1. System Calls are the main communication means to control kernel operations (by drivers).
2. In-band data is constrained to:

**read**(fd, buf, count), write(fd, buf, len)

**settimeofday**(timeval, timezone)

1. Special files in /dev marked with notation:

**c**rw-rw-rw- 1 root wheel 5, 49 Apr 13 00:39 ptys1

1. Major = module (tty, clock, serial ports, disk)
2. Minor = which specific instance (pty49)
3. Kernel data type to store dev node is "dev\_t"
   * Defined in include/linux/types.h
4. Current list in /proc/devices
5. Macros used to extract numbers from dev\_t

* MAJOR(dev\_t dev)
* MINOR(dev\_t dev)
  + Found in include/linux/kdev\_t.h

**Registering a new char**

1. First, allocate a region

int alloc\_chrdev\_region(dev\_t \*dev, unsigned int firstminor, unsigned int count, char \*name)

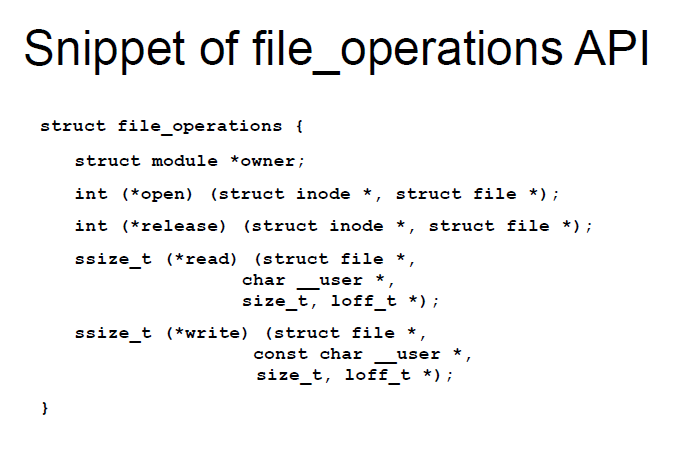
* + alloc\_chrdev\_region() just configures device internal to kernel.
  + No linkage to anything in the upper device subsystem

1. Cleanup when finished.

void unregister\_chrdev\_region(dev\_t first, unsigned int count)

**LINKING MODULES**

1. Structure "file\_operations" provides function pointers into system call interface.



1. cdev represents char devices inside the kernel.
2. Initialized with:

void cdev\_init(struct cdev \*cdev, struct file\_operations \*fops)

1. Added with:

int cdev\_add(struct cdev \*cdev, dev\_t num, unsigned int count)

* + file\_operations must be configured and ready to go before “cdev\_add()”

1. Cleaned up with:

void cdev\_del(struct cdev \*cdev)

1. Need mechanism to copy data into kernel buffers, out of kernel buffers
   * copy\_from\_user
     + [copy\_from\_user](https://elixir.bootlin.com/linux/latest/ident/copy_from_user)(void \*[to](https://elixir.bootlin.com/linux/latest/ident/to), const void [\_\_user](https://elixir.bootlin.com/linux/latest/ident/__user) \*[from](https://elixir.bootlin.com/linux/latest/ident/from),

unsigned long n)

* + copy\_to\_user
  + Example 5 shows use of this but no prototype to be found. Look at lxr?

**PRINTING FROM KERNEL**

1. printk(KERN\_INFO "chainlink=%d\n", chain);
   * KERN\_EMERG, KERN\_ALERT, KERN\_CRIT, KERN\_ERR, KERN\_WARNING, KERN\_NOTICE, KERN\_INFO, KERN\_DEBUG
2. tail -f /var/log/messages
   * filtered by kernel param "loglevel=n"
   * See <linuxsrc>/Documentation/kernel-parameters.txt
3. dmesg
   * not filtered, all msgs show up
   * not saved on disk
4. pr\_info() and friends
   * Friendly wrappers around printk
   * Annotates and stamps who printed the message
   * Can be compiled out based on debug levels
   * Makes printk more portable
5. trace\_printk()
   * Part of the ftrace function-tracer framework
   * Unbuffered, has much less impact to performance/timing
   * More desirable to use during interrupts
   * Can be enabled/disabled on the fly
6. Code warnings
   * BUG(), BUG\_ON(expr)
   * WARN(), WARN\_ON(expr), WARN\_ONCE()
   * BUG stops the kernel thread
   * Both produce stack dump output
   * <http://lxr.free-electrons.com/source/drivers/net/e1000/e1000_main.c#L551>