1.标准IO

- 1.1IO错误码问题
 - 1.1.1错误码工作原理
 - 1.1.2错误码数值获取实例
 - 1.1.3错误码转错误信息函数
 - 1.1.4perror将错误信息打印到终端

1.2标准IO缓冲区问题

- 1.2.1缓冲区的种类
- 1.2.2缓冲区的大小
- 1.2.3缓冲区的刷新方式

1.3fgets函数

- 1.3.1fgets函数的功能
- 1.3.2fgets函数的实例
- 1.3.3fgets函数的练习

1.4fputs函数

- 1.4.1fputs函数的功能
- 1.4.2fputs函数的实例
- 1.4.3fputs函数的练习

1.5fread函数

- 1.5.1fread函数的功能
- 1.5.2fread函数的实例
- 1.5.3使用fread/fwrite拷贝文件

1.6fwrite函数

- 1.6.1fwrite函数的功能
- 1.6.2fwrite函数的实例
- 1.6.3使用fread/fwrite拷贝文件

1.7sprintf/snprintf/fprintf函数

- 1.7.1snprintf函数功能
- 1.7.2snprintf使用实例

2.作业

- 2.1作业要求
- 2.2获取系统时间的API
- 2.3获取系统时间的实例
- 2.4作业的代码实现

1.标准IO

1.110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。

```
#include <errno.h>
errno = ENOENT

fp = fopen("./hello.txt","r");
缓冲区
open("./hello.txt",打开的方式) //系统调用
```

内核层:

sys_open(文件名, 打开的方式);

linux系统中的错误码,一共4K(4095),根据不同的错误,给用户返回不同的错误码

内核给用户返回的是负的错误码 return -ENOENT;

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

1.1.2错误码数值获取实例

```
1 #include <stdio.h>
   #include <errno.h>
3 int main(int argc,const char * argv[])
 4
       FILE *fp;
5
       // 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
6
       // 此时fp接收的是NULL,errno的结果是2,
8
       //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){
9
          printf("fopen error,errno = %d\n",errno);
10
          return -1;
11
12
13
       return 0;
```

1.1.3错误码转错误信息函数

```
      1
      #include <string.h>

      2

      3
      char *strerror(int errnum);

      4
      功能: 将错误码,转换为错误信息(字符串)

      5
      参数:

      6
      @errnum:错误码

      7
      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
```

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

#include <string.h>

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

#include <string.h>

#include <string.h

#include <string
```

```
8
       if((fp = fopen("./hello.txt","r"))==NULL){
9
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
           printf("errmsg:%s\n", strerror(errno));
12
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
      void perror(const char *s);

      3
      voie: 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
```

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

```
#include <head.h>

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

{
    FILE* fp;
    if ((fp = fopen("./hello.txt", "r")) == NULL)
        PRINT_ERR("fopen error");

    return 0;
}
```

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

```
printf("stdin size = %ld\n", stdin->_IO_buf_end - stdin->_IO_buf_base);
 8
9
10
       // 2.行缓存-stdout(1K)
11
       printf("stdout size = %1d\n", stdout->_IO_buf_end - stdout->_IO_buf_base);
12
13
       // 3.全缓存-fp(4K)
14
       FILE* fp;
15
       if ((fp = fopen("./hello.txt", "w")) == NULL)
           PRINT_ERR("fopen error");
16
17
       fputc('t', fp);
       printf("fp size = %ld\n", fp->_IO_buf_end - fp->_IO_buf_base);
18
19
       fclose(fp);
20
       // 4.无缓存-stderr
21
22
       // 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,
23
       // 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
24
25
       printf("stderr size = %ld\n", stderr->_IO_buf_end - stderr->_IO_buf_base);
26
       return 0;
27 }
```

1.2.3缓冲区的刷新方式

行缓存刷新方式:

