Device drivers

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

- `open`: make device available
- `read`: read from device
- `write`: write to device
- `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 commands
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 (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)