嵌入式系统开发上机实验
代码手册

张笑、冯鸣夏、魏乐麒、李欢 编写

张亮 指导

西安电子科技大学嵌入式联合实验室

2014.12.20

Several experiments on BeagleBone Black

- IPC 实验 4

1.Unix Socket 4

- 示例代码： 4

* 程序名称：client.c */ 4

* 程序名称：server.c */ 6

- 实验过程： 8

2.FiFo Pipe 9

- 示例代码： 9

* 程序名称：fifo_read.c */ 9

* 程序名称：fifo_write.c */ 10

* 程序名称：pipetest_simple.c */ 11

* 程序名称：pipetest_second.c */ /与上面类似/ 12

- 实验过程： 13

3.SystemV 15

- 示例代码： 15

* 程序名称：testwrite.c */ 15

* 程序名称：testread.c */ 16

- 实验过程： 17

4.Mmap 18

- 示例代码： 18

* mmap_write.c */ 18

* mmap_read.c */ 19

* Makefile 文件 */ 20

- 实验过程： 20

- 驱动实验 20

1.字符驱动测试 20

- 示例代码 21

* 程序名称：globalvar.c 传入一个参数*/ 21

* 程序名称：globalvar_array.c 与前面类似，不同的是可传入一组参数*/ 23

* 程序名称：test.c */ 26

* Makefile 文件 */ 27

- 实验过程： 27

①module_param()测试 27

②字符驱动测试 28

2.POLL Device 29

- 示例代码： 29

* 程序名称：char_dev.c */ 29

* 程序名称：char_dev_new.c */ 33

* 程序名称：test.c */ 37

* Makefile */ 38

- 实验过程：.. 38

每个实验具体包括：源代码和实验过程。源代码为每个实验示例代码。实验过程主要是讲述如何在 PC 上或者板子上运行程序及其运行结果显示。

声明：我们默认您在阅读本代码文档前，已经充分理解说明文档中的内容。这样，下面的代码阅读起来非常容易，而且你可以轻松地解释运行结果。

不要试图去拷贝下面的源代码。最好亲手敲一遍，虽然会占用一定的时间，但是“扫帚不到，灰尘不会自己跑掉”。如果幸运的话，你可以发现一些 Bug，揪出它们将是一件有意思的工作。

Good luck!

- **IPC 实验**

1.Unix Socket

- 示例代码:

```
/* 程序名称: client.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <sys/types.h>
#include <errno.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/select.h>
#include <unistd.h>
#include <termios.h>
#include <sys/stat.h>
/**********定时器头文件***************/
#include <sys/time.h>
#include <signal.h>
/***********进程间 SOCKET 通信头文件**********/
#include <sys/socket.h>
#include <sys/un.h>

#include <sys/ioctl.h>
#pragma pack(1)          //设定为 1 字节对齐
#define UNIX_DOMAIN2 "/tmp/UNIX2.domain"
static char recv_php_buf[256]={0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07};
struct test
{
   char a;
   int b;
   int c;
}se;

int  main(int argc, char **argv)
{
   int connect_fd;
   int ret=0;
   int i;
   static struct sockaddr_un srv_addr;
   printf("IPC 通信线程\n");
   //while(1)
```

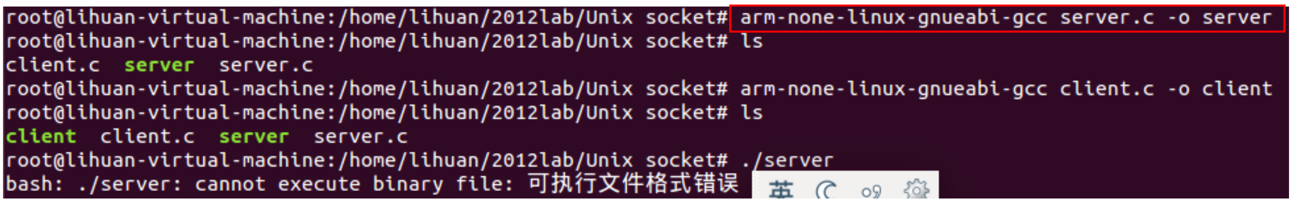
```
//{
//创建用于通信的套接字，通信域为UNIX通信域
connect_fd=socket(AF_UNIX,SOCK_STREAM,0);
printf("%d\n",connect_fd);
if(connect_fd<0)
{
    perror("cannot create communication socket");
    printf("%d\n",connect_fd);
    return -1;
}
else
{
    srv_addr.sun_family=AF_UNIX;
    strcpy(srv_addr.sun_path,UNIX_DOMAIN2);

    //连接服务器
    ret=connect(connect_fd,(struct sockaddr*)&srv_addr,sizeof(srv_addr));
    if(ret==-1)
    {
        close(connect_fd);
        printf("connect fail\n");
        //break;            //重新创建 socket
        close(connect_fd);
        return -1;
    }
    else
    {
        //否则，连接服务器成功
        se.a=0x01;
        se.b=0x01020304;
        se.c=0x05060708;
        write(connect_fd,recv_php_buf,20);//将数据传送到外部应用程序,发送实际长度
        //write(connect_fd,&se,sizeof(struct test));
        memset(recv_php_buf,0,sizeof(recv_php_buf));    //清空 socket_buf
        //sleep(1);
        //fcntl(connect_fd,F_SETEL,O_NONBLOCK);
        read(connect_fd,recv_php_buf,sizeof(recv_php_buf));
        printf("receive from server over\n");
        for(i=0;i<20;i++)
        {
            printf("%x ",recv_php_buf[i]);
        }
        //printf("%x ",se.c);
        printf("\n");
        close(connect_fd);
```

```
            //}
        }
    }
    return 0;
}
```

```
/* 程序名称: server.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <sys/types.h>
#include <errno.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/select.h>
#include <unistd.h>
#include <termios.h>
#include <sys/stat.h>
/**********定时器头文件***************/
#include <sys/time.h>
#include <signal.h>
/***********进程间 SOCKET 通信头文件**********/
#include <sys/socket.h>
#include <sys/un.h>

#define UNIX_DOMAIN "/tmp/UNIX2.domain"

static char recv_php_buf[256];  //接收 client 数据的缓冲
static int recv_php_num=0;      //接收 client 数据的总长度
const char recv_php_buf1[20]={0x00,0x01,0x02,0x03,0x04,0x05,0x06};
int main(int argc, char **argv)
{
    socklen_t clt_addr_len;
    int listen_fd;
    int com_fd;
    int ret=0;
    int i;

    int len;
    struct sockaddr_un clt_addr;
    struct sockaddr_un srv_addr;
    while(1)
    {
        //创建用于通信的套接字，通信域为 UNIX 通信域
        listen_fd=socket(AF_UNIX,SOCK_STREAM,0);
        if(listen_fd<0)
        {
            perror("cannot create listening socket");
            continue;
        }
        else
        {
            while(1)
            {
                //设置服务器地址参数
                srv_addr.sun_family=AF_UNIX;

    strncpy(srv_addr.sun_path,UNIX_DOMAIN,sizeof(srv_addr.sun_path)-1);
                unlink(UNIX_DOMAIN);
                //绑定套接字与服务器地址信息
                ret=bind(listen_fd,(struct
sockaddr*)&srv_addr,sizeof(srv_addr));
                if(ret==-1)
                {
                    perror("cannot bind server socket");
                    close(listen_fd);
                    unlink(UNIX_DOMAIN);
                    break;
                }
                //对套接字进行监听，判断是否有连接请求
                ret=listen(listen_fd,1);
                if(ret==-1)
                {
                    perror("cannot listen the client connect request");
                    close(listen_fd);
                    unlink(UNIX_DOMAIN);
                    break;
                }
                chmod(UNIX_DOMAIN,00777);//设置通信文件权限
                while(1)
                {
                    //当有连接请求时，调用 accept 函数建立服务器与客户机之间的连接
                    len=sizeof(clt_addr);
                    com_fd=accept(listen_fd,(struct sockaddr*)&clt_addr,&len);
                    if(com_fd<0)
                    {
                        perror("cannot accept client connect request");
                        close(listen_fd);
                        unlink(UNIX_DOMAIN);
                        break;
                    }
                    //读取并输出客户端发送过来的连接信息
                    memset(recv_php_buf,0,256);

    recv_php_num=read(com_fd,recv_php_buf,sizeof(recv_php_buf));
                    printf("\n=====recv=====\n");
                    for(i=0;i<recv_php_num;i++)
                    {
                        printf("%d ",recv_php_buf[i]);
                    }
                    printf("\n");
                    /*if(recv_php_buf[0]==0x02)
                      {
                      if(recv_php_buf[recv_php_num-1]==0x00)
                      {
                      recv_php_buf[recv_php_num-1]=0x01;
                      }
                      else
                      {
                      recv_php_buf[recv_php_num-1]=0x00;
                      }
                      }
                      */
                    //recv_php_buf[20]+=1;
                    write(com_fd,recv_php_buf,recv_php_num);
                    printf("\n=====send=====\n");
                    for(i=0;i<recv_php_num;i++)
                    {
                        printf("%d ",recv_php_buf[i]);
                    }
                    printf("\n");
                    //write(com_fd,recv_php_buf,20);
                    close(com_fd);//注意要关闭连接符号，不然会超过连接数而报错
                }

            }

        }
    }
    return 0;
}
```

