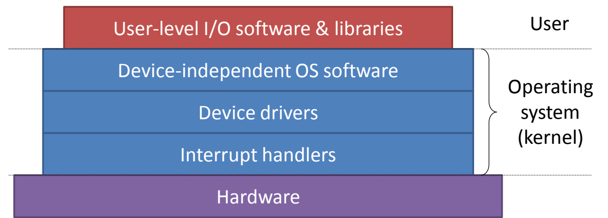
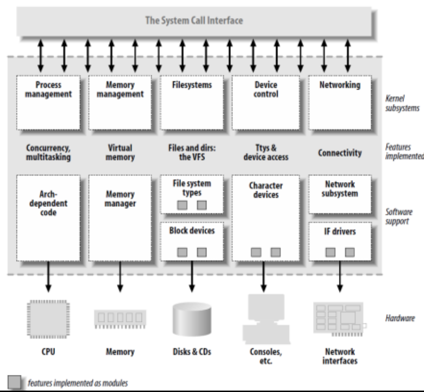
Device Drivers

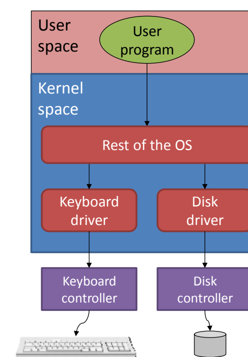
Software Layers



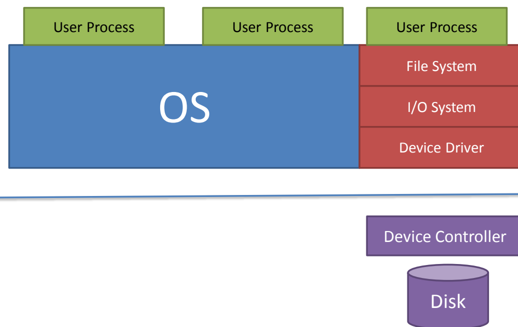
Abstraction via the OS



Device Drivers



I/O System



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
- Policy – How to use a mechanism

- Drivers should be flexible by only providing mechanisms not policies
- Policies should generally be pushed up as much as possible

Device Drivers in Linux

- Can be compiled into the kernel or dynamically loaded as a module

- Allow access to exported module interface
  - Kernel level API
  - Allows access to common kernel functions
    - E.g. Printing to the system log
  - Allows modules to connect to kernel subsystems
    - Register handlers for a specific device
    - Add a new file system
  - Allows modules to export interfaces to user space

File Operations

- Create file with given pathname /a/b/file
  - Traverse pathname, allocate meta-data and directory entry
- Read-from/write-to offset in file
  - Find (or allocate) blocks on disk, update meta-data
- Delete
  - Remove directory entry, free disk blocks
- Rename
  - Change directory entry
- Copy
  - Allocate new directory entry, allocate new space on disk, copy
- Change permissions
  - Change directory entry, or ACL meta-data

Opening Files

- Files must be opened before accessing data
  - User must specify access mode: read and/or write
- OS tracks open files for each process
  - Table containing open file's metadata
  - How do you know which table entry belongs to a file?
  - File descriptor = index into open file table
  - Table contains meta-data + current position and access mode

UNIX file operations

- File operations are implemented inside the kernel
  - Kernel maps operations to the specific file in question
  - Data flows from process memory to/from files on disk
  - But now we have a better picture…
- Data path: process ←→kernel ←→ disk
  - Kernel can intercept data before it reaches the disk
  - Easy mechanism for copying to/from kernel space (device drivers)

/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
- /dev/hda – first hard disk

ProcFS and Sysfs

- ProcFS -- /proc
  - Virtual filesystem with information about the system and running programs
    - e.g `/proc/cpuinfo` – text "file" containing CPU information
- Sysfs -- /sys
  - Exports information about devices and drivers to userspace,
  - Can configure aspects of device

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

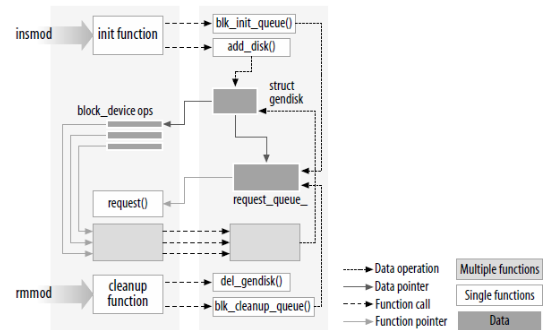
- `insmod` – loads a module
- `rmmod` – unloads a module
- `lsmod` – lists what modules are loaded

- `modprobe` – loads a module checking dependencies
  - 'Modprobe -r' -- removes a module and dependencies

Method of operation

- Device drivers respond to requests
  - They are part of the kernel
  - Interrupts from hardware devices
  - Requests from user space
- During module initialization you register for certain events
  - Interrupts from a specific device
  - File operations on a specific file system
- The Kernel's main responsibility is to route events to the correct device drivers

Loading a Module



Module linking

- The module calls the kernel level API
- The kernel calls the module init function
- There must be a linking step
  - Kernel must be able to find init()
  - Module must be able to find kernel API functions
- When a module is loaded the kernel performs a linking operation on the module
  - Similar to user level linking (ELF format)

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 */
}
```

Race Conditions

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

Why printk?

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

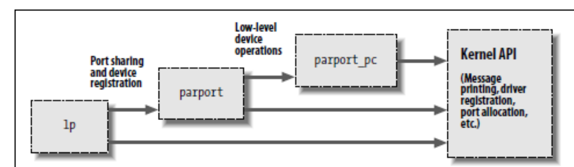
- Modules are linked against the kernel only
- Kernel provides useful set of common functions like strcpy, strcat, etc.

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

Driver Stacking



Module Parameters

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

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

```
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);
```

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

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

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