دانشگاه صنعتی اصفهان

دانشکده برق و کامپیوتر

آزمایشگاه سیستم عامل

دستور کار جلسه چهارم

مهر ۱۴۰۱

فهرست مطالب

۱ IOCTL

توابعی که در جلسه قبل برای درایور یک دستگاه کاراکتری تعریف کردیم به شکل زیر بودند.

```
static struct file_operations fops = {
    .open = iut_open,
    .read = iut_read,
    .write = iut_write,
    .release = iut_release,
};
```

در کرنل لینوکس یک System call دیگر به نام ioctl وجود دارد که برای ارتباط با دستگاه کاراکتری کاربرد دارد. این system call در صورتی که قصد ارسالی دستوری متفرقه برای دستگاه را داشته باشیم به کار می‌آید. تابع متناظر این system call در یک درایور به شکل زیر در file_operations تعریف می‌شود.

```
static struct file_operations fops = {
    .open = iut_open,
    .read = iut_read,
    .write = iut_write,
    .release = iut_release,
    .unlocked_ioctl = iut_ioctl,
};
```

با استفاده از ioctl میتوان دستورات مختلفی تعریف کرد و متناظر با آن دستورات در درایور فعالیت‌هایی را تعریف کرد. برای تعریف تابع ioctl باید تابعی با ساختار زیر تعریف کرد.

```
static long iut_ioctl(struct file *file,  unsigned int req, unsigned long pointer)
```

- پارامتر اول اطلاعاتی درباره فایلی که روی آن کار می‌کنیم، در اختیار قرار می‌دهد.
- پارمتر دوم دستور را مشخص می‌کند. همانطور که مشخص است این پارامتر از نوع unsigned int است. ولی هر عددی را نمی‌توان برای این پارامتر استفاده کرد. ماکرو _IO(chr, number)(برای مطالعه بیشتر درباره این ماکرو به اینجا مراجعه کنید) یک عدد مناسب برای این پارامتر تولید می‌کند.
- پارامتر سوم: پارامتر سوم اشاره گری به فضای حافظه کاربری است. این پارامتر برای ارسال داده از فضای کاربر به فضای کرنل استفاده می‌شود. برای استفاده از این پارامتر می‌توان یک نوع داده تعریف کرد و اشاره گر به آن داده را در دستور IOCTL ارسال کرد.

۲ یک IOCTL ساده

به مثال زیر توجه کنید. در این مثال سه فایل داریم. فایل اول util.h است که در آن دستورات و یک ساختمان داده تعریف شده است. در فایل driver.c کد درایور تعریف شده است. به تابع iut_init و iut_ioctl در این فایل دقت کنید.

فایل Utils.h:

```
enum {
   Command1       = _IO('A', 1),
};

struct iut_data {
    int arg1;
};
```

فایل Driver.c:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/miscdevice.h>
#include <linux/module.h>
#include <linux/netdevice.h>
#include <linux/list.h>
#include <linux/version.h>
#include <linux/wait.h>
#include <asm/uaccess.h>
#include "utils.h"

#define DEVICE_NAME "iut_device"
static int major;

static long iut_ioctl(struct file *file,
                      unsigned int req,
                      unsigned long pointer) {
    struct iut_data *data;
    data = kzalloc(sizeof(struct iut_data), GFP_KERNEL);;
    if (!data)
        return -ENOMEM;
    if (copy_from_user(data, (void *) pointer, sizeof(struct
iut_data))) {
        return -EFAULT;
```

```
    }
    switch (req) {
        case Command1:
            printk(KERN_INFO
            "Command1 %d\n", data->arg1);
            return 1;
    }

    return -EINVAL;
}

static int iut_open(struct inode *inode, struct file *file) {
    printk("device opened\n");
    return 0;
}

static int iut_release(struct inode *inode, struct file *file)
{
    printk("device closed\n");
    return 0;
}

static const struct file_operations fops = {
        .open           = iut_open,
        .release        = iut_release,
        .unlocked_ioctl         = iut_ioctl,
};
static int __init iut_init(void) {
    major = register_chrdev(0, DEVICE_NAME, &fops);
    if (major < 0) {
        printk(KERN_ALERT
        "Device001 load failed!\n");
        return major;
    }
    printk(KERN_INFO
    "iut device module has been loaded: %d\n", major);
    return 0;
}

static void __exit iut_exit(void) {
    unregister_chrdev(major, DEVICE_NAME);
    printk(KERN_INFO
    "iut device module has been unloaded.\n");
}
```

```
module_init(iut_init);
module_exit(iut_exit);
MODULE_LICENSE("GPL");
```

فایل user.c:

```
#include <sys/ioctl.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "utils.h"
#include <stdio.h>


int main(int argc, char **argv)
{
    int fd, ret;
    struct iut_data *iut_data1 = malloc(sizeof(struct
iut_data));

    memset(iut_data1, 0, sizeof(struct iut_data));
    iut_data1->arg1 = 100;

    fd = open("/dev/iut_device", O_RDWR);
    if (fd < 0) {
        perror("open");
        exit(0);
    }
    ret = ioctl(fd, Command1 , iut_data1);
    if (ret < 0) {
        perror("ioctl");
        exit(0);
    }
    close(fd);
    return 0;
}
```