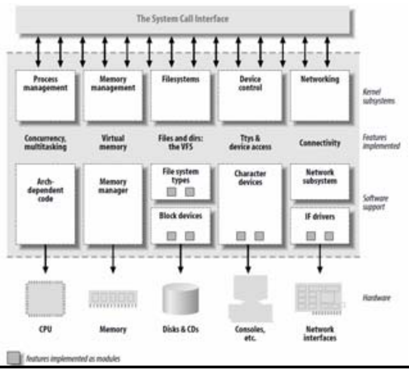
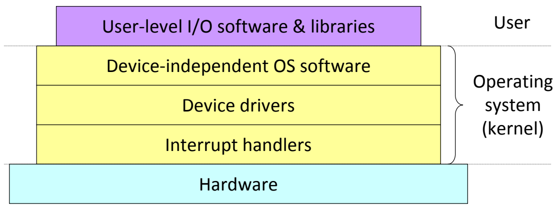
Device Drivers

Jonathan Misurda
jmisurda@cs.pitt.edu

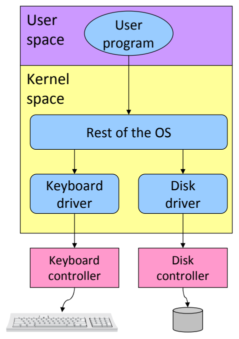
Abstraction via the OS



Software Layers


User-level I/O software & libraries
User
Device-independent OS software
Device drivers
Interrupt handlers
Operating system (kernel)
Hardware

Device Drivers



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 to have (Algorithm)
- Policy – How to use a mechanism (Parameters)
- Drivers should be flexible by only providing mechanisms not policies

Device Drivers in Linux

- Can be compiled into the kernel
- Can be loaded dynamically as Modules

/dev

- Character and block devices can be exposed via a 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

/proc

- Virtual filesystem with information about the system and running programs
- `/proc/cpuinfo` – text "file" containing CPU information

Sysfs

- Exports information about devices and drivers to userspace,
- Can configure aspects of device
- `/sys/`

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);
```

Build and Run

```
% 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#
```

Module Helper Programs

- `insmod` – loads a module
- `rmmod` – unloads a module
- `lsmod` – lists what modules are loaded

- `modprobe` – loads a module checking dependencies

Why printk?

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

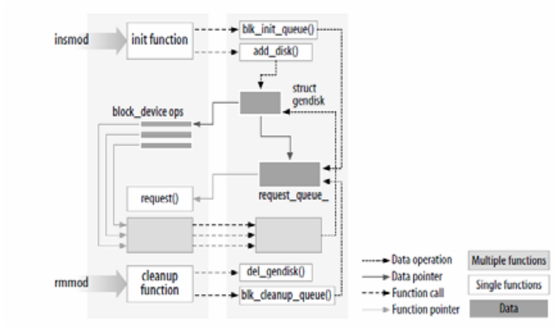
- Modules are linked against the kernel only
- Kernel provides useful set of common functions like strcpy, strcat, etc.

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

Loading a Module



Things you can't do in the kernel

- Stack allocate big arrays
  - The stack is small, maybe only a single page (4KB)
  - Use kmalloc to allocate heap space

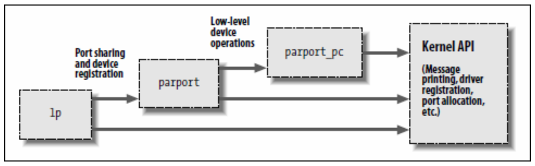
- Floating point arithmetic
  - Context switch into the kernel does not save floating point registers

Error Handling

```
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_that(ptr2, "driver");
   fail_that: unregister_this(ptr1, "driver");
   fail_this: return err; /* propagate the error */
}
```

Driver Stacking


Low-level device operations
Port sharing and device registration
lp
parport
parport_pc
Kernel API
(Message printing, driver registration, port allocation, etc.)

Race Conditions

- The kernel will make calls into your module while your initialization function is still running

Module Parameters

```
insmod hellop howmany=10 whom="Mom"

static char *whom = "world";
static int howmany = 1;
module_param(howmany, int, S_IRUGO);
module_param(whom, charp, S_IRUGO);
```

Parameter Types

- `bool, invbool (int)`
- `charp`
- `int, long, short`
- `uint, ulong, ushort`

- Array parameters, where the values are supplied as a comma-separated list:
  - `module_param_array(name,type,num,perm);`

Permissions

- Module parameters show up as files in the sysfs entry
  - If perm is set to 0, there is no sysfs entry at all

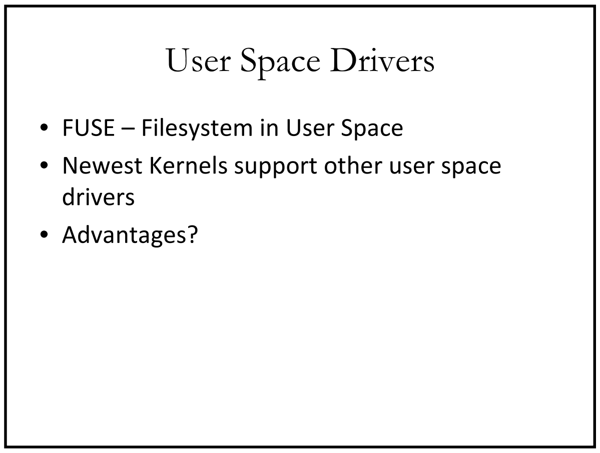
- S_IRUGO – Read Only
- S_IRUGO|S_IWUSR – Writeable by root

Makefile

```
obj-m := hello_dev.o

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

default:
        $(MAKE) -C $(KDIR) M=$(PWD) modules
```



User Space Drivers

- FUSE – Filesystem in User Space
- Newest Kernels support other user space drivers
- Advantages?