
Project 1

陈思贝 (718030290013)

1 实验过程

所有模块均写在同一个 c 文件 homework1.c 中。

1.1 MODULE 1

模块 1 实现过程较为简单，仅需在 `__init` 和 `__exit` 中适当的位置加入以下指令即可。

```
printk(KERN_INFO "Hello, world\n");
printk(KERN_INFO "Goodbye, world\n");
```

1.2 MODULE 2

对模块 2 的实现需要预先定义以下变量。

```
static int paramInt;
static char *paramStr;
static int paramArr[3];
static int pr_arr;
```

然后通过 `module_param` 从传入参数中接收变量。其中 `int` 和 `charp`（字符串）可直接传入，数组则可用专门的 `module_param_array` 进行接收，需要提前确定数组的个数并指定大小。

```
module_param(paramInt, int, 0644);
module_param(paramStr, charp, 0);
module_param_array(paramArr, int, &pr_arr, 0444);
```

输出传入参数的过程与模块 1 的进程类似。

```
printk(KERN_INFO "Param-int:%d;\n", paramInt);
printk(KERN_INFO "Param-str:%s;\n", paramStr);
for (i = 0; i < (sizeof paramArr / sizeof (int)); i++) {
    printk(KERN_INFO "Param-arr[%d]: %d\n", i, paramArr[i]);
}
```

1.3 MODULE 3

模块 3 的实现难度大了许多，在 `\proc` 中创建文件以及退出时移除文件只需以下指令：

```
proc_create("test", 0, NULL, &hello_proc_fops);
remove_proc_entry("test", NULL);
```

重点在于 `hello_proc_fops` 的类型从 `file_operations` 变成了 `proc_ops`。因此要对课件中的实现过程稍作更改：

```
static const struct proc_ops hello_proc_fops = {
```

```

        .proc_open = hello_proc_open,
        .proc_release = single_release,
        .proc_read = seq_read,
        .proc_lseek = seq_lseek
    };

```

因为建立的是只读文件，需要对打开文件时的操作进行定义，实现方法如下：

```

static int hello_proc_show(struct seq_file *m, void *v) {
    seq_printf(m, "Hello proc!\n");
    return 0;
}

static int hello_proc_open(struct inode *inode, struct file *file) {
    return single_open(file, hello_proc_show, NULL);
}

```

1.4 MODULE 4

模块 4 的难点在于对读写文件的读写功能的实现。创建和移除文件夹以及读写文件实现方法如下：

```

static struct proc_dir_entry *dir;

dir = proc_mkdir("hello_dir", NULL);
proc_create("test", 0660, dir, &hello_fops);

remove_proc_entry("test", dir);
remove_proc_entry("hello_dir", NULL);

```

与模块 3 一样，模块 4 中也用到了最新的proc_ops类型。读的实现与模块 3 基本一致。写的功能通过copy_from_user实现。从用户的内存中把字符串复制进入内核的内存中，并在读取该字符串的时候将其提供给读取模块。

```

static char *str = NULL;

static ssize_t hello_write(struct file *file, const char __user *buffer, size_t count,
    loff_t *f_pos) {
    char *tmp = kzalloc((count+1),GFP_KERNEL);
    if(!tmp) return -ENOMEM;
    if(copy_from_user(tmp,buffer,count)){
        kfree(tmp);
        return -EFAULT;
    }
    kfree(str);
    str=tmp;
}

```

```

    return count;
}

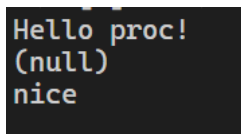
static int hello_show(struct seq_file *m, void *v) {
    seq_printf(m, "%s\n", str);
    return 0;
}

static int hello_open(struct inode *inode, struct file *file) {
    return single_open(file, hello_show, NULL);
}

static const struct proc_ops hello_fops = {
    .proc_open = hello_open,
    .proc_release = single_release,
    .proc_read = seq_read,
    .proc_write = hello_write,
    .proc_lseek = seq_lseek
};

```

2 实验效果截图

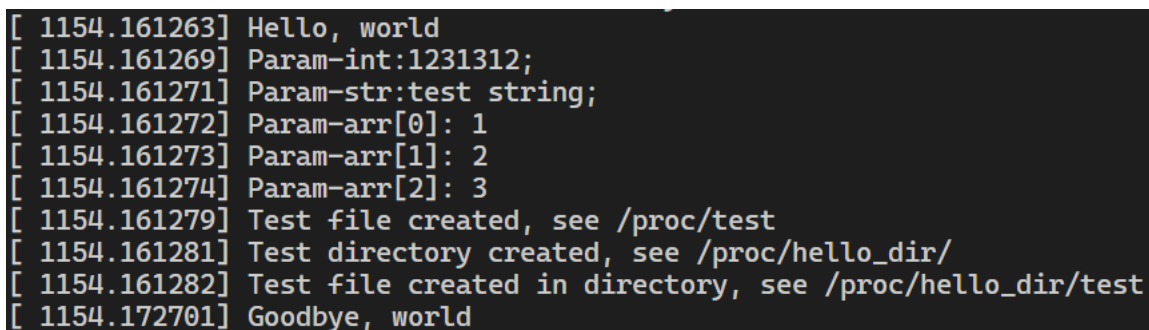


```

Hello proc!
(null)
nice

```

图 1: 输出内容截图



```

[ 1154.161263] Hello, world
[ 1154.161269] Param-int:1231312;
[ 1154.161271] Param-str:test string;
[ 1154.161272] Param-arr[0]: 1
[ 1154.161273] Param-arr[1]: 2
[ 1154.161274] Param-arr[2]: 3
[ 1154.161279] Test file created, see /proc/test
[ 1154.161281] Test directory created, see /proc/hello_dir/
[ 1154.161282] Test file created in directory, see /proc/hello_dir/test
[ 1154.172701] Goodbye, world

```

图 2: 内核日志截图

3 实验心得

本次实验掌握了针对内核进程最基础的操作：写入内核日志，读取传入的参数，在`\proc`中创建不同权限的文件和文件夹，以及对于可读写文件的实现。同时也对模块的加载和卸载环节以及`Makefile`有着更为深刻的了解。为接下来的内核模块编程打下了基础。