;
}
```

1.3fgets函数

1.3.1fgets函数的功能

```
#include <stdio.h>
char *fgets(char *s, int size, FILE *stream);
功能：从stream对应的文件中读字符串到s中，最多读取size-1个字符。
    读的时候遇到EOF、换行、size-1会停止。如果遇到换行符停止的，
    换行符被存到了s中。
参数：
    @s：存放数据的首地址
    @size:读取的字节数（size-1）
    @stream:文件指针
返回值：成功返回s，系统错误或者读到文件的结尾返回NULL
```

1.3.2fgets函数的实例

fgets读文件实例：

```
#include <head.h>

int main(int argc, const char* argv[])
{
    FILE* fp;
    if ((fp = fopen("./hello.txt", "r")) == NULL)
        PRINT_ERR("fopen error");

    char s[30];          // 如果hello.txt文件每一行小于30字节
    fgets(s, 30, fp);    // 读取了第一行
    printf("s = %s", s); // 将第一行打印到终端上
    fgets(s, 30, fp);    // 读取第二行
    printf("s = %s", s); // 将第二行打印到终端上

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

fgets读标准输入实例：

```
#include <head.h>

int main(int argc, const char* argv[])
{
    char s[10];

    // 从终端上读取字符串到s中
    fgets(s, 10, stdin);
    // 如果'\0'前一个字符是'\n'就清零，否则就不清零
    if (s[strlen(s) - 1] == '\n')
        s[strlen(s) - 1] = '\0';

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

    return 0;
}
```

