Linux应用程序编程

1. Linux OS 程序运行可分为2部分：内核态、用户态

隔离

用户态　　　　用户程序（应用程序），如 pwd, ps, ls 等命令
　　　　　　　　　　　　　　　　　　　　　　后台运行

内核态　　系统调用→　内核程序，如驱动程序运行内核空间

2. 应用程序与驱动程序区别？

基于OS
- 用户空间
- 逻辑算法
- 单独编译

如 gcc -c *.c

- 内核空间
- 底层硬件操作控制
- 依赖于内核源码进行编译

3. 用户程序通过三种方式运用

① Linux OS向应用层提供接口（系统调用），是Linux应用层进入内核空间之入口。

② 库函数：标准C库函数封装 { 效率 / 使用便利性 } ↗

③ 不局限于编程语言

可通过 C/C++, python, shell, Qt 来写应用程序。

△ 标准文件I/O.

man 3 fopen

1. FILE指针与fopen函数.

标准I/O使用FILE指针作为文件句柄. 与文件I/O文件描述符非常相似.

2. fopen函数原型: FILE *fopen(const char *path, const char *mode);

fclose函数原型: int fclose(FILE *stream);

mode → r, r+, w, w+, a, a+

读文件 -- fread()函数.

3. size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

ptr → 缓冲区; size → 数据单元/块大小; nmemb → 数据单元的个数; stream → 文件句柄 从fopen()来

4. 写文件 fwrite()函数

size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);

5. 读写位置设置. fseek()函数

int fseek(FILE *stream, long offset, int whence);

与lseek函数意义相同.

如 fseek(file, 0, SEEK_SET) 移到文件头部

fseek(file, 100, SEEK_SET) 移偏100字节.

示例：打开一个文件，写入一段文字。

```c
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

int main(void){
    FILE *f=NULL;
    int ret;  char buf[128]={0};
    f=fopen("./test.txt","w+");      // → 可读可写，不太用，会截为0.
    if(NULL==f){
        perror("fopen error\n");
        return 1;
    }
    ret=fwrite("Hello world",1,11,f);   // 1: 一个字节数据; 11: 11个数据
    if(11>ret){
        printf("fwrite error\n");
        fclose(f);
        return 1;
    }
    ret=fseek(f,0,SEEK_SET);
    if(-1==ret){
        perror("fseek error\n");
        fclose(f);
        return 1;
    }
    ret=fread(buf,1,11,f);
    if(11>ret){
        printf("fread error or End-of-file\n");
        fclose(f);
        return 1;}
    printf("fread:%s\n",buf);  fclose(f); return 0; }
```

fread }
fwrite } 发生错误返回之后并不会重置，所以不用 perror

前面有 fopen 才用 perror

int feof() 函数.

判断是否到达文件的尾部

{ 已到达文件末尾 非0, true
{ 如果到达末尾, 0. false

int ferror() 函数

判断是否发生错误.

{ 发生错误 非0 true
{ 没发生错误 0, False.

void clearerr() 函数 清除标志.

void clearerr(FILE *stream)

→ 清除 feof() 与 ferror() 函数捕获的标志.

```
ret=fread(buf,1,11,f);
if(ret<11){ ---------- // 读取错误.
    if(ferror(f)){
        printf("fread error");
        fclose(f);
        return 1;
    }
    else{
        if(feof(f)){
            printf("fread end-of-file\n");
            fclose(f);
            return 1;
        }
    }
}
clearerr(f); //清除标志.
```

// 判断是哪一种错误!

其他相关IO函数.

① int fgetc(FILE *stream); 从流中读取一个字符. 返回.

② int fputc(int c, FILE *stream); 将c值写到流中.

```
FILE *fp;
fp=fopen("./1.txt","r");
if(NULL==fp){
    printf("fail to fopen\n");
    return -1;
}
int c=fgetc(fp);
printf("c=[%c]-%d\n",c,c);
fclose(fp);
return 0;
```

读取一个字符.

读字符

```
int c;
while((c=fgetc(fp))!=EOF){
    printf("c=[%c]-%d\n",c,c);
}
```

全部读出之后相等.

Ascii码=10 表示换行符.

