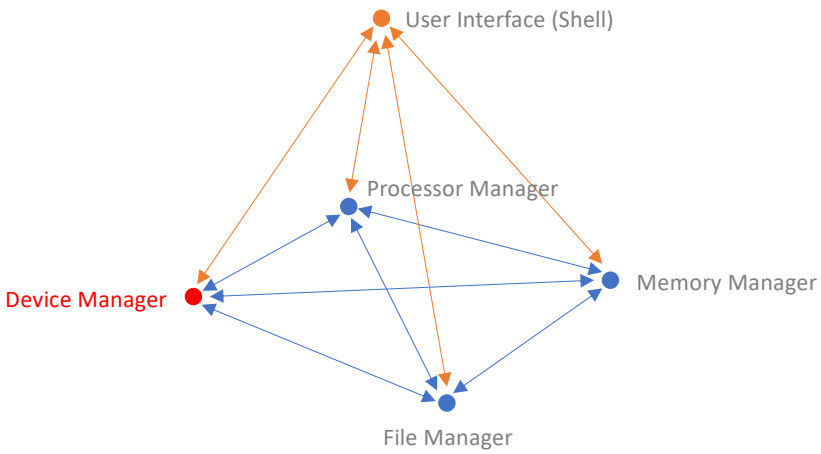
COMP124

Computer Systems

17 | Device Manager | Direct Memory Access

Dr Stuart Thomason

Device Manager

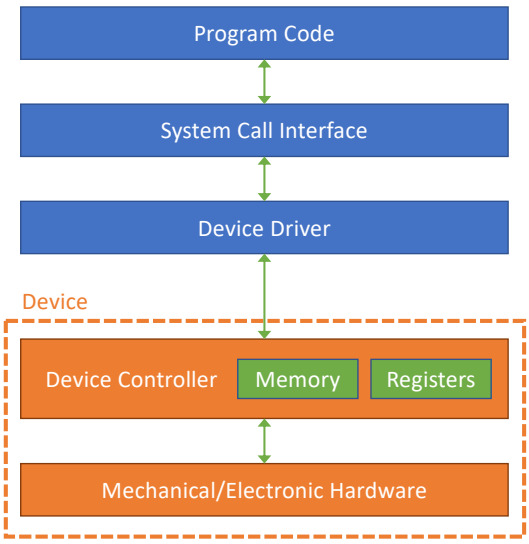


Device Manager

- Monitors every device within the system
- Ensures each process gets fair access to the devices it needs

- Device manager performs various tasks for the operating system…
  - Keeps track of all devices connected to the system
  - Creates virtual files that map onto physical devices
  - Schedules read/write access to devices according to system policy
  - Deals with multiple requests for the same device
  - Communicates with devices during operation
  - Provides standard system calls so other software can use devices
  - Deallocates system resources when devices are no longer needed (eg. unplugged)

Layers of Abstraction



Device Abstraction

- Devices can be categorised by their behaviour
  - Character vs. block
  - Sequential vs. random
  - Shared vs. dedicated
  - Speed of operation
  - Data direction (read-write, read-only, or write-only)

- Device driver hides these differences from the kernel
  - The kernel can treat devices as simple files or streams
  - The same system calls can be used to access any device (to read and write data)
  - For example, by accessing a file in the Linux /dev directory (virtual file system)
  - Device driver sends data to device controller
  - Device controller converts data into electronic signals to operate the hardware

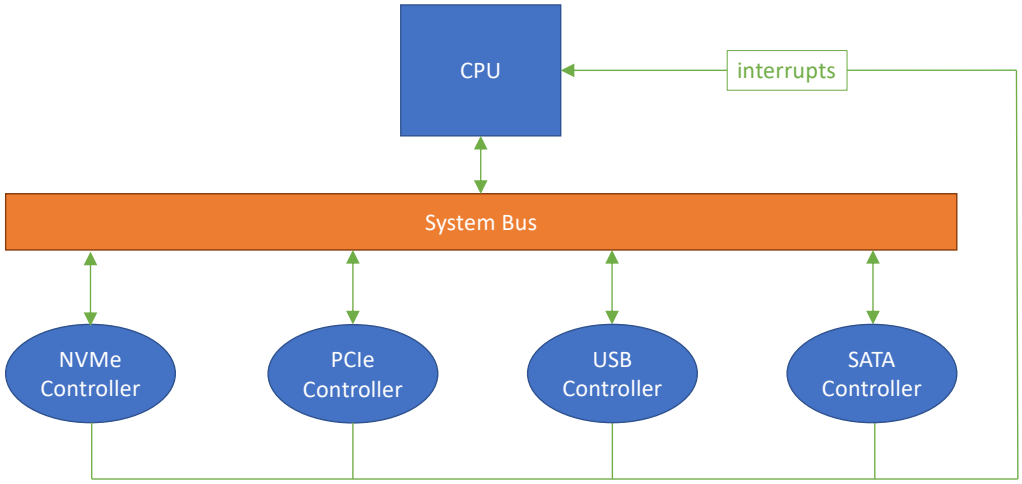
Device Buses

- Devices aren't just things that plug into visible ports
  - External devices are sometimes called peripherals
  - There are other devices inside the system
  - Some are soldered into the main motherboard
  - Others plug into slots or sockets on the motherboard (via a daughterboard)