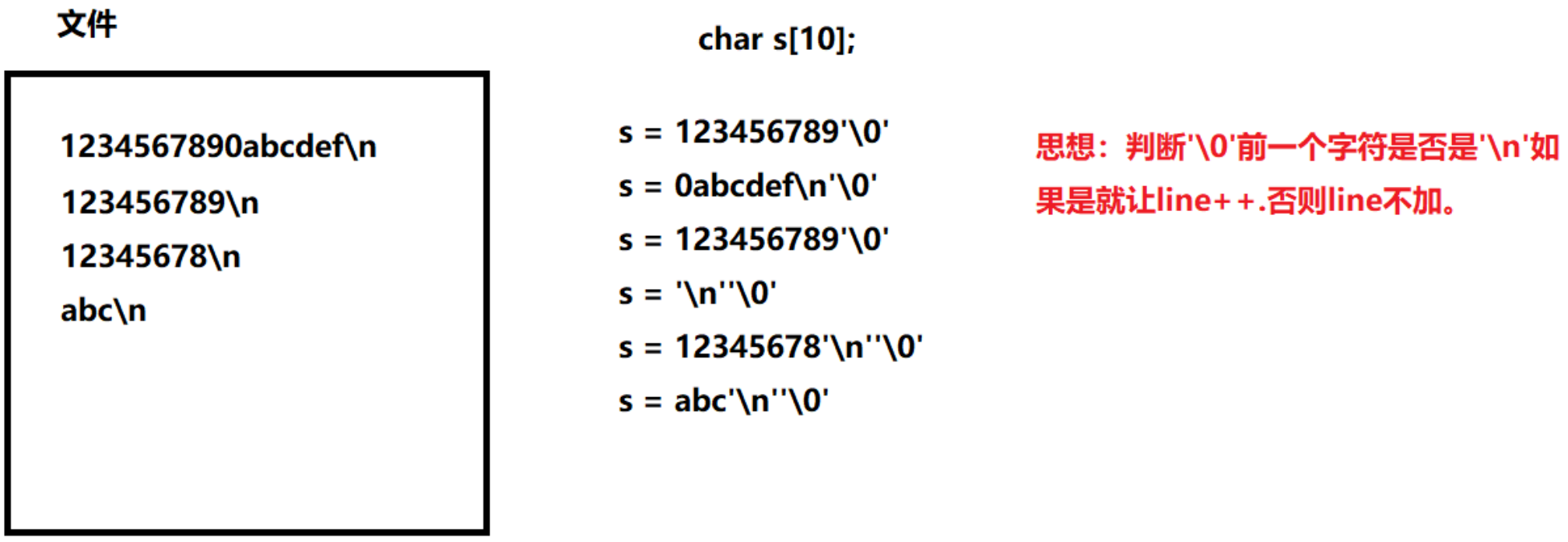
1.3.3fgets函数的练习

练习：使用fgets统计个输入文件的行号，文件的名字通过fgets输入。

./a.out

请输入文件名字 > hello.txt

hello.txt文件的行号 = 10




```c
#include <head.h>
#define N 20

int main(int argc, const char* argv[])
{
    char sn[N],s[N];
    char ch;
    FILE* fp;
    int line = 0;
    // 1.输入文件名
retry:
    printf("请输入文件名 > ");
    if (fgets(sn, N, stdin) == NULL) { //ctrl + d 给程序发送一个EOF
        printf("输入错误，请重试...\n");
        // while ((ch = getchar()) != '\n' && (ch != EOF));
        goto retry;
    }

    if (sn[strlen(sn) - 1] == '\n') {
        sn[strlen(sn) - 1] = '\0';
    } else {
        printf("输入的文件名太长了，请重试...\n");
        while ((ch = getchar()) != '\n' && (ch != EOF));
        goto retry;
    }

    // 2.打开文件
    if ((fp = fopen(sn, "r")) == NULL)
        PRINT_ERR("fopen error");
    // 3.循环读文件
    while (fgets(s, N, fp) != NULL) {
        if (s[strlen(s) - 1] == '\n')
            line++;
    }
    // 4.输出行号
    printf("%s的行号是=%d\n",sn,line);

    // 5.关闭文件
    fclose(fp);
    return 0;
}
```

1.4fputs函数

1.4.1fputs函数的功能

```c
int fputs(const char *s, FILE *stream);
功能：从s中取字符串写入到文件中，遇到'\0'停止，'\0'不会被写到文件中
参数：
    @s:被写数据的首地址
    @stream:文件指针
返回值：成功返回非负数，失败返回EOF(-1)
```

1.4.2fputs函数的实例

```c
#include <head.h>

int main(int argc,const char * argv[])
{
    // fputs可以向标准输出和标准错误中写数据
```

```
    // fputs("stdout:hello fputs func\n",stdout);
    // fputs("stderr:hello fputs func\n",stderr);

    FILE *fp;

    if((fp = fopen("./hello.txt","w"))==NULL)
        PRINT_ERR("fopen error");

    char s[] = "hello everyone!!!!";
    printf("fputs = %d\n",fputs(s,fp));

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

1.4.3fputs函数的练习

练习：使用fgets、fputs实现文件的拷贝

./a.out srcfile destfile

```
#include <head.h>
// ./a.out srcfile destfile
int main(int argc, const char* argv[])
{
    FILE *fp1, *fp2;
    char s[10];
    int line = 0;
    // 1.校验命令行参数个数
    if (argc != 3) {
        printf("input error,try again\n");
        printf("usage: ./a.out srcfile destfile\n");
        return -1;
    }
    // 2.以只读的方式打开源文件，以只写方式打开目标文件
    if ((fp1 = fopen(argv[1], "r")) == NULL)
        PRINT_ERR("fopen src error");

    if ((fp2 = fopen(argv[2], "w")) == NULL)
        PRINT_ERR("fopen dest error");

    // 3.循环拷贝
    while (fgets(s,10,fp1)!=NULL) {
        fputs(s, fp2);
    }

    // 4.关闭文件
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

1.5fread函数

1.5.1fread函数的功能

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
功能：从文件中读取nmemb项数据，每一项的大小是size(字节)，写到ptr
参数：
    @ptr:读到数据的首地址
    @size:大小
    @nmemb:项目个数
    @stream:文件指针
返回值：成功返回项目的个数，如果是失败或者到文件结尾返回小于项目的个数
        int feof(FILE *stream);
        //如果读取到了文件的结尾，feof(fp)会返回真，说明到达了文件的结尾
        int ferror(FILE *stream);
        //如果读取文件的时候发生了错误，ferror(fp)会返回真，说明读错误了
```

1.5.2fread函数的实例

```
#include <head.h>
typedef struct{
    char name[20];
    int age;
    char sex;
}stu_t;
int main(int argc,const char * argv[])
{
    FILE *fp;
    if((fp = fopen("./hello.txt","r"))==NULL)
```