- 实验过程：

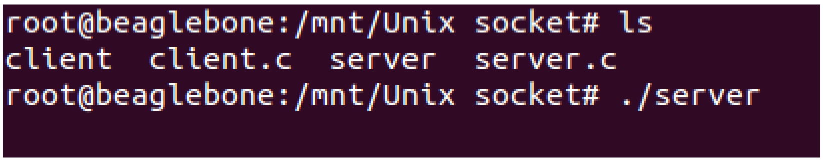
我们使用交叉编译链编译得到两个可执行文件。



```
root@lihuan-virtual-machine:/home/lihuan/2012lab/Unix socket# arm-none-linux-gnueabi-gcc server.c -o server
root@lihuan-virtual-machine:/home/lihuan/2012lab/Unix socket# ls
client.c  server  server.c
root@lihuan-virtual-machine:/home/lihuan/2012lab/Unix socket# arm-none-linux-gnueabi-gcc client.c -o client
root@lihuan-virtual-machine:/home/lihuan/2012lab/Unix socket# ls
client  client.c  server  server.c
root@lihuan-virtual-machine:/home/lihuan/2012lab/Unix socket# ./server
bash: ./server: cannot execute binary file: 可执行文件格式错误
```

试着运行，会发现提示格式错误，而它们可以在板子上运行。

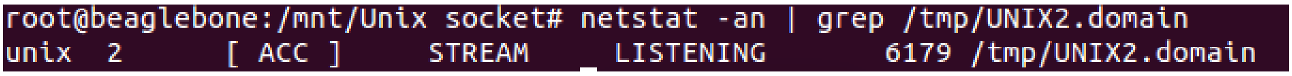
先运行./server,



```
root@beaglebone:/mnt/Unix socket# ls
client  client.c  server  server.c
root@beaglebone:/mnt/Unix socket# ./server
```

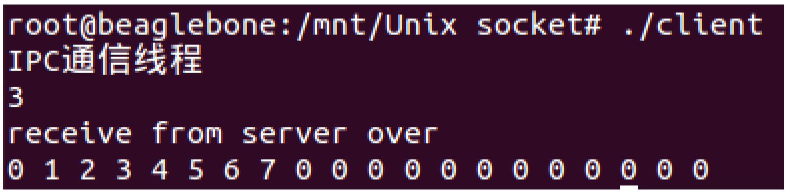
当运行 srv 程序后，该程序将处于监听状态。这时，可以通过 netstat 命令查看 LISTENING。

#netstat -an | grep /tmp/UNIX2.domain



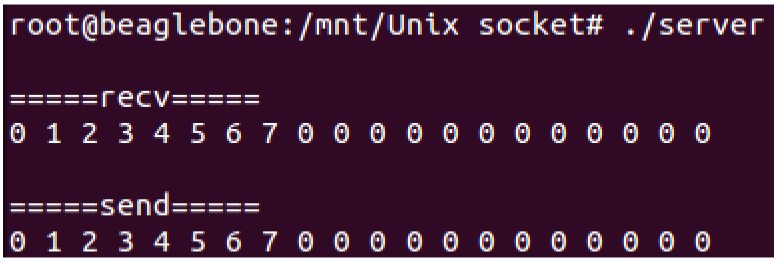
```
root@beaglebone:/mnt/Unix socket# netstat -an | grep /tmp/UNIX2.domain
unix  2      [ ACC ]     STREAM     LISTENING       6179 /tmp/UNIX2.domain
```

再运行./client



```
root@beaglebone:/mnt/Unix socket# ./client
IPC通信线程
3
receive from server over
0 1 2 3 4 5 6 7 0 0 0 0 0 0 0 0 0 0 0 0
```

server 端收到数据并发给 client



```
root@beaglebone:/mnt/Unix socket# ./server

=====recv=====
0 1 2 3 4 5 6 7 0 0 0 0 0 0 0 0 0 0 0 0

=====send=====
0 1 2 3 4 5 6 7 0 0 0 0 0 0 0 0 0 0 0 0
```

2.FiFo Pipe

- 示例代码:

```
/* 程序名称: fifo_read.c */
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
```

```
#include<sys/stat.h>
#include<fcntl.h>
#include<errno.h>

#define FIFO_FILE "/tmp/myfifo"

int main()
{
    char buf[100];
    int n = 0;
    int fd;

    if ((mkfifo(FIFO_FILE,S_IRWXU) < 0) && (errno != EEXIST)) //如果该fifo文件不存在，创建之
    {
        perror("mkfifo error");
        exit(-1);
    }

    if ((fd = open(FIFO_FILE,O_RDONLY | O_NONBLOCK)) < 0) //非阻塞方式打开
    {
        perror("open error");
        exit(-1);
    }

    while (1)
    {
        if ((n = read(fd,buf,100)) < 0)
        {
            if (errno == EAGAIN)
            {
                printf("No data yet\n");
            }

        }
    else if(n == 0)
        printf("No opened by write_only\n");
    else
        write(STDOUT_FILENO,buf,n);
        sleep(1); //sleep
    }
    unlink(FIFO_FILE);
    return 0;
}
```

```
/* 程序名称：fifo_write.c */
```

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<errno.h>
#include<fcntl.h>
#define FIFO_FILE "/tmp/myfifo"

int main()
{
    int fd = 0;
    int n;
    char buf[100];

    if ((fd = open(FIFO_FILE,O_WRONLY | O_NONBLOCK)) < 0) //非阻塞方式打开
    {
        perror("open error");
        exit(-1);
    }
    while (1)
    {
        fgets(buf,100,stdin);
        n = strlen(buf);
        if ((n = write(fd,buf,n)) < 0)
        {
            if (errno == EAGAIN)
                printf("The FIFO has not been read yet.Please try later\n");
        }
    }
    return 0;
}
```

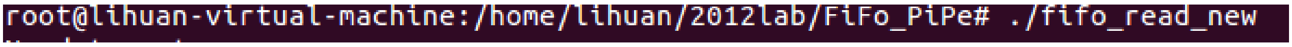
```
/* 程序名称：pipetest_simple.c */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <errno.h>

int main(unsigned int argc, unsigned char **argv)
{
    int pipe_fd[2];
    pid_t pid;
    char r_buf[100];
    char w_buf[4];
    int r_num;
    int cmd;

    memset(r_buf,0,sizeof(r_buf));
    memset(w_buf,0,sizeof(r_buf));

    if(pipe(pipe_fd)<0)
    {
        printf("FILE: %s, LINE: %d.pipe create error\n",__FILE__, __LINE__);
        return -1;
    }

    if((pid=fork())==0) //子进程中
    {
        printf("##########################\n");
        close(pipe_fd[1]);   //关闭写端
        sleep(3); //确保进程关闭写端
        r_num=read(pipe_fd[0],r_buf,100);
        printf("child read num is %d , the data read from the pipe 
is %d\n",r_num,atoi(r_buf));
        close(pipe_fd[0]);   //关闭读端
        //exit(0);
    }
    else if(pid>0)   //父进程中
    {
        close(pipe_fd[0]);   //关闭读端
        strcpy(w_buf,"111");
        if(write(pipe_fd[1],w_buf,4)!=-1)
        {
            printf("parent write over\n");
        }
        close(pipe_fd[1]);   //关闭写端
        printf("parent close fd[1]-write over\n");
        sleep(3);
    }
    return 0;
}