```
1 #include <head.h>
2
 3 | int main(int argc,const char * argv[])
4
   {
 5
       // 1.行缓存遇到换行符会刷新缓冲区
 6
       // printf("111111\n");
 7
       // while(1);
 8
 9
       // 2.当程序结束的时候会刷新行缓存
       // printf("11111");
10
11
12
       // 3. 当关闭文件指针的时候会刷新行缓存
13
       // printf("1111111");
14
       // fclose(stdout);
       // while(1);
15
16
       // 4. 当输入和输出发生切换的时候会刷新行缓存
17
18
       // printf("11111111");
19
       // int num;
20
       // scanf("%d",&num);
21
       // while(1);
22
23
       // 5.行缓存满1k也会刷新缓冲区
       // for(int i=0;i<1025;i++){
24
25
       // fputc('h',stdout);
       // }
26
27
       // while(1);
28
29
       // 6.主动刷新缓冲区
30
       printf("1111111");
       fflush(stdout); //主动刷新缓冲区
31
32
       while(1);
       return 0;
33
34 }
```

全缓存刷新方式:

```
1 #include <head.h>
3
   int main(int argc,const char * argv[])
4
 5
       FILE *fp;
       if((fp = fopen("./hello.txt","w+"))==NULL)
 6
           PRINT_ERR("fopen error");
8
       // 1.全缓存遇到换行符不会刷新缓冲区
9
       // fputc('t',fp);
10
11
       // fputc('\n',fp);
       // while(1);
12
13
       // 2. 当程序结束的时候会刷新全缓存
14
15
       // fputc('G',fp);
16
       // 3. 当关闭文件指针的时候会刷新全缓存
17
       // fputc('G',fp);
18
19
       // fclose(fp);
       // while(1);
20
21
```

```
22
       // 4. 当输入和输出发生切换的时候会刷新全缓存
23
       // fputc('L',fp);
24
       // fgetc(fp);
25
       // while(1);
26
       // 5.全缓存满4k也会刷新缓冲区
27
28
       // for(int i=0;i<4097;i++){
29
       // fputc('b',fp);
       // }
30
31
       // while(1);
32
33
       // 6.主动刷新缓冲区
34
       fputc('s',fp);
35
       fflush(fp); //主动刷新缓冲区
36
       while(1);
37
       return 0;
38 }
```

1.3fgets函数

1.3.1fgets函数的功能

```
      1
      #include <stdio.h>

      2
      char *fgets(char *s, int size, FILE *stream);

      3
      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。

      4
      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。

      5
      参数:

      6
      参数:

      8
      @size:读取的字节数 (size-1)

      9
      @stream:文件指针

      10
      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:

```
1 #include <head.h>
   int main(int argc, const char* argv[])
 4
   {
       FILE* fp;
       if ((fp = fopen("./hello.txt", "r")) == NULL)
 6
           PRINT_ERR("fopen error");
 7
 8
9
                         // 如果hello.txt文件每一行小于30字节
       char s[30];
       fgets(s, 30, fp); // 读取了第一行
10
11
       printf("s = %s", s); // 将第一行打印到终端上
       fgets(s, 30, fp); // 读取第二行
12
       printf("s = %s", s); // 将第二行打印到终端上
13
14
15
       fclose(fp);
16
       return 0;
17 }
```

fgets读标准输入实例:

```
1 #include <head.h>
3 int main(int argc, const char* argv[])
      char s[10];
 6
7
       // 从终端上读取字符串到s中
8
       fgets(s, 10, stdin);
       // 如果'\0'前一个字符是'\n'就清零,否则就不清零
9
       if (s[strlen(s) - 1] == '\n')
10
11
           s[strlen(s) - 1] = '\setminus 0';
12
       printf("s = %s\n", s);
13
14
15
        return 0;
16 }
```

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

文件

1234567890abcdef\n 123456789\n 12345678\n abc\n char s[10];

s = 123456789'\0'

s = 0abcdef(n')(0')

s = 123456789'\0'

 $s = '\n''\0'$

s = 12345678'\n''\0'

