A sample Linux Character Device Driver

I Introduction

Linux Character Diver is a dynamic loadable Linux module, which contains all the routines for the services of real character hardwares.

Before reading this document, we assume the reader has basic understanding of Linux device drivers. If not so, you can read some books about it. For example, Phil Cornes's <<The Linux A – Z >>, or so forth.

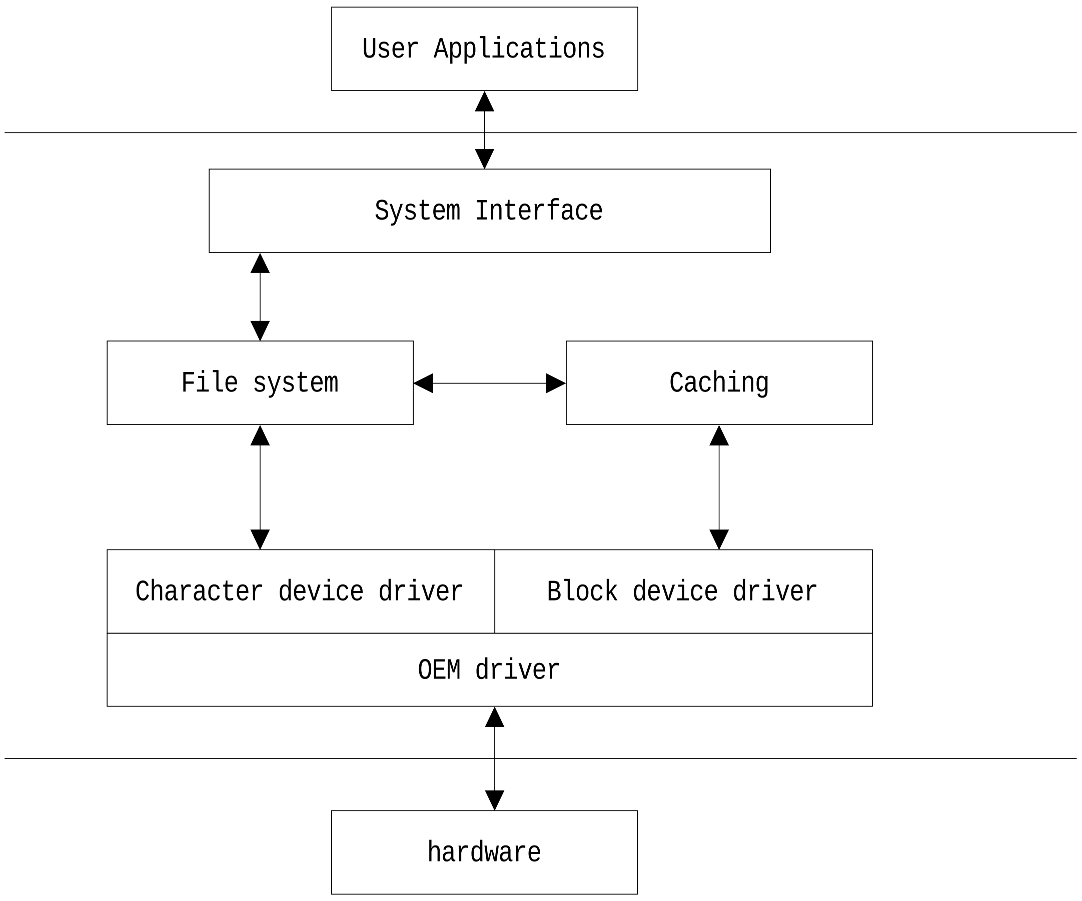
Constrained by the hardware supporting, this document provides a skeleton Linux character device driver, which needs no real hardware supporting. If ported for real use, the reader can rewrite all the specified routines for hardware supporting.

This device driver is implemented and tested in kernel version 2.4.7-8 (Redhat Linux 7.2).

II Character and Block Device Drivers

Before discussing about the character device drivers, we take a brief look at the differences and similarities of character device drivers and its counterpart of block device driver.

The architecture of the device driver management in the Linux kernels is illustrated in the following figure:



In the above figure, the difference between character device driver and the block device driver can be observed at ease.

Generally saying, Linux character device drivers can be easily visited and activated by using user level application interfaces provided for the file operations (for example, open()s, read()s and write()s).