/* 程序名称: pipetest_second.c */ /与上面类似/

#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
```

```
{
    int pipe_fd[2];
    pid_t pid;
    char r_buf[4];
    char* w_buf;
    int writenum;
    int cmd;

    memset(r_buf,0,sizeof(r_buf));
    if(pipe(pipe_fd)<0)
    {
        printf("pipe create error\n");
        return -1;
    }

    if((pid=fork())==0)
    {
        close(pipe_fd[0]);
        close(pipe_fd[1]);
        sleep(10);
        exit(0);
    }
    else if(pid>0)
    {
        sleep(1);  //等待子进程完成关闭读端的操作
        close(pipe_fd[0]);//write
        w_buf="111";
        printf("FILE: %s, LINE: %d\r\n", __FILE__, __LINE__);
        if((writenum=write(pipe_fd[1],w_buf,4))==-1)
            printf("write to pipe error\n");
        else
            printf("the bytes write to pipe is %d \n", writenum);

        printf("FILE: %s, LINE: %d\r\n", __FILE__, __LINE__);
        close(pipe_fd[1]);
    }
}
```

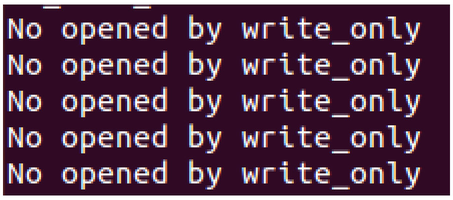
- 实验过程：

对于 Fifo，交叉编译两个文件：fifo_read_new.和 fifo_write_new.c。

一开始运行./fifo_read_new，



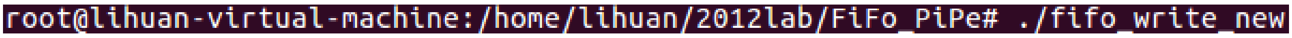
```
root@lihuan-virtual-machine:/home/lihuan/2012lab/FiFo_PiPe# ./fifo_read_new
```



```
No opened by write_only
No opened by write_only
No opened by write_only
No opened by write_only
No opened by write_only
```

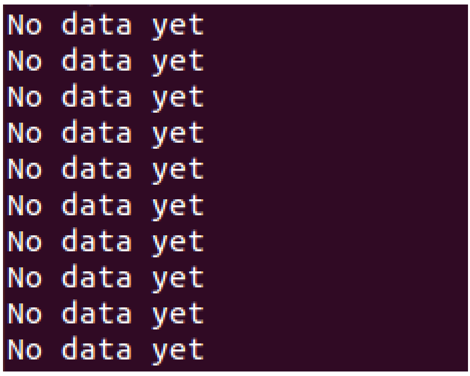
思考为何打印这句？

再运行./fifo_write_new，



```
root@lihuan-virtual-machine:/home/lihuan/2012lab/FiFo_PiPe# ./fifo_write_new
```

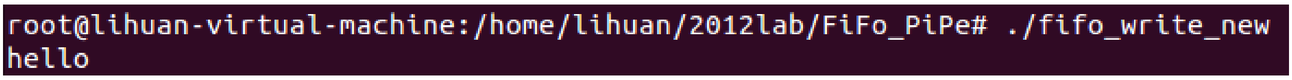
可看到 read 终端显示，



```
No data yet
No data yet
No data yet
No data yet
No data yet
No data yet
No data yet
No data yet
No data yet
No data yet
```

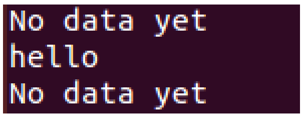
思考此时为何会打印这句？

在 write 终端输入 hello，



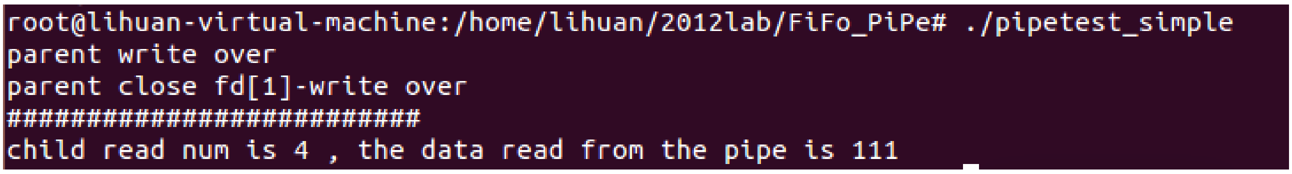
```
root@lihuan-virtual-machine:/home/lihuan/2012lab/FiFo_PiPe# ./fifo_write_new
hello
```

在 read 端显示，



```
No data yet
hello
No data yet
```

对于 Pipe，交叉编译 pipetest_simple.c。



```
root@lihuan-virtual-machine:/home/lihuan/2012lab/FiFo_PiPe# ./pipetest_simple
parent write over
parent close fd[1]-write over
############################
child read num is 4 , the data read from the pipe is 111
```

尝试多次运行./pipetest_simlpe，观察运行结果。思考为何会改变，并尝试解释。

pipetest_second.c 与 pipetest_simple.c 同理，不再赘述。

扩展问题：

Q:

PIPE 普遍用于 SHELL 中。根据上面的讲解，思考 ls | more 背后发生了什么？结合管道工作流程描述这个过程。

A:

step1：shell 创建一个 pipe（假设其 file descriptor 是 3 和 4，这样方便讨论）。

step2：shell fork 出两个子进程。此时，这两个子进程同继承了父进程的文件描述符，也就是说，他们同样有 3 和 4.

step3：第一个子进程调用 dup2(4, 1)。这就将 1（标准输出）的 file object 关闭了，并且，1 这个文件描述符，此时指向了 4 文件描述符指向的 file object，即 pipe 的写端。子进程关闭 3 和 4。子进程 execve()，执行 ls 命令。由于 execve 也是用的同一个文件描述符表，所以此时 ls 的输出实际上是 pipe 的写端。

step4：第二个子进程调用 dup2(3, 0)。如上，0（标准输入）的 file object 关闭，并且，0 这个文件描述符，指向了 3 这个文件描述符指向的 file object，即 pipe 的读端。子进程关闭 3 和 4. 子进程 execve()，执行 more 命令。此时，more 的输入实际上是 pipe 的读端。

于是，ls 的输出就顺利的重定向到 more 的输入了。

Are you understand?

3.SystemV

- 示例代码：

```
/* 程序名称：testwrite.c */
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>
typedef struct
{
    char name[4];
    int age;
} people;
int main(int argc, char** argv)
{
    int shm_id,i;
    key_t key;
    char temp;
    people *p_map;
    char* name = "/dev/shm/myshm2";
    key = ftok(name,0);
    if(key==-1)
    {
        perror("ftok error");
        return -1;
    }
    shm_id=shmget(key,4096,IPC_CREAT);
    if(shm_id==-1)
    {
        perror("shmget error");
        return -1;
    }
    p_map=(people*)shmat(shm_id,NULL,0);
    temp='a';
    for(i = 0;i<10;i++)
    {
        temp+=1;
        memcpy((*(p_map+i)).name,&temp,1);
        (*(p_map+i)).age=20+i;
    }
    if(shmdt(p_map)==-1)
    {
        perror(" detach error ");
        return -1;
    }
    return 0;
}

