# Linux平台文件操作程序设计

实验名称：Linux平台文件操作程序设计

学时安排：2课时 指导老师：李赞

实验类别：验证型、设计型 实验要求：1人1组

学号： 姓名： 班级：

## 一、实验目的和任务

1. 本实验要求复习掌握ubuntu中C程序的编译方法。

2. Linux文件操作的C语言接口的使用。

## 二、实验设备介绍

1.软件需求： win10操作系统，VMware workstation，ubuntu18。

2.硬件需求: PC内存大于1G，硬盘空间大于20G。

## 三、注意事项和要求

1.理解linux平台文件操作相关函数。

2.引用正确的函数头文件。

## 四、实验内容和步骤

### 4.0Linux平台文件操作相关函数

文件是Linux平台重要的资源，Linux平台文件主要有四种：1.普通文件，2.字符设备文件，3.块设备文件，4.Socket文件。文件描述符可表示设备文件和普通文件。Linux平台<fcntl.h>头文件定义了下面的文件操作函数。Linux平台的socket文件与普通文件一样，支持read函数write函数。Led灯、蜂鸣器电机等设备在嵌入式Linux系统上被当成文件，对电机控制也是通过文件读写实现的。下表列出了Linux平台常用的文件操作函数与相应的头文件。

|  |  |  |
| --- | --- | --- |
| 函数名 |  |  |
| open | <sys/stat.h> | 打开一个已有文件或创建一个新文件。 |
| create | <sys/stat.h> | 创建一个新文件 |
| access | <unistd.h> | 检查文件或目录是否存在 |
| close | <unistd.h> | 关闭一个文件，包括设备文件，套接字和普通磁盘文件。 |
| read | <unistd.h> | 从打开的文件中读数据到缓冲区。 |
| write | <unistd.h> | 将缓冲区数据写入文件。 |
| lseek | <unistd.h> | 文件偏移函数，用于定位读写位置。 |
| fstat | <sys/stat.h> | 获得文件状态。 |
| fcntl | <fcntl.h> | 向已经打开的发送命令更改其属性。 |
| ioctl | <sys/ioctl.h> | 对文件描述符发送命令，用来控制设备。 |
| fseek | <stdio.h> | 控制文件指针读写位置。 |

### 4.1打开文件操作open和关闭文件close

Linux平台操作文件或设备时使用open方法，"test.txt"表示一个普通文件，"/dev/sda1"表示磁盘第一个分区，"/dev/dcm/0raw"表示直流电机设备文件，可以看出不同类型文件是由其文件命名方式来区分的。下面程序尝试打一个名为 text.txt 的文件，文件不存在则显示打开失败，打开成功后即关闭文件。

《Linux网络编程》page74

/\*013.c open-01.c打开文件的例子\*/

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

int main(void)

{

int fd = -1; /\*文件描述符声明\*/

char filename[] = "test.txt"; /\*打开的文件名\*/

fd = open(filename,O\_RDWR); /\*打开文件为可读写方式\*/

if(-1 == fd){ /\*打开失败\*/

printf("Open file %s failure!, fd:%d\n",filename,fd);

} else { /\*打开成功\*/

printf("Open file %s success,fd:%d\n",filename,fd);

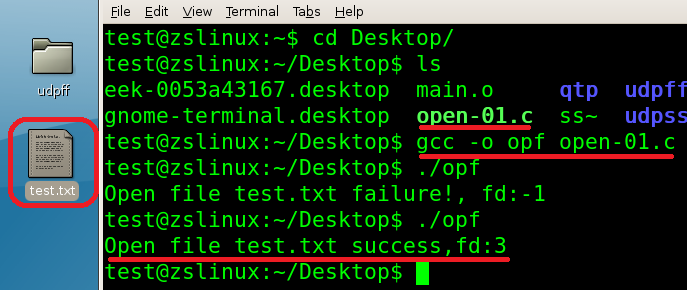
close(fd);

}

return 0;

}

下面是Debian上的运行效果图。



### 4.2读文件操作

《Linux网络编程》page78

Linux中使用read方法从文件中读出字节内容，并将字节内容放入缓冲区。

/\*014.c 读文件read-01.c,O\_CREAT和O\_EXCL的使用\*/

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <stdio.h>

int main(void)

{

int fd = -1,i;

ssize\_t size = -1;

char buf[10]; /\*存放数据的缓冲区\*/

char filename[] = "test.txt";

fd = open(filename,O\_RDONLY); /\*打开文件，如果文件不存在，则报错\*/

if(-1 == fd){ /\*文件已经存在\*/

printf("Open file %s failure,fd:%d\n",filename,fd);

}else { /\*文件不存在，创建并打开\*/

printf("Open file %s success,fd:%d\n",filename,fd);

}

/\*循环读取数据，直到文件末尾或者出错\*/

while(size){

size = read(fd, buf,10); /\*每次读取10个字节数据\*/

if( -1 == size) { /\*读取数据出错\*/

close(fd); /\*关闭文件\*/

printf("read file error occurs\n");

return -1; /\*返回\*/

}else{ /\*读取数据成功\*/

if(size >0 ){

printf("read %d bytes:",size); /\*获得size个字节数据\*/

printf("\""); /\*打印引号\*/

for(i = 0;i<size;i++){ /\*将读取的数据打印出来\*/

printf("%c",\*(buf+i));

}

printf("\"\n"); /\*打印引号并换行\*/

}else{//size = 0

printf("reach the end of file\n");

}

}

}

return 0;

}

在读文件时，文件读方法read方法返回字节个数是0时则到达文件结束。请不要根据标准C语言库定义的EOF符来判断文件结束，EOF符号根本不存在文件中，因为普通文件数据本质是字节序列，文件最后并没有代表EOF的这个字节。