 $s = abc'\n''\0'$

思想:判断'\0'前一个字符是否是'\n'如果是就让line++.否则line不加。

```
1 #include <head.h>
    #define N 20
   int main(int argc, const char* argv[])
 4
 5
 6
        char sn[N],s[N];
        char ch;
       FILE* fp;
8
       int line = 0;
 9
       // 1.输入文件名
10
11 retry:
12
        printf("请输入文件名 > ");
       if (fgets(sn, N, stdin) == NULL) { //ctrl + d 给程序发送一个EOF
13
14
           printf("输入错误, 请重试...\n");
           // while ((ch = getchar()) != '\n' && (ch != EOF));
15
           goto retry;
16
17
       }
18
19
       if (sn[strlen(sn) - 1] == '\n') {
20
           sn[strlen(sn) - 1] = '\setminus 0';
       } else {
21
           printf("输入的文件名太长了,请重试...\n");
22
23
           while ((ch = getchar()) != '\n' && (ch != EOF));
24
           goto retry;
25
       }
26
       // 2.打开文件
27
       if ((fp = fopen(sn, "r")) == NULL)
28
29
           PRINT_ERR("fopen error");
       // 3.循环读文件
30
31
       while (fgets(s, N, fp) != NULL) {
           if (s[strlen(s) - 1] == '\n')
32
33
               line++;
34
       }
       // 4.输出行号
35
        printf("%s的行号是=%d\n",sn,line);
36
37
        // 5.关闭文件
38
39
        fclose(fp);
40
        return 0;
41 }
```

1.4fputs函数

1.4.1fputs函数的功能

```
      1
      int fputs(const char *s, FILE *stream);

      2
      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

      3
      参数:

      4
      @s:被写数据的首地址

      5
      @stream:文件指针

      6
      返回值:成功返回非负数,失败返回EOF(-1)
```

1.4.2fputs函数的实例

```
#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);
 6
        // fputs("stderr:hello fputs func\n",stderr);
 8
 9
        FILE *fp;
10
        if((fp = fopen("./hello.txt","w"))==NULL)
11
12
            PRINT_ERR("fopen error");
13
        char s[] = "hello everyone!!!!";
14
        printf("fputs = %d\n", fputs(s, fp));
15
16
17
        fclose(fp);
18
        return 0;
19 }
```

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

```
1 #include <head.h>
 2 // ./a.out srcfile destfile
 3 | int main(int argc, const char* argv[])
 4
        FILE *fp1, *fp2;
        char s[10];
 6
       int line = 0;
 8
       // 1.校验命令行参数个数
9
       if (argc != 3) {
           printf("input error,try again\n");
10
11
           printf("usage: ./a.out srcfile destfile\n");
12
           return -1;
13
        }
14
       // 2.以只读的方式打开源文件,以只写方式打开目标文件
15
        if ((fp1 = fopen(argv[1], "r")) == NULL)
16
           PRINT_ERR("fopen src error");
17
        if ((fp2 = fopen(argv[2], "w")) == NULL)
18
           PRINT_ERR("fopen dest error");
19
20
        // 3.循环拷贝
21
        while (fgets(s,10,fp1)!=NULL) {
22
23
           fputs(s, fp2);
       }
24
25
       // 4. 关闭文件
26
27
        fclose(fp1);
        fclose(fp2);
28
29
        return 0;
30 }
```

1.5fread函数

1.5.1fread函数的功能

```
1 #include <stdio.h>
2 size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
3 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
5
      @size:大小
      @nmemb:项目个数
8
      @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
9
10
         int feof(FILE *stream);
11
         //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
12
         int ferror(FILE *stream);
13
         //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

```
1 #include <head.h>
2
   typedef struct{
       char name[20];
3
4
       int age;
5
       char sex;
6 }stu_t;
   int main(int argc,const char * argv[])
8
9
       FILE *fp;
10
        if((fp = fopen("./hello.txt","r"))==NULL)
```