/* 程序名称：testread.c */

#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>
typedef struct
{
    char name[4];
    int age;
} people;
int main(int argc, char** argv)
{
    int shm_id,i;
    key_t key;
    people *p_map = NULL;
    char* name = "/dev/shm/myshm2";
    key = ftok(name,0);
    if(key == -1)
    {
        perror("ftok error");
        return -1;
    }
    shm_id = shmget(key,4096,IPC_CREAT);
    if(shm_id == -1)
    {
        perror("shmget error");
        return -1;
    }
    p_map = (people*)shmat(shm_id,NULL,0);
    if (p_map == NULL)
    {
        printf("shmat failed\r\n");
        return -1;
    }
    for(i = 0;i<10;i++)
    {
        printf( "name:%s\n",(*(p_map+i)).name );
        printf( "age %d\n",(*(p_map+i)).age );
    }
    if(shmdt(p_map) == -1)
    {
        perror(" detach error ");
    }
    return 0;
}
```

- 实验过程：

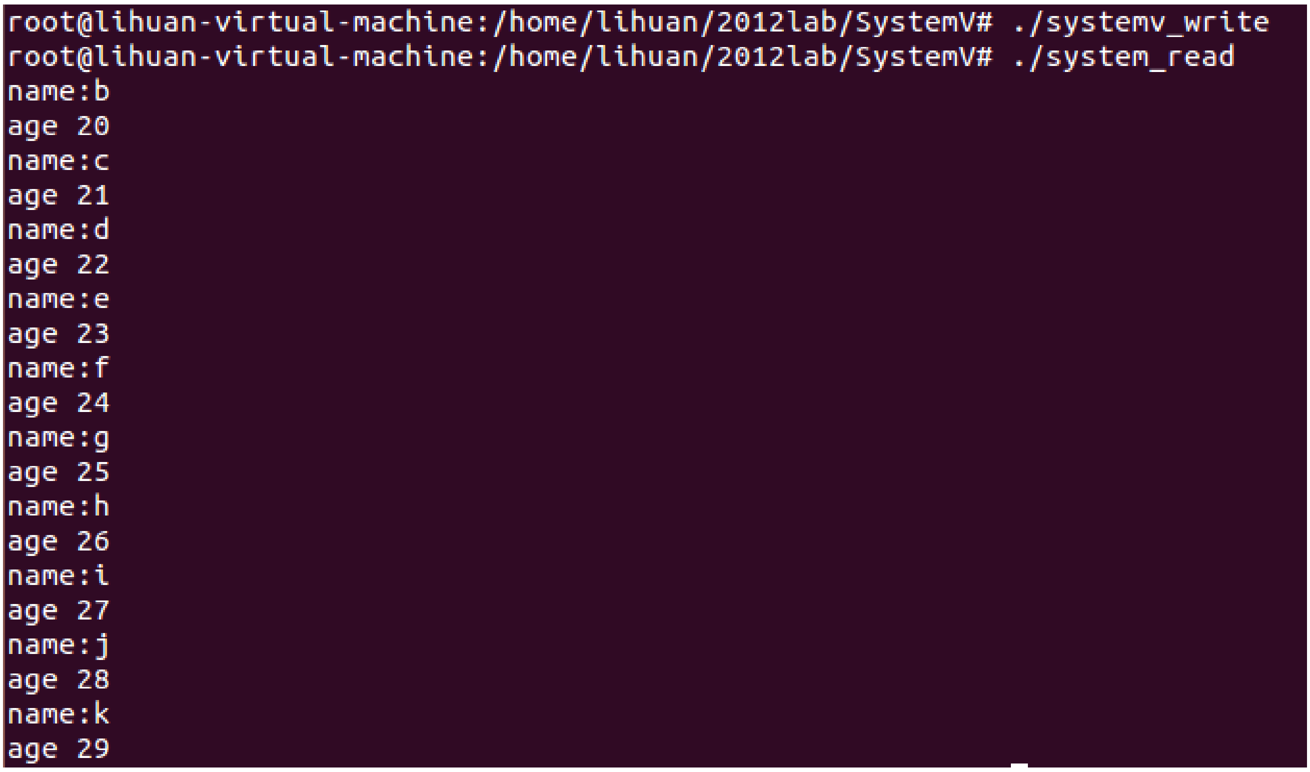
注意在运行./systemv_write 前，先创建文件/dev/shm/myshm2。（为什么？原因在于函数 ftok 是把一个已存的路径和一个整数标识符转换成一个 key_t 值，所以需要提前创建）

首先你需要交叉编译两个文件。

第一步运行./systemv_write

第二步运行./system_read

当然，你也可以颠倒运行顺序，看看会发生些什么，尝试解释。



```
root@lihuan-virtual-machine:/home/lihuan/2012lab/SystemV# ./systemv_write
root@lihuan-virtual-machine:/home/lihuan/2012lab/SystemV# ./system_read
name:b
age 20
name:c
age 21
name:d
age 22
name:e
age 23
name:f
age 24
name:g
age 25
name:h
age 26
name:i
age 27
name:j
age 28
name:k
age 29
```

4.Mmap

- 示例代码:

```
/* mmap_write.c */

#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <stdio.h>
//#define FILENAME "/home/zhangxiao/embededSystem/example/Mmap/test"

#define FILENAME "/tmp/test"
#define BUFLEN   256
typedef struct
{
    char name[BUFLEN];
    int  id;
}people;

int main(int argc, char** argv) // map a normal file as shared mem:
{
    int i;
    unsigned int pmap=0;
    int fd;
    fd=open(FILENAME ,O_CREAT|O_RDWR|O_TRUNC,00777 );
    assert(fd !=-1);
    pmap = (unsigned
int)mmap(0,sizeof(people),PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
    write(fd,"",sizeof(people));
    unsigned int addr;
    addr=pmap;
//  assert(pmap != 0);
    char tempname[30]="zhangxiao";
    int tempid=253;
//  printf("Input your name & stuID:\r\n");
//  scanf("%s %d",tempname,&tempid);
//  ((people*)pmap)->id=tempid;
    addr = pmap + sizeof(char)*BUFLEN;
    //memcpy(((people*)pmap)->name,&tempname,strlen(tempname));
    //addr=(unsigned int)&tempid;
    memcpy((void *)pmap,tempname,strlen(tempname));
    memcpy((void *)addr, &tempid,sizeof(int));
//  memcpy((void *)addr,&tempid,sizeof(int));
//  pmap = pmap + sizeof(char)*BUFLEN;
//  memcpy((char *)pmap,&tempid,sizeof(int));
    //memcpy((int)((people*)pmap)->id,&tempid,sizeof(int));

    munmap((void *) pmap,sizeof(char)*BUFLEN);
    close(fd);
    printf("umap ok\r\n");
    return 0;
}
```

```
/* mmap_read.c */

#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <stdio.h>
//#define FILENAME "/home/zhangxiao/embededSystem/example/Mmap/test"
#define FILENAME "/tmp/test"
#define BUFLEN   256
typedef struct
{
    char name[BUFLEN];
    int  id;
}people;

int main(int argc, char** argv) // map a normal file as shared mem:
{
    int i;
    unsigned int pmap=0;
    int fd;
    fd=open(FILENAME ,O_CREAT|O_RDWR,00777 );
    assert(fd !=-1);
    pmap = (unsigned
int)mmap(0,sizeof(people),PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
    unsigned int addr;
    addr=pmap;
    addr = pmap + sizeof(char)*BUFLEN;
    printf("id=%d   name=%s\n\r",*((int *)addr),(char *)pmap);
    munmap((void *) pmap,sizeof(char)*BUFLEN);
    close(fd);
    printf("umap ok\r\n");
    return 0;
}
```

```
/* Makefile 文件 */
CROSS = arm-none-linux-gnueabi-gcc
#CROSS = gcc
flags=-o
all:mmap_read mmap_write

mmap_read:mmap_read.c
    $(CROSS) $(flags) mmap_read mmap_read.c
mmap_write:mmap_write.c
    $(CROSS) $(flags) mmap_write mmap_write.c

clean:
    rm -rf mmap_read mmap_write test
```

- 实验过程：

需要在板子测试，根据需要修改 Makefile 文件中的编译器。

在 Mmap 目录下 make，两个二进制文件分别执行./mmap_write 和./mmap_read。

● 驱动实验

1.字符驱动测试

- 示例代码

```
/* 程序名称：globalvar.c 传入一个参数*/
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/slab.h>
#include <asm/uaccess.h>
#include <linux/moduleparam.h>