//015.c 显示文件内容

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <stdio.h>

int main(void)

{

int fd = -1,i;

ssize\_t size = -1;

char buf[10]; /\*存放数据的缓冲区\*/

char filename[] = "test.txt";

fd = open(filename,O\_RDONLY); /\*打开文件，如果文件不存在，则报错\*/

if(-1 == fd){ /\*文件已经存在\*/

printf("Open file %s failure,fd:%d\n",filename,fd);

}else { /\*文件不存在，创建并打开\*/

printf("Open file %s success,fd:%d\n",filename,fd);

}

/\*循环读取数据，直到文件末尾或者出错\*/

/\*每次读取10个字节数据\*/

size = read(fd, buf,10);

while(size)

{

for(i = 0;i<size;i++)

{ /\*将读取的数据打印出来\*/

printf("%c",\*(buf+i));

}

size = read(fd, buf,10);

}

close(fd);

printf("file closed.\n");

return 0;

}

下面是Debian(UBUNTU)上的运行效果图:

### 4.3写文件操作

《Linux网络编程》page79

Linux中使用write方法将缓冲区中字节内容，从文件当前位置写入文件。每次成功的写入会返回写入的字节数，出错时返回-1。

/\*016.c 文件操作write-01.c,write()函数的使用\*/

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include <string.h>

#include <fcntl.h>

#include <stdio.h>

int main(void)

{

int fd = -1;

ssize\_t size = -1;

char buf[]="quick brown fox jumps over the lazy dog"; /\*存放数据的缓冲区\*/

char filename[] = "test.txt";

fd = open(filename,O\_RDWR); /\*打开文件，如果文件不存在，则报错\*/

if(-1 == fd){ /\*文件已经存在\*/

printf("Open file %s failure,fd:%d\n",filename,fd);

} else { /\*文件不存在，创建并打开\*/

printf("Open file %s success,fd:%d\n",filename,fd);

}

size = write(fd, buf,strlen(buf)); /\*将数据写入到文件test.txt中\*/

printf("write %d bytes to file %s\n",size,filename);

close(fd); /\*关闭文件\*/

return 0;

}

下面是Debian(UBUNTU)上的运行效果图:

### 4.4 向文件写入300个整数

请在下面程序基础上补充代码，实现向文件写入300个整数的功能。

//017.c 向文件写入字符串

#include <unistd.h>

#include <sys/file.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define BUF\_LEN 5

int main(void)

{

int fd,iCharLen;

char buf\_num[BUF\_LEN];

int i=0;

fd=open("300.txt",O\_RDWR | O\_CREAT, 0666);

if(fd < 0){

perror("open");

exit(1);

}

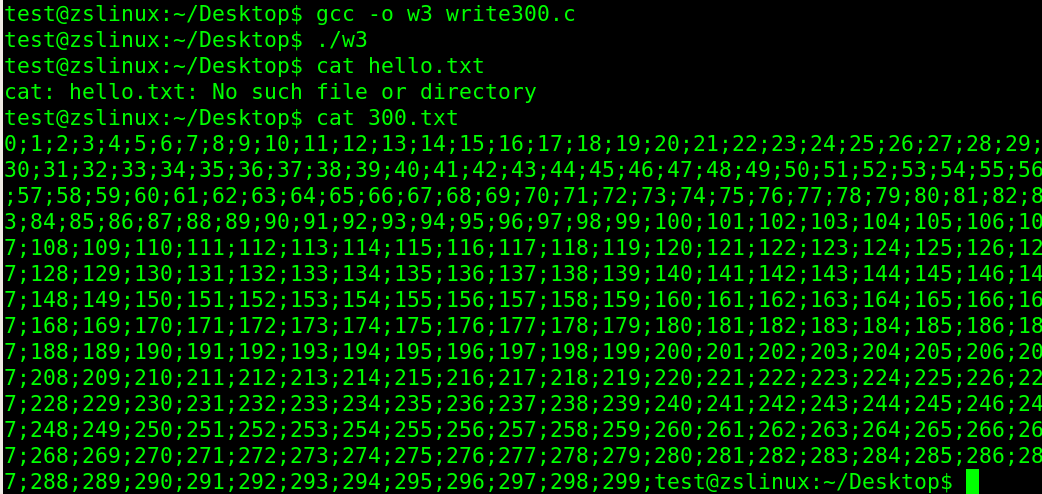
//请在此编写代码，实现向文件写入0-299字符格式的整数。

close(fd);

exit(0);

}

程序运行后的Debian结果如下图，该程序最终完成的源码和运行结果截屏后提交到学习通平台。



### 4.5 向文件最后追加内容

lseek方法可以对文件读写位进行定位，下面程序打开文件后将文件位置定位到文件末尾，并写入了给定字符串。也可以在打开文件的时候指定O\_APPEND参数，直接定位到文件最后。

//018.c 在文件最后追加内容

#include <unistd.h>

#include <sys/file.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define BUF\_LEN 5

int main(void)

{

int fd,iCharLen;

char \*buf="a quick brown fox jump over the lazy dog";

int i=0;

fd=open("300.txt",O\_WRONLY);

if(fd < 0){

perror("open");

exit(1);

}else{

lseek(fd,0,SEEK\_END);

write(fd,buf,strlen(buf));

close(fd);

}

exit(0);

}

## 五、程序调试中遇到的问题和解决过程及运行结果

1.将上述每个程序先使用gcc 编译并运行，分析过程中遇到问题，排查程序出错的原因

2.请将向文件写入300个整数的程序源码和运行结果截图后，提交到学习通平台。

3.编写一个程序实现文件拷贝的基本功能./mycopy 1.txt 2.txt 程序源码和运行结果截图后，提交到学习通平台。