写字符.

```
fp=fopen("","w");
if(fp==NULL)出错-1.
fputc('w',fp);
fputc('h',fp);
fputc('\n',fp);
fputc('0',fp);
return 0
```

3. 一次读取一个字符串.

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

在读取时遇到换行符或文件末尾时停止读取. 在读取内容后加'\0'作为字符串二结尾. 失败返回 NULL.

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

含义: 将 s 指向字符串写到 stream 所指定文件中.

返回成功写入字节数. 失败返回 -1.

```
FILE *fp;
fp = fopen("..", "r");
以( )为准 -1,
char buf[32] = "";  → 最多读取一行内容
fgets(buf, 8, fp);
printf("buf = %s\n", buf);
```

等价于: fputs("welcome\n", fp);
fputs("Nanjing", fp);

5、fread() 函数完成结构体之读入，写出操作。
fwrite()

```c
typedef struct{
    int a;
    int b;
    char c;
}MSG;

FILE *fp;
fp = fopen("..", "w+");   // 清空文件，再写入。
if(fp==NULL){-1};
//向文件中写入一个结构体
MSG msg[4]={1,2,'a',3,4,'b','5,6,'c',7,8,'d'};
fwrite(msg, sizeof(MSG), 4, fp);
rewind(fp); 将文件偏移量设置为起始位置
MSG rcv[4];
fread(rcv, sizeof(MSG), 4, fp);
int i=0;
for(i=0; i<4; i++){
    printf("%d %d %c\n", rcv[i].a, rcv[i].b, rcv[i].c);
}
```

专题一：文件I/O

文件I/O 指的是对文件的输入、输出操作。（对文件的读写操作）

一个通用的IO模型通常包括打开文件、读写文件、关闭文件等基本操作。open(), read(), write(), close() 4个函数。

一个读写文件的例子。

→ man 2 open 查手册。

★ 文件描述符 {0 标准输入；1 标准输出；2 标准错误}

非负整数，文件描述符，的值。如：0,1,2,... 3 第一个未被使用的最小非负整数！

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
int main(void){
    char buff[1024];
    int fd1, fd2;
    int ret;

    fd1 = open("./srcfile", O_RDONLY);   // 打开只读方式
    if(-1 == fd1) return fd1;
    fd2 = open("./destfile", O_WRONLY);  // 打开只写方式
    if(-1 == fd2){ ret = fd2, goto out1; }

    ret = read(fd1, buff, sizeof(buff));
    if(-1 == ret) goto out2;

    ret = write(fd2, buff, sizeof(buff));
    if(-1 == ret) goto out2;

    ret = 0;

out2:
    close(fd2);
out1: close(fd1);
    return ret; }
```

1. 打开文件 open(函数) 打开或创建文件.

函数原型.