MODULE_LICENSE("GPL");

int globalvar_open(struct inode * inode, struct file * filp);
int globalvar_release(struct inode *, struct file *);
int globalvar_read(struct file *, char *, size_t, loff_t *) ;
int globalvar_write(struct file *, char *, size_t, loff_t *) ;
int globalvar_ioctl( struct file * filp, unsigned int cmd, unsigned long args);


int dev_major = 50;
int dev_minor = 0;

struct file_operations globalvar_fops =
{
    owner:THIS_MODULE,
     open:globalvar_open,
     release:globalvar_release,
     read:globalvar_read,
     write:globalvar_write,
     unlocked_ioctl:globalvar_ioctl,
};

struct globalvar_dev
{
    int global_var;
    struct cdev cdev;
};

struct globalvar_dev *my_dev;

static void __exit globalvar_exit(void)
{
    dev_t devno= MKDEV(dev_major, dev_minor);
    cdev_del(&my_dev->cdev);
    kfree(my_dev);
    unregister_chrdev_region(devno, 1);
    printk("globalvar_exit called.\r\n");
    return;
}
static int test_var = 0xFF;
module_param(test_var, int, 0644);
static int __init globalvar_init(void )
{
    int ret, err;
    dev_t devno = MKDEV(dev_major, dev_minor);
    ret = alloc_chrdev_region(&devno, dev_minor, 1, "globalvar");
    dev_major = MAJOR(devno);
    if (ret < 0)
    {
        printk("register failed.\r\n");
        globalvar_exit();
        return ret;
    }
    else
    {
        printk("globalvar init succeed\r\n");
    }
    my_dev = kmalloc(sizeof(struct globalvar_dev), GFP_KERNEL);
    if (my_dev == NULL)
    {
        printk("kmalloc failed.\r\n");
    }
    else
    {
        printk("globalvar kmalloc succeed.\r\n");
        my_dev->global_var = 0;
        cdev_init(&my_dev->cdev, &globalvar_fops);
        my_dev->cdev.owner = THIS_MODULE;
        err = cdev_add(&my_dev->cdev, devno, 1);
        if (err < 0)
        {
            printk("add dev failed.\r\n");
        }
        printk("globalvar cdev_add succeed.\r\n");
    }
    printk("test_var = 0x%x.\r\n", test_var);
    return ret;
}

int globalvar_open(struct inode * inode, struct file * filp)
```

```
{
    struct globalvar_dev *dev;
    dev = container_of(inode->i_cdev, struct globalvar_dev, cdev);
    filp->private_data = dev;
    printk("globalvar open called\r\n");
    return 0;
}
int globalvar_release(struct inode * inode, struct file * filp)
{
    printk("globalvar release called\r\n");
    return 0;
}
int globalvar_read(struct file * filp, char * buf, size_t len, loff_t * off)
{
    struct globalvar_dev *dev = filp->private_data;
    if (copy_to_user((void *)buf, (const void *)&dev->global_var, sizeof(int))
< 0 )
    {
        return -EFAULT;
    }
    printk("globalvar read called, global_var = 0x%x.\r\n", dev->global_var);
    return sizeof(int);
}
int globalvar_write(struct file * filp, char * buf, size_t len , loff_t * off)
{
    struct globalvar_dev *dev = filp->private_data;
    if (copy_from_user((void *)&dev->global_var,  (const void *)(buf),
sizeof(int)) < 0 )
    {
        return -EFAULT;
    }
    printk("globalvar write called\r\n");
    return sizeof(int);
}
int globalvar_ioctl( struct file * filp, unsigned int cmd, unsigned long args)
{
    printk("cmd = 0x%x\r\n", cmd);
    return 0;
}
module_init(globalvar_init);
module_exit(globalvar_exit);
```

```
/* 程序名称：globalvar_array.c 与前面类似，不同的是可传入一组参数*/
```

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
```

```
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/slab.h>
#include <asm/uaccess.h>
#include <linux/moduleparam.h>

MODULE_LICENSE("GPL");

int globalvar_open(struct inode * inode, struct file * filp);
int globalvar_release(struct inode *, struct file *);
int globalvar_read(struct file *, char *, size_t, loff_t *) ;
int globalvar_write(struct file *, char *, size_t, loff_t *) ;


int dev_major = 50;
int dev_minor = 0;

struct file_operations globalvar_fops =
{
   owner:THIS_MODULE,
   open:globalvar_open,
   release:globalvar_release,
   read:globalvar_read,
   write:globalvar_write,
};

struct globalvar_dev
{
   int global_var;
   struct cdev cdev;
};

struct globalvar_dev *my_dev;

static void __exit globalvar_exit(void)
{
   dev_t devno= MKDEV(dev_major, dev_minor);
   cdev_del(&my_dev->cdev);
   kfree(my_dev);
   unregister_chrdev_region(devno, 1);
   printk("globalvar_exit called.\r\n");
   return;
}
static int test_var = 0xFF;
module_param(test_var, int, 0644);
static int test_array[16];
static int test_num= 0;
module_param_array(test_array,int, &test_num, 0644);
static int __init globalvar_init(void )
{
   int ret, err;
   dev_t devno = MKDEV(dev_major, dev_minor);
   ret = alloc_chrdev_region(&devno, dev_minor, 1, "globalvar");
   dev_major = MAJOR(devno);
   if (ret < 0)
   {
      printk("register failed.\r\n");
      globalvar_exit();
      return ret;
   }
   else
   {
      printk("globalvar init succeed\r\n");
   }
   my_dev = kmalloc(sizeof(struct globalvar_dev), GFP_KERNEL);
   if (my_dev == NULL)
   {
      printk("kmalloc failed.\r\n");
   }
   else
   {
      printk("globalvar kmalloc succeed.\r\n");
      my_dev->global_var = 0;
      cdev_init(&my_dev->cdev, &globalvar_fops);
      my_dev->cdev.owner = THIS_MODULE;
      err = cdev_add(&my_dev->cdev, devno, 1);
      if (err < 0)
      {
         printk("add dev failed.\r\n");
      }
      printk("globalvar cdev_add succeed.\r\n");
   }
   printk("test_var = 0x%x.\r\n", test_var);
   printk("test_num = %d\r\n", test_num);
   {
      unsigned int i = 0;
      for (i = 0; i < test_num; i++)
      {
         printk("test_array index %d = 0x%x\r\n", i, test_array[i]);
      }
   }
   return ret;
}

int globalvar_open(struct inode * inode, struct file * filp)
{
    struct globalvar_dev *dev;
    dev = container_of(inode->i_cdev, struct globalvar_dev, cdev);
    filp->private_data = dev;
    printk("globalvar open called\r\n");
    return 0;
}
int globalvar_release(struct inode * inode, struct file * filp)
{
    printk("globalvar release called\r\n");
    return 0;
}
int globalvar_read(struct file * filp, char * buf, size_t len, loff_t * off)
{
    struct globalvar_dev *dev = filp->private_data;
    if (copy_to_user((void *)buf, (const void *)&dev->global_var, sizeof(int))
< 0 )
    {
        return -EFAULT;
    }
    printk("globalvar read called, global_var = 0x%x.\r\n", dev->global_var);
    return sizeof(int);
}
int globalvar_write(struct file * filp, char * buf, size_t len , loff_t * off)
{
    struct globalvar_dev *dev = filp->private_data;
    if (copy_from_user((void *)&dev->global_var,  (const void *)(buf),
sizeof(int)) < 0 )
    {
        return -EFAULT;
    }
    printk("globalvar write called\r\n");
    return sizeof(int);
}
module_init(globalvar_init);
module_exit(globalvar_exit);
```

```
/* 程序名称：test.c  */

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

```
#include <assert.h>
int main(int argc, char **argv)
{
    unsigned int writenum =0x253 ;
    unsigned int readnum=0;
    int ret = 0;
    int fd = open("/dev/globalvar", O_RDWR, S_IRUSR|S_IWUSR);
    assert(fd != 0);
#if 0
    if (write(fd, &writenum, sizeof(writenum)) < 0 )
    {
        printf("write failed.\r\n");
        close(fd);
    }
#endif
    if ( (ret = read(fd, &readnum, sizeof(readnum))) < 0 )
    {
        printf("read failed.\r\n");
        close(fd);
    }
    printf("readnum var is 0x%x\r\n",readnum);
#if 0
    ioctl(fd, 0x11, NULL);
#endif
    close(fd);
    return 0;
}
```