Whereas block device driver is more complicated than character ones, as the user programming interfaces provided by them can only be authorized to interact with Caching.

For the convenience of implementation and maintenance, if no real hardware is involved, the approach of character device driver is far more attractive than the block ones.

Are the character device drivers much slower than the block ones, as the name explicitly means they can only transfer data with the real hardwares byte-by-byte? Not absolutely. That is true that all build in character device drivers are much slower than the block device drivers in Linux. But they are all supporting the real hardwares. It is the hardwares that limit the speed!

If there is no real hardware supporting in the implementation, the speed of data transferring between user applications and the device driver is decided only by the speed of copies between user level applications and the kernel (for example the copy_from_user and copy_to_user). By this means, the speed of data transferring are the same for the character ones and the block ones.

For example, if the device driver is build up for managing the network communications, there will be no difference in the aspect of speed between the character ones and the block ones. The speed is ultimately decide by your networking.

III Source Code with notes

The sample skeleton character device driver in the paper does simple things. If somebody writes to this character device, the message is saved in the kernel space by a link. Later, somebody else reads from this device. The message written by the former is retrieved.

Besides the device driver, you should firstly make a device driver file by the command lines:

```
#makenode /dev/tdd c 30 0
```

And after the installation of the device driver by inserting the module, you should write user level applications to test the implementation. The source codes of testing is also exhibited in the following section.

The source codes for the device driver (driver.c):

```
/*for the module*/
#define __KERNEL__
#define MODULE

/*include the head file involved    in the project*/
#if defined(MODVERSIONS)
#include <linux/modversions.h>
#endif

#include <asm/types.h>
#include <asm/atomic.h>
#include <asm/uaccess.h>
#include <asm/signal.h>
#include <asm/unistd.h>
#include <asm/semaphore.h>


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

```
#include <linux/init.h>
#include <linux/timer.h>
#include <linux/list.h>
#include <linux/types.h>
#include <linux/netdevice.h>
#include <linux/config.h>
#include <linux/random.h>
#include <linux/spinlock.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/inet.h>
#include <linux/udp.h>
#include <linux/netfilter_ipv4.h>
#include <linux/utsname.h>
#include <linux/in.h>
#include <linux/if.h>
#include <linux/netdevice.h>
#include <linux/if_arp.h>
#include <linux/fs.h>
#include <linux/string.h>
#include <linux/errno.h>
#include <linux/sched.h>
#include <linux/smp_lock.h>
#include <linux/vmalloc.h>
#include <linux/un.h>
#include <linux/unistd.h>
#include <linux/wrapper.h>
#include <linux/wait.h>
#include <linux/ctype.h>
#include <linux/igmp.h>
```

```
#define TDD_WRITE      0
#define  TDD_READ      1
#define  FALSE        0
#define  TRUE         1
#define MAX_BUF        120
#define  TDD_TRON      (('M'<<0)|0x01)
#define  TDD_TROFF     (('M'<<0)|0x02)

struct tdd_buf{
       int buf_size;
       char buffer[MAX_BUF];
       struct tdd_buf * link;
};

#define  MAJORDEVICE      30

static inttdd_trace;
static intwrite_busy;
static intread_busy;
static struct tdd_buf   *qhead;
static struct tdd_buf   *qtail;

static size_t  tdd_read(struct file *, char *, size_t, loff_t * );
static size_t  tdd_write(struct file *, const char *, size_t, loff_t * );
static inttdd_ioctl(struct inode *, struct file *, unsigned int, unsigned long );
static inttdd_open(struct inode *, struct file * );
static inttdd_release( struct inode *, struct file * );

struct file_operations tdd_fops={
       read:       tdd_read,
       write:      tdd_write,
       ioctl:      tdd_ioctl,
       open:       tdd_open,
       release: tdd_release,
};

void tdd_init( void )
{
       if ( register_chrdev( MAJORDEVICE, "tdd", &tdd_fops) )
           printk( "Can not register tdd driver as Major Device 30\n" );
       else
           printk( "Tiny device dirver registered successfully.\n" );


       qhead = NULL;
       write_busy = FALSE;
       read_busy = FALSE;
       return;
}

void tdd_uninstall(void)
{
       unregister_chrdev( MAJORDEVICE, "tdd" );
       qhead = NULL;
       write_busy = FALSE;
       read_busy = FALSE;
       return;
}