```
int open(const char * pathname, int flags);
int open(const char* pathname, int flages, mode_t mode);
```

-1 表示返回值，失败

pathname → 文件路径.

flages → 标志，可进行"|"组合

mode → 权限.

- O_RDONLY } 互斥
- O_WRONLY } 互斥
- O_RDWR
- O_CREAT 文件不存在就创建
- O_EXCL 与 O_CREAT 一起使用.
- O_DIRECTORY
- O_NOFOLLOW.
- O_TMPFILE

（O_CREAT、O_TMPFILE → 有效）

mode 权限

000 000 000 000，如 777，444 等

S U G O

rwx

→ 文件的操作权限.

可以用宏代替，如 S_IRUSR 允许文件所属者读文件

- S_IWUSR 允许文件所属者写文件.
- S_IXUSR 允许文件 …… 执行文件.
- S_IRWXU …… 读、写、执行文件.
- S_IRGRP / S_IWGRP / S_IRWXG / S_IROTH / S_IRWXG / S_IWOTH / S_IRWXO 等等.

可以用"|"进行组合.

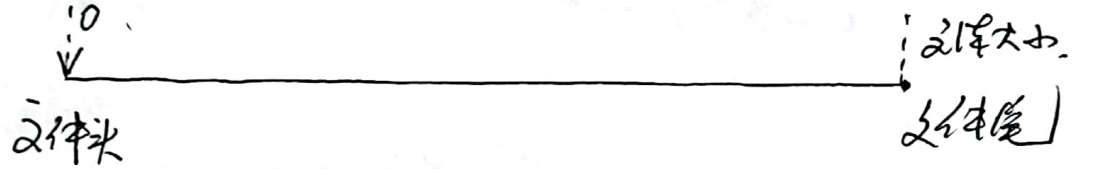
2. 写文件 write() 函数.

文件描述符　写入数据对应缓冲区　写字节数

ssize_t write(int fd, const void *buf, size_t count);

man 2 write　帮助文档. 写出错 -1.

相关: 位置偏移量(读写文件之位置), 读写指针.



☆ 当写文件时, 指针向后偏移, 因此读文件时要注意文件指针所处之位置。可以用 lseek 进行位置调整.

3. 读文件 read() 函数

ssize_t read(int fd, void *buf, size_t count);

返回 long long / long int 数据类型, 读取字节数.　→ 读取字节数.

读出错 返回 -1.

4. 关闭文件: close() 函数.

int close(int fd), 成功0, 失败-1.

应用举例：

- 新建一个文件 test.txt，写入"This is a text file!"。
- 对新建文件的要求 O、G用户对文件只有可读权限、U用户对文件具有可读可写权限
- 读取文件 Test.txt 内容，并打印出来。

代码如下：

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int main(void){          char buf[20] = "I ...";
  int fd;
  int ret;
打开 → fd = open("./test.txt", O_WRONLY|O_CREAT|O_EXCL, 0644)   ← 如果文件存在，则出错！
  if(-1==fd){
      printf("打开异常\n");
      return -1;}
写入 → ret = write(fd, "This is a text file!", 20);   (20 → strlen(buff))
  if(ret==-1){
      printf("write error\n");   → perror("write 失败");
      close(fd);
      return 1;
  }
  printf("write succeful, %byte\n", ret);
关闭 → close(fd);
  return 0;
}
```

手册 { man 2 open/close/write
　　 { man 3 printf

编译：gcc -o xxx ***.c

读出文件内容:

```c
char buf[25]={0};
int fd;
fd= open("./test.txt",O_RDONLY);
if(-1==fd){
    printf("read error\n");
    return 1;
}
ret = read(fd, buf, 20);
if(-1==ret){
    printf("read error\n");
    close(fd);
    return 1;
}
printf("read %d bytes :%s\n",ret, buf);
close(fd);
return 0
```

或者:

```c
memset(buf,0x0,
       sizeof(buf));
char buff[20]="I am a ";
```

5、lseek()函数，文件读写位置二函数，由读写指针决定。

文件刚打开时，位于文件头部，当每读写若干字节后，向后偏移。

```
off_t lseek(int fd, off_t offset, int whence);
```

man 2 lseek 手册。

fd：打开文件 ｜ offset：偏移量 ｜ whence：参考值 { 文件头 SEEK_SET；一般：SEEK_CUR；当前指向位置 SEEK_END

返回值：成功时文件头部到偏移量！失败 -1。可计算文件大小，移到文件尾部。

示例①. 将读写位置移到文件头处。

```
off_t off = lseek(fd, 0, SEEK_SET);
if(off == -1) return -1;
```

② 移动文件尾。

```
off_t off = lseek(fd, 0, SEEK_END);
```

③ 移到开头100字节处。

```
off_t off = lseek(fd, 100, SEEK_SET);
if(off == -1) return -1;
```

④ 获取当前偏移量：

```
off_t off = lseek(fd, 0, SEEK_CUR);
if(off == -1) return -1;
```

上机练习.

1. 实现一个文件复制功能，给定一个源文件，读取源文件中之所有数据，将其写入到另一个目标文件中，如：

```
./app "src_file" "tar_file"
```

2. 给定一个文件，计算其大小并打印出来。

* 分析：1. 第一步：获取外部传参. int main(int argc, char **argv){ } 或 char *argv[].
2. char buf[30]={0};

第二步：
```
int size;
size=lseek(fd, 0, SEEK_END);
```

```
int calFilelen(const char *filename){
    int fd=-1;
    int ret=-1;
    fd=open(filename, O_RDONLY);
    if(-1==fd){ perror("open error\n");
                return -1;
    }
    ret=lseek(fd, 0, SEEK_END);
    return ret;
}
```

main中调用. calfilelen(argv[1]) 即可实现功能封装.