## Linux Device Drivers

## Device Drivers and Device Controllers

- We already saw that a device controller is the hardware needed within laptop, PC or other computer to connect to an external device.
- A **device driver** is *software* that allows other programs to interface with external devices through the device controllers.
- Device drivers run in kernel mode (some device management software may well run in user mode, but then typically these wouldn't be called 'drivers').

## Device drivers

View from user space:
Have special file in `/dev` associated with it, together with five systems calls:

- `open`: make device available
- `read`: read from device
- `write`: write to device
- `ioctl`: Perform operations on device (optional)
- `close`: make device unavailable

## Kernel side

Each file may have functions associated with it which are called when corresponding system calls are made

`linux/fs.h` lists all available operations on files

Device driver implements at least functions for `open`, `read`, `write` and `close`.

## Automatic recognition of devices

So far: Have seen how devices can be added and used via explicit
Nowadays, automatic HW recognition and insertion and removal of devices important
Requires suitable HW support: Each device responds with unique vendor id and product ID when probed
For certain devices (eg usb) device also responds with type (eg usb-storage)
Each device driver keeps a list for which devices and types it is responsible
All device-related information available to user space via `/sys`-filesystem

Special program goes at installation time through all device drivers and records device id's and type

Steps:

- At boot time, kernel probes devices, which respond with unique id indicating vendor and device type
- For each device found, kernel sends info to userspace
- Special program in userspace (udev) generates entry in `/dev` and loads appropriate module

## Categorising devices

Kernel also keeps track of

- Physical dependencies between devices. Example: devices connected to a USB-hub
- Buses: Channels between processor and one or more devices. Can be either physical (eg pci, usb), or logical
- Classes: Sets of devices of the same type, eg keyboards, mice

## Handling Interrupts in Device Drivers

Normal cycle of interrupt handling for devices:

- Device sends interrupt
- CPU selects appropriate interrupt handler
- Interrupt handler processes interrupt
  Two important tasks to be done:
  - Data to be transferred to/from device
  - Waking up processes which wait for data transfer to be finished
- Interrupt handler clears interrupt bit of device
  Necessary for next interrupt to arrive

Interrupt processing time must be as short as possible
Data transfer fast, rest of processing slow
⇒ Separate interrupt processing in two halves:

- Top Half is called directly by interrupt handler
  Only transfers data between device and appropriate kernel buffer and schedules software interrupt to start Bottom half
- Bottom half still runs in interrupt context and does the rest of the processing (eg working through the protocol stack, and waking up processes)

## Summary

Three topics:

Memory management

- Management of a limited resource $\Rightarrow$ serious effort required
- Need to isolate memory for each process
- If memory demand is too high, need to swap out part of the memory of a process
- Looked at paging and segmentation to achieve this
- Requires hardware support

## Kernel Programming

- Kernel programs have access to all resources with no constraints
- Have separate area of memory for user process and kernel
- have two main for kernel code: one for processing interrutps and one for working for user programs via system calls

## Device drivers

- implement `open`, `read`, `write` and `close` with common structure