int init_module( void )
```

```
{
        printk( "initializing...." );
        tdd_init();
        return 0;
}

void cleanup_module( void )
{
        tdd_uninstall();
        printk( "Bye bye...\n\n\n" );
}

static inttdd_open(struct inode * inode, struct file * file)
{
        printk( "tdd_open" );

        switch( MINOR( inode-> i_rdev ) ){
        case TDD_WRITE:
                if( write_busy )
                        return -EBUSY;
                else
                        write_busy = TRUE;
                return 0;
        case TDD_READ:
                if( read_busy )
                        return -EBUSY;
                else
                        read_busy = FALSE;
                return 0;
        default:
            return -ENXIO;
        }
    }

    static int tdd_release( struct inode * inode, struct file * file)
    {
        printk( "tdd_release\n" );

        switch( MINOR(inode->i_rdev) ){
        case TDD_WRITE:
            write_busy = FALSE;
            return 0;
        case TDD_READ:
            read_busy = FALSE;
            return 0;
        }
        return 0;
    }

    static size_t tdd_write(struct file * file, const char *buffer, size_t count, loff_t
*offset)
    {
        int   len;
        struct tdd_buf      *ptr;

        printk( "tdd_write\n" );

        if ( (ptr=kmalloc( sizeof(struct tdd_buf), GFP_KERNEL) ) == NULL )
            return -ENOMEM;
```

```
	len = count<MAX_BUF?count:MAX_BUF;

	if ( copy_from_user( (void*)(ptr->buffer), (void*)buffer, len ) != 0 ){
		printk( "Copy from user Error!\n" );
		return -ENOMEM;
	}
	printk( "W" );

	ptr->link = NULL;

	if( qhead == NULL )
		qhead = ptr;
	else
		qtail -> link = ptr;

	qtail = ptr;
	printk( "\n" );

	ptr->buf_size = len;
	return len;
}

static size_t tdd_read(struct file * file, char * buffer, size_t count, loff_t * offset)
{
	int	len;
	struct tdd_buf *ptr;

	char buffer[100];

	printk( "tdd_read\n" );
```

```
        if (qhead == NULL)
            return -ENODATA;

        ptr = qhead;
        qhead = qhead->link;

        len = count<ptr->buf_size?count:ptr->buf_size;

        if ( copy_to_user( (void*)buffer, (void*)(ptr->buffer), len ) != 0 ){
            printk( "Copy to User Error!\n" );
            return -1;
        }

        kfree( ptr );

        return len;
    }

    static int tdd_ioctl(struct inode * inode, struct file * file, unsigned int cmd,
unsigned long arg)
    {
        printk( "tdd_ioctl\n" );

        switch( cmd ){
        case TDD_TRON:
            tdd_trace = TRUE;
            return 0;
        case TDD_TROFF:
            tdd_trace = FALSE;
        return 0;
    default:
        return -EINVAL;
    }
}
```

The testing user programs:

1) Write to the device (writedev.c):

```
#include <stdint.h>
#include <inttypes.h>
#include <time.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(int argc, char **argv){
    char buffer[100];
    int devfile = -1;
    int req_sz = 100;

    strcpy( buffer, "Hello World!" );

    devfile = open("/dev/tdd", O_WRONLY);
    if(devfile < 0){
        perror("open");
        exit(-1);
    }
    if((write(devfile, buffer, req_sz))!=req_sz){
        printf( "Device file Write Error!\n" );
    }
    close(devfile);
    return(0);
}
```

2) Read from the device (readdev.c):

```
#include <stdint.h>
#include <inttypes.h>
#include <time.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(int argc, char **argv){
    char buffer[200];
    int devfile = -1;
    int read_sz = 0;

    devfile = open("/dev/tdd", O_RDONLY);
    if(devfile < 0){
        perror("open");
        exit(-1);
        }

        read_sz = read(devfile, buffer, 200);
        if(read_sz==0){
            printf("No data available.\n");
            close(devfile);
            return 0;
        }

        printf( "The actual read size is:%d.\n", read_sz );
        printf( "The read content is:%s\n", buffer );
        close(devfile);
        return(0);
    }
```

That is all. Enjoy that!

V Copyrights

If you have some suggestions, bugs-reportings, or acknowledgements, please send mails to the author.