```
11
            PRINT_ERR("fopen error");
12
13
        // 1.读取整数
14
        // int num;
        // fread(&num, sizeof(num), 1, fp);
15
16
        // printf("num = %d\n",num);
17
18
       // 2.读取字符串
        // char s[100] = {0};
19
        // fread(s,sizeof(char),sizeof(s),fp);
20
        // printf("s = %s\n",s);
21
22
23
        // 3.从文件读取结构体
24
        stu_t stu;
25
        fread(&stu, sizeof(stu), 1, fp);
        printf("name=%s,age=%d,sex=%c\n",stu.name,stu.age,stu.sex);
26
27
28
        fclose(fp);
29
        return 0;
30 }
```

1.5.3使用fread/fwrite拷贝文件

./a.out srcfile destfile

正确写法:

```
1 #include <head.h>
    // ./a.out srcfile destfile
 3 int main(int argc, const char* argv[])
 4
    {
        FILE *fp1, *fp2;
        char s[20] = \{0\};
 6
        int line = 0, ret;
 7
 8
        // 1.校验命令行参数个数
 9
        if (argc != 3) {
10
            printf("input error,try again\n");
11
            printf("usage: ./a.out srcfile destfile\n");
            return -1;
12
13
        }
14
        // 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)
15
            PRINT_ERR("fopen src error");
16
17
        if ((fp2 = fopen(argv[2], "w")) == NULL)
18
            PRINT_ERR("fopen dest error");
19
20
        // 3.循环拷贝
21
        while (!(feof(fp1) || ferror(fp1))) {
22
23
            ret = fread(s, 1, sizeof(s), fp1);
24
            fwrite(s, 1, ret, fp2);
        }
25
26
        // 4.关闭文件
27
        fclose(fp1);
28
29
        fclose(fp2);
30
        return 0;
31 }
```

只能拷贝文本文件写法:

```
1 #include <head.h>
 2 // ./a.out srcfile destfile
   int main(int argc, const char* argv[])
4
   {
 5
       FILE *fp1, *fp2;
       char s[21] = \{0\};
6
       int line = 0, ret;
       // 1.校验命令行参数个数
8
9
       if (argc != 3) {
10
           printf("input error,try again\n");
           printf("usage: ./a.out srcfile destfile\n");
11
           return -1;
12
13
       }
       // 2.以只读的方式打开源文件,以只写方式打开目标文件
14
       if ((fp1 = fopen(argv[1], "r")) == NULL)
15
           PRINT_ERR("fopen src error");
16
17
18
       if ((fp2 = fopen(argv[2], "w")) == NULL)
19
           PRINT_ERR("fopen dest error");
20
       // 3.循环拷贝(只能拷贝文本文件,不能拷贝二进制文件)
21
       while (!(feof(fp1) || ferror(fp1))) {
22
```

```
23
            memset(s,0,sizeof(s));
24
            fread(s, 1, sizeof(s)-1, fp1);
25
            fwrite(s, 1, strlen(s), fp2);
26
27
28
        // 4.关闭文件
29
        fclose(fp1);
30
        fclose(fp2);
31
        return 0;
32 }
```

1.6fwrite函数

1.6.1fwrite函数的功能

1.6.2fwrite函数的实例

```
1 #include <head.h>
    typedef struct{
        char name[20];
        int age;
 4
 5
        char sex;
   }stu_t;
    int main(int argc,const char * argv[])
8
    {
9
        FILE *fp;
        if((fp = fopen("./hello.txt","w"))==NULL)
10
            PRINT_ERR("fopen error");
11
12
        // 1.向文件中写整数
13
        // int num=12345;
14
        // fwrite(&num, sizeof(num), 1, fp);
15
16
        // 2.向文件中写字符串
17
        // char s[] = "hello DC23032 everyone!!!!";
18
        // fwrite(s,sizeof(char),strlen(s),fp);
19
20
        // 3.向文件写入结构体
21
22
        stu_t stu = {
            .name = "zhangsan",
23
24
            .age = 22,
25
            .sex = 'm',
26
        };
27
        fwrite(&stu, sizeof(stu), 1, fp);
28
        fclose(fp);
29
30
        return 0;
31 }
```

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