```
/*  Makefile 文件 */

ifneq ($(KERNELRELEASE),)
#obj-m := globalvar_proc.o
    obj-m := globalvar.o
#   obj-m := globalvar_array.o
else
    KERNELDIR ?= /lib/modules/$(shell uname -r)/build
    PWD := $(shell pwd)
all:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
clean:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) clean
endif
```

- 实验过程：

①module_param()测试

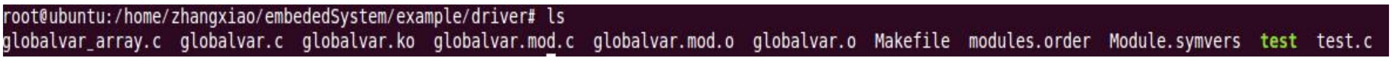
在用户态下编程可以通过 main()来传递命令行参数，而编写一个内核模块则可通过 module_param()来传递命令行参数。

这里只针对 globalvar.c 进行说明。

进入 driver/目录下

#make

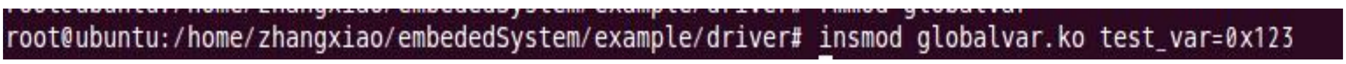
生成 globalvar.ko 文件



```
root@ubuntu:/home/zhangxiao/embededSystem/example/driver# ls
globalvar_array.c  globalvar.c  globalvar.ko  globalvar.mod.c  globalvar.mod.o  globalvar.o  Makefile  modules.order  Module.symvers  test  test.c
```

命令行输入

#insmod globalvar.ko test_var=0x123

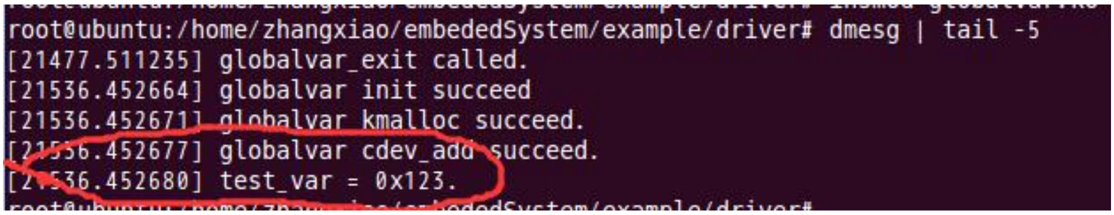


```
root@ubuntu:/home/zhangxiao/embededSystem/example/driver# insmod globalvar.ko test_var=0x123
```

命令行输入

#dmesg | tail -5

查看打印信息



```
root@ubuntu:/home/zhangxiao/embededSystem/example/driver# dmesg | tail -5
[21477.511235] globalvar_exit called.
[21536.452664] globalvar init succeed
[21536.452671] globalvar kmalloc succeed.
[21536.452677] globalvar cdev_add succeed.
[21536.452680] test_var = 0x123.
```

②字符驱动测试

主要测试对字符驱动文件的 open、release、read、write、ioctl 操作。

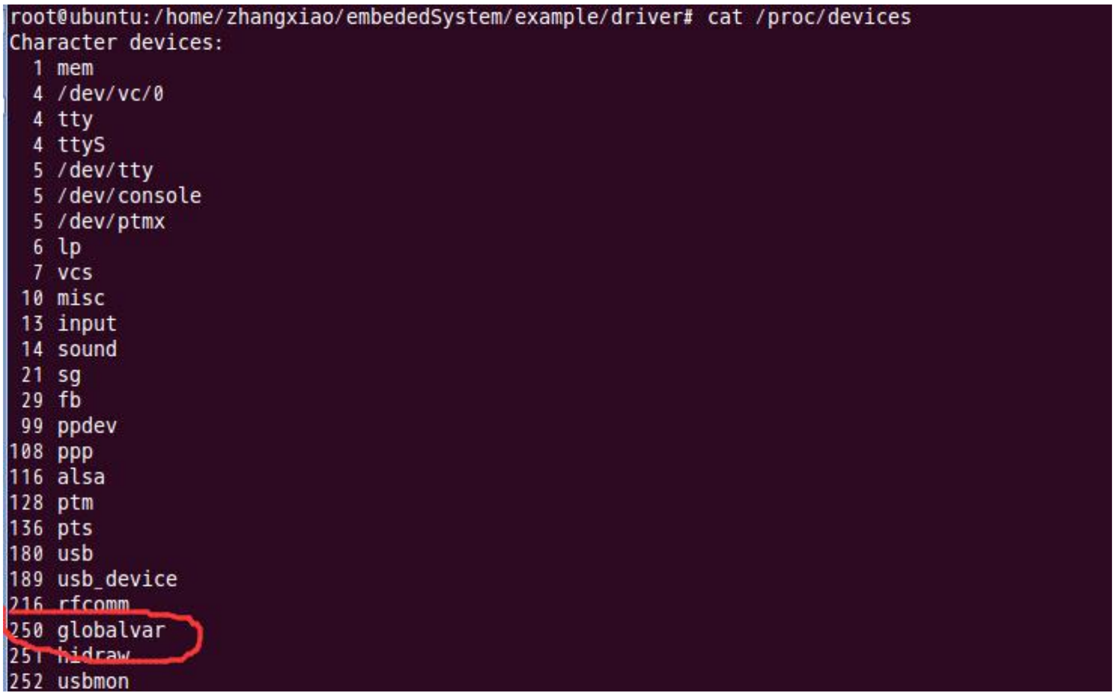
实验代码：

test.c

加载字符驱动后使用

#cat /proc/devices

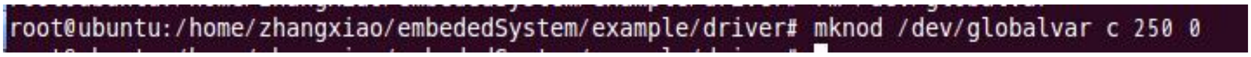
查看字符设备的主设备号。



```
root@ubuntu:/home/zhangxiao/embededSystem/example/driver# cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  6 lp
  7 vcs
 10 misc
 13 input
 14 sound
 21 sg
 29 fb
 99 ppdev
108 ppp
116 alsa
128 ptm
136 pts
180 usb
189 usb_device
216 rfcomm
250 globalvar
251 hidraw
252 usbmon
```

由上一步得出的打印信息得出字符设备的主设备号为 250，下面开始创建设备节点：

```
#mknod /dev/globalvar c 250 0
```

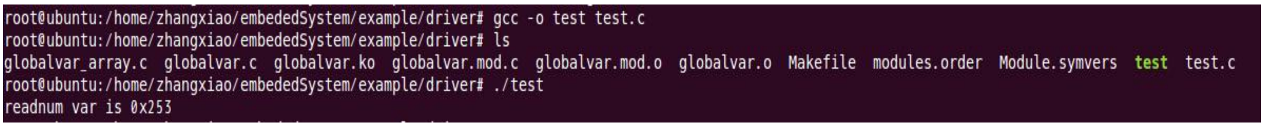


```
root@ubuntu:/home/zhangxiao/embededSystem/example/driver# mknod /dev/globalvar c 250 0
```

编译并运行测试文件

```
#gcc -o test test.c
```

```
#./test
```

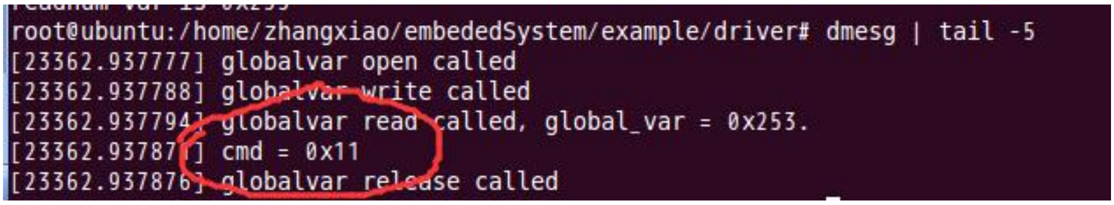


```
root@ubuntu:/home/zhangxiao/embededSystem/example/driver# gcc -o test test.c
root@ubuntu:/home/zhangxiao/embededSystem/example/driver# ls
globalvar_array.c  globalvar.c  globalvar.ko  globalvar.mod.c  globalvar.mod.o  globalvar.o  Makefile  modules.order  Module.symvers  test  test.c
root@ubuntu:/home/zhangxiao/embededSystem/example/driver# ./test
readnum var is 0x253
```

打开 test.c 源文件，反注释掉 ioctl 的部分，保存退出重复上述步骤后，在终端输入