```
        PRINT_ERR("fopen error");

    // 1.读取整数
    // int num;
    // fread(&num,sizeof(num),1,fp);
    // printf("num = %d\n",num);

    // 2.读取字符串
    // char s[100] = {0};
    // fread(s,sizeof(char),sizeof(s),fp);
    // printf("s = %s\n",s);

    // 3.从文件读取结构体
    stu_t stu;
    fread(&stu,sizeof(stu),1,fp);
    printf("name=%s,age=%d,sex=%c\n",stu.name,stu.age,stu.sex);

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

1.5.3使用fread/fwrite拷贝文件

./a.out srcfile destfile

正确写法：

```
#include <head.h>
// ./a.out srcfile destfile
int main(int argc, const char* argv[])
{
    FILE *fp1, *fp2;
    char s[20] = {0};
    int line = 0, ret;
    // 1.校验命令行参数个数
    if (argc != 3) {
        printf("input error,try again\n");
        printf("usage: ./a.out srcfile destfile\n");
        return -1;
    }
    // 2.以只读的方式打开源文件，以只写方式打开目标文件
    if ((fp1 = fopen(argv[1], "r")) == NULL)
        PRINT_ERR("fopen src error");

    if ((fp2 = fopen(argv[2], "w")) == NULL)
        PRINT_ERR("fopen dest error");

    // 3.循环拷贝
    while (!(feof(fp1) || ferror(fp1))) {
        ret = fread(s, 1, sizeof(s), fp1);
        fwrite(s, 1, ret, fp2);
    }

    // 4.关闭文件
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

只能拷贝文本文件写法：

```
#include <head.h>
// ./a.out srcfile destfile
int main(int argc, const char* argv[])
{
    FILE *fp1, *fp2;
    char s[21] = {0};
    int line = 0, ret;
    // 1.校验命令行参数个数
    if (argc != 3) {
        printf("input error,try again\n");
        printf("usage: ./a.out srcfile destfile\n");
        return -1;
    }
    // 2.以只读的方式打开源文件，以只写方式打开目标文件
    if ((fp1 = fopen(argv[1], "r")) == NULL)
        PRINT_ERR("fopen src error");

    if ((fp2 = fopen(argv[2], "w")) == NULL)
        PRINT_ERR("fopen dest error");

    // 3.循环拷贝(只能拷贝文本文件，不能拷贝二进制文件)
    while (!(feof(fp1) || ferror(fp1))) {
```

```
        memset(s,0,sizeof(s));
        fread(s, 1, sizeof(s)-1, fp1);
        fwrite(s, 1, strlen(s), fp2);
    }

    // 4.关闭文件
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

1.6fwrite函数

1.6.1fwrite函数的功能

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb,FILE *stream);
功能：将ptr中的数据写入到文件中，写nmemb项，每一项的大小是size
参数：
    @ptr:被写数据的首地址
    @size:大小
    @nmemb:项目个数
    @stream:文件指针
返回值：成功返回写入项目的个数，失败返回小于项目的个数
```

1.6.2fwrite函数的实例

```
#include <head.h>
typedef struct{
    char name[20];
    int age;
    char sex;
}stu_t;
int main(int argc,const char * argv[])
{
    FILE *fp;
    if((fp = fopen("./hello.txt","w"))==NULL)
        PRINT_ERR("fopen error");

    // 1.向文件中写整数
    // int num=12345;
    // fwrite(&num,sizeof(num),1,fp);

    // 2.向文件中写字符串
    // char s[] = "hello DC23032 everyone!!!!";
    // fwrite(s,sizeof(char),strlen(s),fp);

    // 3.向文件写入结构体
    stu_t stu = {
        .name = "zhangsan",
        .age = 22,
        .sex = 'm',
    };
    fwrite(&stu,sizeof(stu),1,fp);

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

1.6.3使用fread/fwrite拷贝文件

实例见1.5.3标题

1.7sprintf/snprintf/fprintf函数

1.7.1snprintf函数功能

```
int snprintf(char *str, size_t size, const char *format, ...);
功能：将format控制格式中的内容写入到str中
参数：
    @str:字符数组的首地址
    @size:大小（size-1）
    @format:和printf参数一样
返回值：成功返回大于0的数，失败返回负数
```