```
1int snprintf(char *str, size_t size, const char *format, ...);2功能: 将format控制格式中的内容写入到str中3参数:4@str:字符数组的首地址5@size:大小(size-1)6@format:和printf参数一样7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[])

{
```

```
5
        time_t ts;
 6
        struct tm *tm;
        char tim[50] = \{0\};
 8
        if((ts = time(NULL))==-1)
 9
            PRINT_ERR("time error");
10
11
        if((tm = localtime(&ts))==NULL)
            PRINT_ERR("localtime error");
12
13
        snprintf(tim, sizeof(tim), "%d-%02d-%02d %02d:%02d:%02d\n",
14
        tm->tm_year+1900,tm->tm_mon+1,tm->tm_mday,tm->tm_hour,tm->tm_min,tm->tm_sec);
15
16
17
        fputs(tim,stdout);
        return 0;
18
19 }
```

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

```
#include <time.h>
3
   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
5
6
      @tloc:NULL
7
   返回值:成功返回秒钟数,失败返回-1置位错误码
8
   struct tm *localtime(const time_t *timep)
9
10
   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
11
   参数:
       @timep:秒钟数变量地址
12
13
   返回值:成功返回tm结构体指针,失败返回NULL,置位错误码
14
          struct tm {
              int tm_sec;    /* Seconds (0-60) */
15
              16
              int tm_hour; /* Hours (0-23) */
17
              int tm_mday; /* Day of the month (1-31) */
18
                         /* Month (0-11) */ //month+1
19
              int tm_mon;
              int tm_year; /* Year - 1900 */ //year+1900
20
              int tm_wday; /* Day of the week (0-6, Sunday = 0) */
21
22
              int tm_yday; /* Day in the year (0-365, 1 \text{ Jan} = 0) */
              int tm_isdst; /* 夏令时,被废弃了*/
23
24
          };
25
```

2.3获取系统时间的实例

```
#include <head.h>
    int main(int argc,const char * argv[])
 4
    {
 5
        time_t ts;
 6
        struct tm *tm;
        if((ts = time(NULL))==-1)
 8
            PRINT_ERR("time error");
 9
        if((tm = localtime(&ts))==NULL)
10
             PRINT_ERR("localtime error");
11
12
        printf("%d-%02d-%02d %02d:%02d:%02d\n",
13
        \label{tm-stm_year+1900,tm-stm_mon+1,tm-stm_mday,tm-stm_hour,tm-stm_min,tm-stm_sec);} \\
14
15
16
        return 0;
17 }
```

2.4作业的代码实现

```
1 | #include <head.h>
2 int get_file_line(FILE* fp)
 4
       int line=0;
       char s[30];
6
       // 循环读文件
 7
       while (fgets(s, 30, fp) != NULL) {
           if (s[strlen(s) - 1] == '\n')
8
               line++;
9
10
       }
11
       return line;
12
   int main(int argc, const char* argv[])
13
14
   {
15
       time_t ts, ots;
       struct tm* tm;
16
17
       char tim[50] = { 0 };
       FILE* fp;
18
       int line = 0;
19
       if ((fp = fopen("./time.txt", "a+")) == NULL)
20
           PRINT_ERR("fopen error");
21
22
23
       line = get_file_line(fp);
24
25
       ts = ots = 0;
26
       while (1) {
           if ((ts = time(NULL)) == -1)
27
               PRINT_ERR("time error");
28
29
           if (ts != ots) {
30
               ots = ts;
               if ((tm = localtime(&ts)) == NULL)
31
32
                  PRINT_ERR("localtime error");
33
               34
                  line++, tm->tm\_year + 1900, tm->tm\_mon + 1, tm->tm\_mday,
35
                  tm->tm_hour, tm->tm_min, tm->tm_sec);
36
               fputs(tim, fp);
37
               fflush(fp);
38
39
           }
       }
40
41
       fclose(fp);
42
       return 0;
43
44 }
```