

Device Driver and Userspace Continued

**Advanced Embedded Linux
Development**
with **Dan Walkes**



University of Colorado **Boulder**

Learning objectives:

Summary of Kernel Module Load
Process

Memory Allocation

Read and Write Handling

Kernel Module Load/Tools Overview



Linux Kernel

procfs/devices:

241 mydriver

fops:

241 : &my_driver_fops

register_chrdev() <&0xXXXXXXXX>:

allocate new chrdev

Register fops with chardev

mydriver.ko

®ister_chrdev = 0xXXXXXXXX

&my_driver_fops.open = 0xYYYYYYYY

mydriverparam = XXXX

module_entry function:

call register_chrdev with &my_driver_fops

open() <&0xYYYYYYYY>:

read /proc/devices

module_load script

```
major=$(awk "\$2==" "$module\  
{print \$1}" /proc/devices)
```

mknod

```
mknod /dev/mydriver c ${major} 0
```

open

user space application:

open /dev/mydriver

root filesystem:

c 241 0 /dev/mydriver

insmod or modprobe

mydriverparam=XXXX

mydriver.ko

®ister_chrdev = ?

&my_driver_fops.open = ?

make

mydriver.c

Kernel Memory Allocation

- `kmalloc(size_t size, int flags);`
 - Size - in bytes (same as `malloc`)
 - flags - will discuss later, for now use `GFP_KERNEL`
- `kfree(void *ptr);`
- Why not just use `malloc()` and `free()`?
 - Those are glibc functions, we don't have access in kernel space.
- Large buffers: use dynamic allocation

read and write handling

- `ssize_t read(struct file *filep, char __user *buff, size_t count, loff_t *offp);`
- `ssize_t write(struct file *filep, char __user *buff, size_t count, loff_t *offp);`
 - `buff` is a user space pointer
 - Cannot be directly accessed by kernel code
- **Why?**
 - On some architectures might not be accessible/valid in kernel at all
 - May be in a RAM page not available
 - Supplied by a user program which should not be trusted

Accessing __user pointers

- References to user space memory
- Use functions below to access
- unsigned long copy_to_user(void __user *to, const void *from, unsigned long count);
- unsigned long copy_from_user(void *to, const void __user *from, unsigned long count);
 - Similar to a memcpy but copy between separate kernel/user memory spaces

Accessing __user pointers

- read Example Implementation

