Sheaff

Nicholas LaJoie
ECE 331
2/28/17

* Board should be ready by Thursday (kernel driver proj.)
  ⇒ ATxMEGA128A3U

---

* iNodes Refresher:

ps agx

- INODE TABLE (split across all block groups)
  - → Contains metadata for files
    - → size
    - → type
      - → char dev
      - → socket
      - → block special
      - → symbolic link
      - → dir.
      - → reg. file
      - → FIFO (named pipe)

      IPC (interprocess communication)
    - → permissions
      - → user (u)
      - → group (g)
      - → other (o)

        u g o
        rwx rwx rwx

        9 bits
    - → timestamp
    - →

* Dirent (dir. entry)

  Inode # | len (entry) | len (name) | filename

Block Group

- Inode (Part) Table
- Bitmaps → used data blocks
  " inode entries
- Data
- Copy of the Super Block
- Block descriptor

---

* might as well get 32 bits
  - => suid (execute as root)
  - => sgid
  - => Sticky bit (used in (temp) directory)
* type has 5 bits of those

---

diff -ur ← make your own patch
  ↑ unicode

patch < xyz.patch

---

* Process 0 waits for interupt (low power energy)

* config files live in /etc/
* ~~(santos)~~ compiled source code, executable
  => /usr/local/bin/
* log files => /var/log/
* man pages => /usr/shr/man/
* /usr/include (headers/defines)
* shell:
  * where is santos defined
  * >> grep -r santos /usr/include

or

grep santos /usr/include/*

grep '^#define\s+santos' /usr/include/*

(grep -P
↑ works w/ \s, etc.)

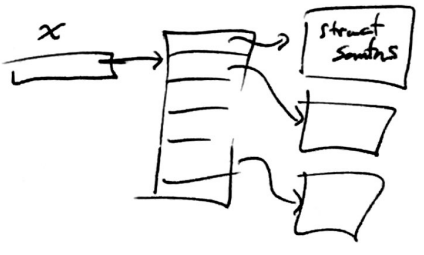
~~malloc(42)~~ x = (struct santos *) malloc(42);
(struct santos *) gives constant
↑ sizeof( )
(be careful, sizeof(ptrs) might not be what you want...)

```
struct ~~santos~~ **x;

struct santos {
        int time;
        char x;
};

x = (struct santos **) malloc(42 * sizeof(struct santos *));
for (i=0; i<42; i++) {
        x[i] = (struct santos *) malloc(sizeof(struct santos));
        if (x[i] == NULL) {
                perror("out of mem");
                for (j=0; j<i; j++) {
                        free(x[j]);
                }
                free(x);
                return 4;
        }
```

error check here



Homework over Break:

{-> get rudimentary driver compiled & loaded in kernel}

---

* interface for hardware is in /dev/
  - => i2c-1 is interface to that hardware
  - => do something it:
    - open, write/read, close
      - sometimes you have to set baud rates (ioctl sets the settings)
* kernel:
  - ptrs to functions, populate it w/ptrs to my function
* "close" actually calls "release" in kernel space

struct file_operations

This describes how the VFS can manipulate an open file. As of kernel 2.6.13, the following members are defined:

```
struct file_operations {
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*aio_read) (struct kiocb *, char __user *, size_t, loff_t);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*aio_write) (struct kiocb *, const char __user *, size_t, loff_t);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *, int datasync);
        int (*aio_fsync) (struct kiocb *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*sendfile) (struct file *, loff_t *, size_t, read_actor_t, void *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
        int (*check_flags)(int);
        int (*dir_notify)(struct file *filp, unsigned long arg);
        int (*flock) (struct file *, int, struct file_lock *);
};
```

Again, all methods are called without any locks being held, unless otherwise noted.

llseek:	called when the VFS needs to move the file position index
read:	called by read(2) and related system calls
aio_read:	called by io_submit(2) and other asynchronous I/O operations
write:	called by write(2) and related system calls
aio_write:	called by io_submit(2) and other asynchronous I/O operations
readdir:	called when the VFS needs to read the directory contents
poll:	called by the VFS when a process wants to check if there is activity on this file and (optionally) go to sleep until there is activity. Called by the select(2) and poll(2) system calls
ioctl:	called by the ioctl(2) system call
unlocked_ioctl:	called by the ioctl(2) system call. Filesystems that do not require the BKL should use this method instead of the ioctl() above.
compat_ioctl:	called by the ioctl(2) system call when 32 bit system calls are used on 64 bit kernels.
mmap:	called by the mmap(2) system call
open:	called by the VFS when an inode should be opened. When the VFS opens a file, it creates a new "struct file". It then calls the open method for the newly allocated file structure. You might think that the open method really belongs in "struct inode_operations", and you may be right. I think it's done the way it is because it makes filesystems simpler to implement. The open() method is a good place to initialize the "private_data" member in the file structure if you want to point to a device structure
flush:	called by the close(2) system call to flush a file
release:	called when the last reference to an open file is closed
fsync:	called by the fsync(2) system call
fasync:	called by the fcntl(2) system call when asynchronous (non-blocking) mode is enabled for a file
lock:	called by the fcntl(2) system call for F_GETLK, F_SETLK, and F_SETLKW commands
readv:	called by the readv(2) system call
writev:	called by the writev(2) system call
sendfile:	called by the sendfile(2) system call
get_unmapped_area:	called by the mmap(2) system call
check_flags:	called by the fcntl(2) system call for F_SETFL command
dir_notify:	called by the fcntl(2) system call for F_NOTIFY command
flock:	called by the flock(2) system call

Note that the file operations are implemented by the specific filesystem in which the inode resides. When opening a device node (character or block special) most filesystems will call special support routines in the VFS which will locate the required device driver information. These support routines replace the filesystem file operations with those for the device driver, and then proceed to call the new open() method for the file. This is how opening a device file in the filesystem eventually ends up calling the device driver open() method.