1.7.2snprintf使用实例

```
#include <head.h>

int main(int argc,const char * argv[])
{
```

```c
    time_t ts;
    struct tm *tm;
    char tim[50] = {0};
    if((ts = time(NULL))==-1)
        PRINT_ERR("time error");

    if((tm = localtime(&ts))==NULL)
        PRINT_ERR("localtime error");

    snprintf(tim,sizeof(tim),"%d-%02d-%02d %02d:%02d:%02d\n",
    tm->tm_year+1900,tm->tm_mon+1,tm->tm_mday,tm->tm_hour,tm->tm_min,tm->tm_sec);

    fputs(tim,stdout);
    return 0;
}
```

2.作业

2.1作业要求

将当前的时间写入到time.txt的文件中，如果ctrl+c退出之后，在再次执行支持断点续写

1.2022-03-25 19:10:20

2.2022-03-25 19:10:21

3.2022-03-25 19:10:22

//按下ctrl+c停止，再次执行程序

4.2022-03-25 20:00:00

5.2022-03-25 20:00:01

2.2获取系统时间的API

```c
#include <time.h>

time_t time(time_t *tloc);
功能：获取自1970-01-01 00:00:00到当前时间的秒钟数
参数：
    @tloc:NULL
返回值：成功返回秒钟数，失败返回-1置位错误码

struct tm *localtime(const time_t *timep)
功能：将秒钟数准换为年月日，时分秒等存储到tm的结构体中
参数：
    @timep:秒钟数变量地址
返回值：成功返回tm结构体指针，失败返回NULL，置位错误码
        struct tm {
            int tm_sec;    /* Seconds (0-60) */
            int tm_min;    /* Minutes (0-59) */
            int tm_hour;   /* Hours (0-23) */
            int tm_mday;   /* Day of the month (1-31) */
            int tm_mon;    /* Month (0-11) */ //month+1
            int tm_year;   /* Year - 1900 */  //year+1900
            int tm_wday;   /* Day of the week (0-6, Sunday = 0) */
            int tm_yday;   /* Day in the year (0-365, 1 Jan = 0) */
            int tm_isdst;  /* 夏令时，被废弃了*/
        };

```

2.3获取系统时间的实例

```c
#include <head.h>

int main(int argc,const char * argv[])
{
    time_t ts;
    struct tm *tm;
    if((ts = time(NULL))==-1)
        PRINT_ERR("time error");

    if((tm = localtime(&ts))==NULL)
        PRINT_ERR("localtime error");

    printf("%d-%02d-%02d %02d:%02d:%02d\n",
    tm->tm_year+1900,tm->tm_mon+1,tm->tm_mday,tm->tm_hour,tm->tm_min,tm->tm_sec);

    return 0;
}
```

2.4作业的代码实现

```c
#include <head.h>
int get_file_line(FILE* fp)
{
    int line=0;
    char s[30];
    // 循环读文件
    while (fgets(s, 30, fp) != NULL) {
        if (s[strlen(s) - 1] == '\n')
            line++;
    }
    return line;
}
int main(int argc, const char* argv[])
{
    time_t ts, ots;
    struct tm* tm;
    char tim[50] = { 0 };
    FILE* fp;
    int line = 0;
    if ((fp = fopen("./time.txt", "a+")) == NULL)
        PRINT_ERR("fopen error");

    line = get_file_line(fp);

    ts = ots = 0;
    while (1) {
        if ((ts = time(NULL)) == -1)
            PRINT_ERR("time error");
        if (ts != ots) {
            ots = ts;
            if ((tm = localtime(&ts)) == NULL)
                PRINT_ERR("localtime error");

            snprintf(tim, sizeof(tim), "%d.%d-%02d-%02d %02d:%02d:%02d\n",
                line++, tm->tm_year + 1900, tm->tm_mon + 1, tm->tm_mday,
                tm->tm_hour, tm->tm_min, tm->tm_sec);
            fputs(tim, fp);
            fflush(fp);
        }
    }

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