```
#dmesg | tail -5
```

查看打印信息。



```
root@ubuntu:/home/zhangxiao/embededSystem/example/driver# dmesg | tail -5
[23362.937777] globalvar open called
[23362.937788] globalvar write called
[23362.937794] globalvar read called, global_var = 0x253.
[23362.937871] cmd = 0x11
[23362.937876] globalvar release called
```

2.POLL Device

- 示例代码：

```
/*  程序名称：char_dev.c */
```

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/mm.h>
#include <asm/uaccess.h>
#include <linux/init.h>
#include <linux/poll.h>
#define READ_BUF_SIZE  1024
#define WRITE_BUF_SIZE 1024
#define MAX_DATABUF_SIZE 1024

//为了便于参数传递宏的使用,将主设备号和设备名进行了相应的修改
static int DEV_MAJOR = 0;
static char *DEV_NAME = "char_dev";
static char* DataBuf = NULL;
int major;

//设定参数传递宏
module_param(DEV_MAJOR,int,0);
module_param(DEV_NAME,charp,0);

struct Mydevice
{
   const char *name;       /* DEV name */
   unsigned int major;     /* Major num */
   unsigned int minor;     /* Minor num */
   unsigned char *read_buffer;    /* Read Buffer area */
   unsigned char *write_buffer;   /* Write Buffer area */
   wait_queue_head_t read_queue;   /* Read Queue */
   wait_queue_head_t write_queue;  /* дWrite Queue */
   struct semaphore sem;    /* Semaphore for lock */
};

int my_open(struct inode *inode,struct file *filp)
{
   struct Mydevice  *dev = kmalloc(sizeof(struct Mydevice), GFP_KERNEL);
   if(dev == NULL) {
      printk(" KERN_ALERT allocate device memory failed.\n");
      return(-ENOMEM);
   }
   dev->name = DEV_NAME;
   dev->major = MAJOR(inode->i_rdev);
   dev->minor = MINOR(inode->i_rdev);
   dev->read_buffer = kmalloc(READ_BUF_SIZE,GFP_KERNEL);
   if(dev->read_buffer == NULL)
      printk(" KERN_ALERT allocate read buffer memory failed.\n");
   dev->write_buffer = kmalloc(WRITE_BUF_SIZE,GFP_KERNEL);
```

```
    if(dev->read_buffer == NULL)
        printk(" KERN_ALERT allocate write buffer memory failed.\n");
    init_waitqueue_head(&dev->read_queue);
    init_waitqueue_head(&dev->write_queue);
    if(filp->private_data == NULL)
        filp->private_data = dev;

    DataBuf = kmalloc(MAX_DATABUF_SIZE,GFP_KERNEL);
    if(DataBuf == NULL)
        printk(" KERN_ALERT allocate DataBuf memory failed.\n");
    printk("The function of my_open has been called!\n");

    return 0;
}

int my_release(struct inode *inode,struct file *filp)
{
    struct Mydevice *dev = filp->private_data;
    if(dev->read_buffer != NULL)
        kfree(dev->read_buffer);
    if(dev->write_buffer != NULL)
        kfree(dev->write_buffer);
    kfree(dev);
    printk("The function of my_release has been called £¡\n");
    return 0;
}

ssize_t my_read(struct file *filp,char *buf,size_t count,loff_t *offp)
{
    int accountread = 0;
    char *pdata = kmalloc(count,GFP_KERNEL);
    if(pdata == NULL)
        return (-ENOMEM);
    //防止copy_to_user的警告
    accountread = copy_to_user(buf,DataBuf,count);
    *offp += count;
    printk("The function of my_read has been called £¡\n");
    return count;
}

ssize_t my_write(struct file *filp,char *buf,size_t count,loff_t *offp)
{
    int accountwrite = 0;
    char *pdata = kmalloc(count,GFP_KERNEL);
    if(pdata == NULL)
        return (-ENOMEM);
    //防止copy_to_user的警告
    accountwrite = copy_from_user(DataBuf,buf,count);
    *offp += count;
    printk("The function of my_write has been called £¡\n");
    return count;
}
#define DRIVER_EVENT_BIT   13
int my_poll(struct file *filp, poll_table *wait)
{
    unsigned int mask = 0;

    mask |= 1<<DRIVER_EVENT_BIT;
    return mask;
}
int  my_ioctl (struct inode *inode,struct file *filp,unsigned int cmd,unsigned long arg)
{
    switch(cmd){
        case 1 :{
             printk("This is command 1 !\n");
             break;
          }
        case 2 :{
             printk("This is command 2 !\n");
             break;
          }
        case 3 :{
             printk("This is command 3 !\n");
             break;
          }
        default :{
              printk("There is no such command !\n");
              return -1;
           }
    }
    printk("The function of my_ioctl has been called %d",cmd);
    return 0;
}

struct file_operations fops = {
open  :    my_open,          /* open º¯Ê */
      release:   my_release,   /* write º¯Ê */
      read:   my_read,       /* read º¯Ê */
      write:  my_write,      /* write º¯Ê */
      ioctl:  my_ioctl,      /* ioctl º¯Ê */
      poll:   my_poll,       /* ioctl º¯Ê */
```

```
};

int my_init(void){
    int res = register_chrdev(DEV_MAJOR,DEV_NAME,&fops);
    if(res < 0){
        printk("My device register failed !\n");
        return res;
    }
    if(res > 0) major = res;
    printk("My device register success !, major = %d name = %s\n", major,DEV_NAME);

    return 0;
}

int my_cleanup(void){
    unregister_chrdev(major,DEV_NAME);
    printk("My device release success !\n");
    return 0;
}

module_init(my_init);
module_exit(my_cleanup);
MODULE_LICENSE("GPL");//为了避免”no license”警告
```

```
/*  程序名称: char_dev_new.c */
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/mm.h>
#include <asm/uaccess.h>
#include <linux/init.h>
#include <linux/poll.h>

#define READ_BUF_SIZE   1024
#define WRITE_BUF_SIZE  1024
#define MAX_DATABUF_SIZE 1024

//为了便于参数传递宏的使用,将主设备号和设备名进行了相应的修改
static int DEV_MAJOR = 0;
static char *DEV_NAME = "char_dev_new";
static char* DataBuf = NULL;
int major;

//设定参数传递宏
module_param(DEV_MAJOR,int,0);
module_param(DEV_NAME,charp,0);
```

```
struct Mydevice
{
    const char *name;      /* DEV name */
    unsigned int major;    /* Major num */
    unsigned int minor;    /* Minor num */
    unsigned char *read_buffer;   /* Read Buffer area */
    unsigned char *write_buffer;  /* Write Buffer area */
    wait_queue_head_t read_queue;   /* Read Queue */
    wait_queue_head_t write_queue;  /* дWrite Queue */
    struct semaphore sem;   /* Semaphore for lock */
};

int my_open(struct inode *inode,struct file *filp)
{
    struct Mydevice  *dev = kmalloc(sizeof(struct Mydevice), GFP_KERNEL);
    if(dev == NULL) {
        printk(" KERN_ALERT allocate device memory failed.\n");
        return(-ENOMEM);
    }
    dev->name = DEV_NAME;
    dev->major = MAJOR(inode->i_rdev);
    dev->minor = MINOR(inode->i_rdev);
    dev->read_buffer = kmalloc(READ_BUF_SIZE,GFP_KERNEL);
    if(dev->read_buffer == NULL)
        printk(" KERN_ALERT allocate read buffer memory failed.\n");
    dev->write_buffer = kmalloc(WRITE_BUF_SIZE,GFP_KERNEL);
    if(dev->read_buffer == NULL)
        printk(" KERN_ALERT allocate write buffer memory failed.\n");
    init_waitqueue_head(&dev->read_queue);
    init_waitqueue_head(&dev->write_queue);
    if(filp->private_data == NULL)
        filp->private_data = dev;

    DataBuf = kmalloc(MAX_DATABUF_SIZE,GFP_KERNEL);
    if(DataBuf == NULL)
        printk(" KERN_ALERT allocate DataBuf memory failed.\n");
    printk("The function of my_open has been called!\n");

    return 0;
}

int my_release(struct inode *inode,struct file *filp)
{
    struct Mydevice *dev = filp->private_data;
    if(dev->read_buffer != NULL)
```

```
        kfree(dev->read_buffer);
    if(dev->write_buffer != NULL)
        kfree(dev->write_buffer);
    kfree(dev);
    printk("The function of my_release has been called £¡\n");
    return 0;
}

ssize_t my_read(struct file *filp,char *buf,size_t count,loff_t *offp)
{
    int accountread = 0;
    char *pdata = kmalloc(count,GFP_KERNEL);
    if(pdata == NULL)
        return (-ENOMEM);
    //防止 copy_to_user 的警告
    accountread = copy_to_user(buf,DataBuf,count);
    *offp += count;
    printk("The function of my_read has been called £¡\n");
    return count;
}

ssize_t my_write(struct file *filp,char *buf,size_t count,loff_t *offp)
{
    int accountwrite = 0;
    char *pdata = kmalloc(count,GFP_KERNEL);
    if(pdata == NULL)
        return (-ENOMEM);
    //防止 copy_to_user 的警告
    accountwrite = copy_from_user(DataBuf,buf,count);
    *offp += count;
    printk("The function of my_write has been called £¡\n");
    return count;
}
int my_poll(struct file *filp, poll_table *wait)
{
    unsigned int mask = 0;

    mask |= POLLIN;
    return mask;
}
int  my_ioctl (struct inode *inode,struct file *filp,unsigned int cmd,unsigned
long arg)
{
    switch(cmd){
        case 1 :{
                printk("This is command 1 !\n");
```

```
             break;
          }
       case 2 :{
             printk("This is command 2 !\n");
             break;
          }
       case 3 :{
             printk("This is command 3 !\n");
             break;
          }
       default :{
              printk("There is no such command !\n");
              return -1;
           }
    }
    printk("The function of my_ioctl has been called %d",cmd);
    return 0;
}

struct file_operations fops = {
open  :   my_open,         /* openº¯Ê */
      release:   my_release,  /* writeº¯Ê */
      read:   my_read,      /* readº¯Ê */
      write:  my_write,     /* writeº¯Ê */
      ioctl:  my_ioctl,     /* ioctlº¯Ê */
      poll:   my_poll,      /* ioctlº¯Ê */

};

int my_init(void){
    int res = register_chrdev(DEV_MAJOR,DEV_NAME,&fops);
    if(res < 0){
       printk("My device register failed !\n");
       return res;
    }
    if(res > 0) major = res;
    printk("My device register success !, major = %d name = %s\n", major,DEV_NAME);

    return 0;
}

int my_cleanup(void){
    unregister_chrdev(major,DEV_NAME);
    printk("My device release success !\n");
    return 0;
}
```

```
module_init(my_init);
module_exit(my_cleanup);
MODULE_LICENSE("GPL");//为了避免”no license”警告
```

```
/*  程序名称: test.c */
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/fcntl.h>
#include <sys/poll.h>

int main(int argc, char **argv)
{
        int fd = 0;
        int fd_new = 0;
        char* writebuf = "I am xidian";
        char* readbuf = malloc(sizeof("I am xidian"));
        fd = open("/dev/char_dev",O_RDWR);
        fd_new = open("/dev/char_dev_new",O_RDWR);
        if (fd <= 0)
        {
                printf("open failed.\n");
        }
        else
        {
                int size=0;
                printf("fd = %d.\n", fd);
                printf("fd_new = %d.\n", fd_new);
                if((size=write(fd,writebuf,sizeof("I am KuangRen"))>0))
                {
                        printf("write success.\n%s\n",writebuf);
                        size=0;

                }
                else
                {
                        printf("write error.\n");
                }

                if((size=read(fd,readbuf,sizeof("I am KuangRen")))>0)
                {
                        printf("read success.\n%s\n",readbuf);
                }
                else
                {
                        printf("read error.\n");
                }
                ioctl(fd, 0x01, NULL);
                ioctl(fd, 0x02, NULL);
                ioctl(fd, 0x03, NULL);

                {
                        struct pollfd fds[2];
                        fds[0].fd = fd;
                        fds[0].events = POLLIN;
                        fds[1].fd = fd_new;
                        fds[1].events = POLLIN;

                        if (poll(fds, 1, 3000) > 0)
                        {
                                if (fds[0].revents  )
                                {
                                        printf("POLL0 event = 
0x%x\r\n",fds[0].revents);
                                }
                                if (fds[1].revents & POLLIN )
                                {
                                        printf("POLL1 in\r\n");
                                }

                        }
                        else
                        {
                                printf("timeout\r\n");
                        }
                }
                close(fd);
        }

        printf("The End\n");
        return 0;
}
```

```
/*  Makefile */

obj-m += char_dev.o
obj-m += char_dev_new.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

- 实验过程：

1、测试 char_dev.c 和 hello_param.c 的编译，应用程序调用。

2、测试 char_dev.c 和 char_dev_new.c 两个一起进行 poll 调用，按照 test.c 文件执行。

1、创建文件完成后，输入

#make

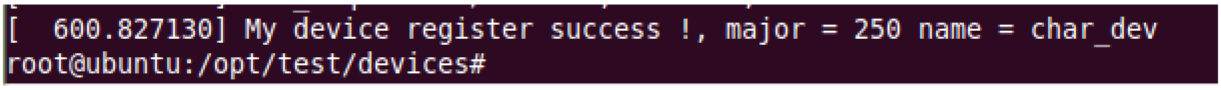
会在 devices 目录下生成.ko 文件（模块文件）

加载 char_dev 模块键入

#insmod char_dev.ko

加载完成后，输入

#dmesg | tail -12　（输出最后 12 行）查看加载信息。



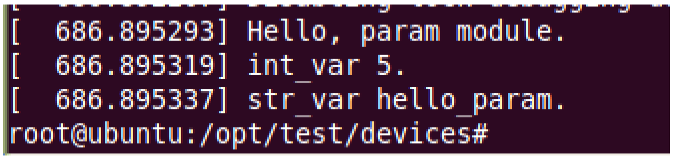
```
[  600.827130] My device register success !, major = 250 name = char_dev
root@ubuntu:/opt/test/devices#
```

加载带参数 hello_param 模块，键入

insmod hello_param.ko int_var=5 str_var=”hello param!”

加载 hello_param 模块

加载完成后，输入#dmesg | tail -12　（输出最后 12 行）查看加载信息

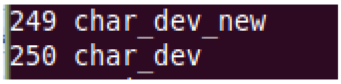


