Device Drivers

CS449 Fall 2015
Abstraction via the OS
Software Layers

User
- level I/O software & libraries

Device-independent OS software

Device drivers

Interrupt handlers

Hardware

Operating system (kernel)
Device Drivers

User space

Kernel space

Rest of the OS

Keyboard driver

Disk driver

Keyboard controller

Disk controller
Types of Devices

• **Block Devices**
  – A device that stores data in fixed-sized blocks, each uniquely addressed, and can be randomly accessed
  – E.g., Disks, Flash Drives

• **Character Devices**
  – Device that delivers or accepts a stream of characters
  – E.g., Keyboard, mouse, terminal
Mechanism vs. Policy

• Mechanism – What capabilities are provided
  – E.g. Hard disk driver: exposes the disk as a continuous array of data blocks

• Policy – How to use those capabilities
  – E.g. Data blocks can be organized using a file system, or they can be used as a raw block device
  – E.g. Data blocks may only be accessible by certain users

• Drivers should be flexible by only providing mechanisms not policies
Devfs

- Mounted on `/dev/
- Character and block devices exposed via the filesystem
- `/dev/` typically contains “files” that represent the different devices on a system
- `/dev/console` – the console
- `/dev/fd/` - a process’s open file descriptors
Sysfs

• Mounted on `/sys/

• Exports information about devices and drivers to userspace,

• Can configure aspects of device
Device Drivers in Linux

- Can be compiled into the kernel
- Can be loaded dynamically as Modules
Hello World Module

#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");
static int hello_init(void)
{
    printk(KERN_ALERT "Hello, world\n");
    return 0;
}
static void hello_exit(void)
{
    printk(KERN_ALERT "Goodbye, cruel world\n");
}
module_init(hello_init);
module_exit(hello_exit);
Why printk?

• The kernel does not have access to libraries
• Can’t use printf or many other standard functions (FILE stuff, strtok, etc.)

• Modules are linked against the kernel only
• Kernel provides useful set of common functions like strcpy, strcat, etc.
MODULE_LICENSE

• Informs the kernel what license the module source code is under
• Affects which symbols (functions, variables, etc.) it may access in the kernel

• A GPL-licensed module can access everything
• Certain (or not specifying one) module license will “taint” the kernel
Building & Running

% make
make[1]: Entering directory `/usr/src/linux-2.6.10'
   CC [M] /home/ldd3/src/misc-modules/hello.o
Building modules, stage 2.
MODPOST
   CC /home/ldd3/src/misc-modules/hello.mod.o
   LD [M] /home/ldd3/src/misc-modules/hello.ko
make[1]: Leaving directory `/usr/src/linux-2.6.10'

% su
root# insmod ./hello.ko
Hello, world
root# rmmod hello
Goodbye cruel world
root#
obj-m := hello_dev.o

KDIR := /u/SysLab/shared/linux-2.6.23.1
PWD := $(shell pwd)

default:
    $(MAKE) -C $(KDIR) M=$(PWD) modules
Module Helper Programs

• `insmod` – loads a module
• `rmmod` – unloads a module
• `lsmod` – lists what modules are loaded
• `modprobe` – loads a module checking dependencies
Loading & Unloading a Module
Device Operation Callbacks

```c
struct file_operations {
    struct module *owner;
    int (*open) (struct inode *, struct file *);
    ssize_t (*read) (struct file *, char *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
    ...
};

• Struct containing callbacks passed to OS when driver is loaded
• OS does callbacks on functions on corresponding system call
```
```c
int __init my_init_function(void)
{
    int err;
    /* registration takes a pointer and a name */
    err = register_this(ptr1, "driver");
    if (err) goto fail_this;
    err = register_that(ptr2, "driver");
    if (err) goto fail_that;
    err = register_those(ptr3, "driver");
    if (err) goto fail_those;
    return 0; /* success */

    fail_those: unregister_those(ptr2, "driver");
    fail_that: unregister_this(ptr1, "driver");
    fail_this: return err; /* propagate the error */
}
```
Driver Stacking
Things not to do in the kernel

• Stack allocate big arrays
  – The stack is small, maybe only a single page (4KB)
  – Use kmalloc to allocate heap space
• Leave memory unfreed
  – Will stay around forever until the next reboot!
• Floating point arithmetic
  – Context switch into the kernel does not save floating point registers
Race Conditions

- The kernel will make calls into your module while your initialization function is still running
- Multiple applications will attempt to access your driver simultaneously
User Space Drivers

• Advantages?
  – Full Standard C Library can be linked in
  – Can use a conventional debugger like GDB
  – Problems with driver will not crash entire system
  – Can be swapped to disk using virtual memory

• FUSE – Filesystem in User Space
  – Useful for implementing virtual file systems (e.g. by communicating with cloud storage)