Device Driver and Userspace Continued

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

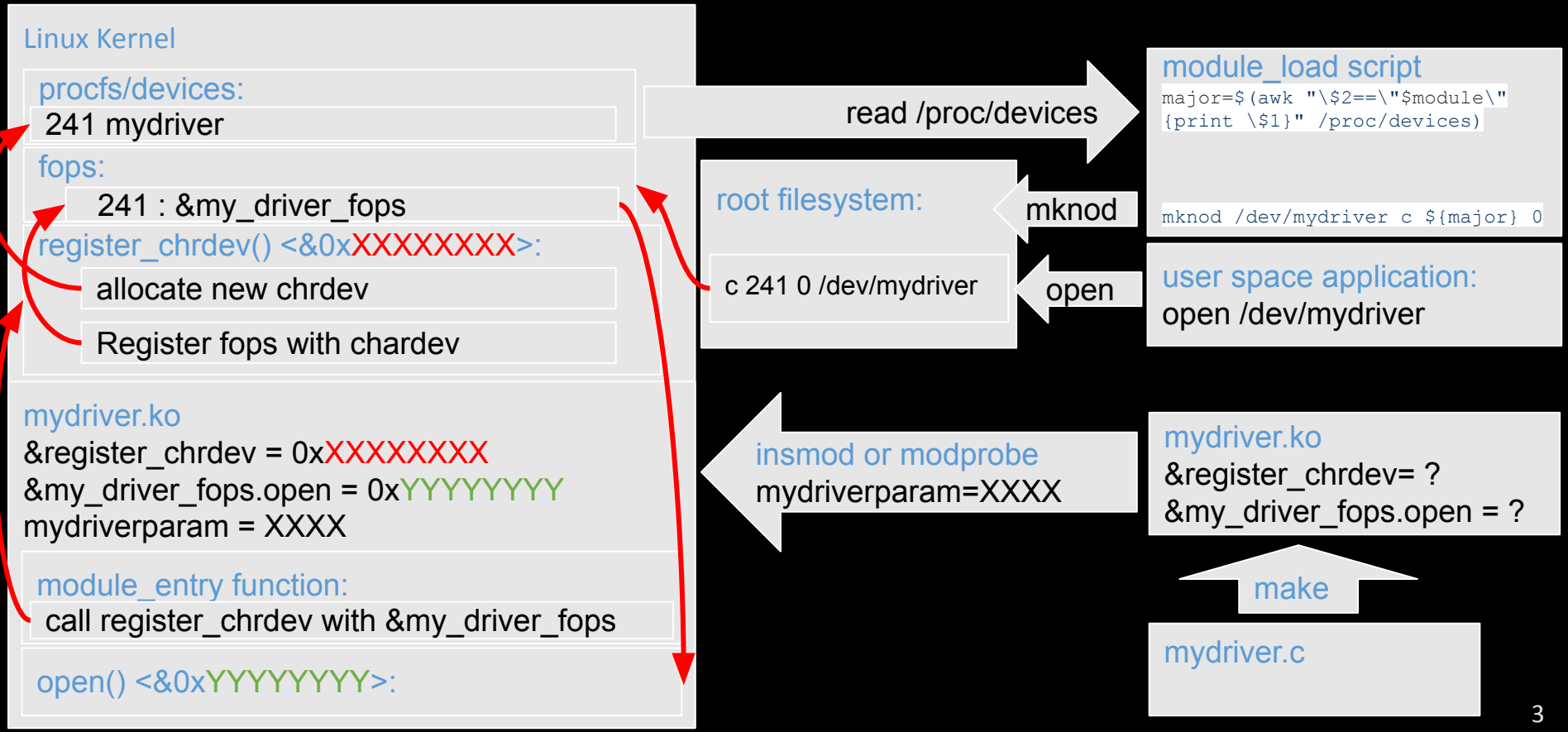
Learning objectives:

Summary of Kernel Module Load Process

Memory Allocation

Read and Write Handling

Kernel Module Load/Tools Overview



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

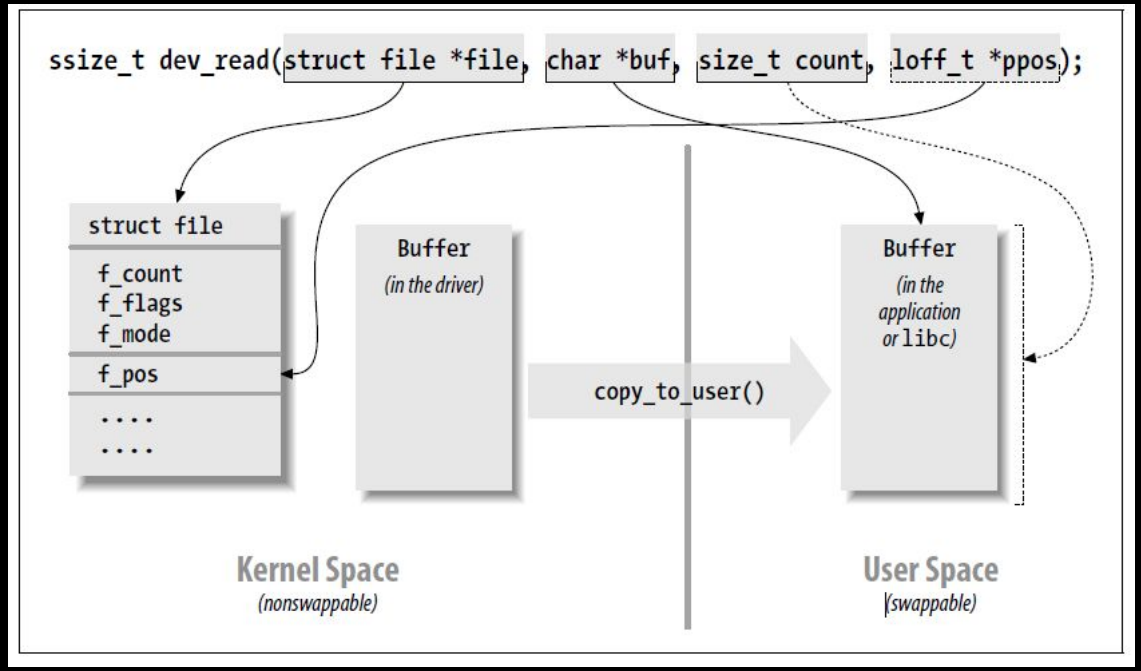
- ssize_t read(struct file *filep, char __user *buff, size_t count, loff_t *offp);
- ssize_t write(struct file *filep, char __user *buff, size_t count, loff_t *offp);
  - buff is a user space pointer
  - Cannot be directly accessed by kernel code
- Why?
  - On some architectures might not be accessible/valid 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



Linux Device Drivers 3rd Edition Chapter 3