```
[  686.895293] Hello, param module.
[  686.895319] int_var 5.
[  686.895337] str_var hello_param.
root@ubuntu:/opt/test/devices#
```

2、为模块创建对应的虚拟设备

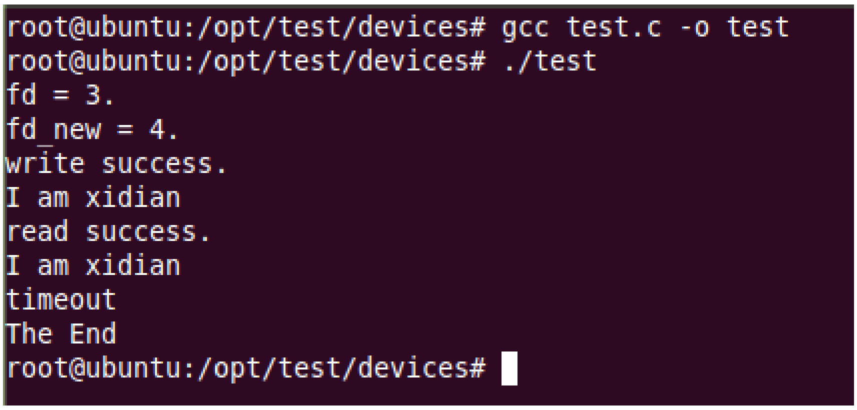
mknod /dev/char_dev c 250 0

mknod /dev/char_dev_new c 249 0



```
249 char_dev_new
250 char_dev
```

在虚拟机下用 gcc 编译 test.c 文件，运行，可以看到 pool 调用两个设备的信息



```
root@ubuntu:/opt/test/devices# gcc test.c -o test
root@ubuntu:/opt/test/devices# ./test
fd = 3.
fd_new = 4.
write success.
I am xidian
read success.
I am xidian
timeout
The End
root@ubuntu:/opt/test/devices# 
```

恭喜你到达这里，意味着即将功德圆满。中神通张师尊携全真四子张冯魏李：祝您一路顺风！