- There are multiple buses that are designed for different purposes
  - PCIe – Peripheral Component Interconnect Express – general purpose
  - USB – Universal Serial Bus – general purpose
  - NVMe – Non-Volatile Memory Express (often called M.2) – flash disks (SSD)
  - SATA – Serial AT Attachment – mechanical disks (HDD)

- Each bus has its own protocol and format (and speed) for sending and receiving data

System Bus Connections



Communication

- The device controller has several registers to pass commands and data to the device
  - CPU might have special instructions to change and inspect these device registers
  - Can also use the ioctl system call to communicate directly with device controller

- Most operating systems implement memory-mapped I/O
  - Virtual addresses in main memory are mapped on to device registers
  - Allows CPU to use its standard instructions to manipulate device registers
  - Device registers look and behave exactly like normal memory

- Another approach is port-mapped I/O
  - Supported by the Intel instruction sets
  - Devices have a separate address space assigned by the CPU via dedicated pins
  - Can use in and out instructions to move data between EAX and device addresses

Device and Disk Scheduling

- The device manager schedules I/O activities to maximise system performance
  - Minimise time wasted by moving the HDD read/write head
  - Prioritise I/O requests for particular processes
  - Ensure disk access is shared equally between processes
  - Guarantee that certain requests are met before a deadline (in real time systems)
  - Minimise overall wait time for processes in the blocked state

- Each disk has an I/O wait queue, which the scheduler can reorder depending on policy
  - First come first served
  - Shortest seek first
  - Elevator algorithm
  - Completely fair queueing
  - Anticipatory scheduling

Disk Block Scheduling

- The scheduler tries to efficiently read/write disk blocks to minimise process wait time
- Completely fair queueing
  - Requests from each process are placed in their own queue
  - Queues are serviced via time slices according to process priority values
- Anticipatory scheduling
  - Processes might want to deal with data from disk before issuing another request
  - Normally other requests would be serviced during this gap in disk activity
  - But these take the read/write head away from where it was
  - Anticipates further nearby requests by pausing for a short time (a few milliseconds)
- Elevator algorithm
  - Behaves like an elevator in a building
  - Schedules block I/O in the order of travel of the read/write head
  - Only accesses a block when the head is moving in that direction

Buffering

- Consider a program that reads characters from a file, one by one
  - Accessing the disk for each read request is costly
  - Device manager needs to liaise with file manager to access file on disk
  - Disk head (HDD) needs to physically move for each read request

- A more efficient method is to set up an area of memory called a buffer
  - First read request reads a whole chunk of data into the buffer
  - Subsequent requests read from the buffer instead of the disk
  - Only need to involve the disk again when the buffer empties

- This is also used when writing data to a file (and other devices)
  - Put each character into the buffer and write in one operation when buffer is full
  - Danger of losing data, so program can use fflush system call to write the buffer

Buffering

- The system could use double buffering
  - Use one buffer to read/write data while another buffer is being filled/emptied
  - Swap between them to maximise efficiency

- Buffering can happen at multiple places in the device abstraction
  - Software programs
  - Library subroutines
  - Operating system
  - Hardware (eg. memory buffer built into disk drive)

- Buffering can cause inconsistency issues
  - Data physically on disk doesn't match what the system thinks it wrote
  - Important to flush buffers regularly via system call or timer inside device controller

Spooling

- Some devices are non-shareable
  - For example, a printer can only print one document at once
  - Even if it is on a network and available to lots of systems

- These devices use a daemon process called a spooler
  - Imagine spooling tape from a reel
  - Backronym: simultaneous peripheral operations on line (SPOOL)

- Processes send their data to the spooler daemon
  - Spooler creates a temporary file for each process
  - Process writes data into the file
  - No need to worry about availability of device (file will always be available)
  - Spooler sends data to actual device when it becomes free (de-spooling)

Direct Memory Access

- Hardware devices can access main memory independently of the CPU
  - Known as direct memory access (DMA)
  - The CPU initiates the data operation (read or write)
  - Can continue to run other processes while the data transfer takes place
  - Will receive an interrupt when the transfer is complete

- Overall system performance can be maximised by bypassing the CPU
  - Graphics cards can process data faster than the CPU can generate it
  - Disk drives are much slower than the CPU

- The DMA controller has its own registers (MAR, byte counter, control register)
  - CPU loads the registers with the device address and data to be transferred
  - DMA controller carries out the data transfer in the background

Devices in Linux

- Linux creates a virtual file within the /dev directory for each device in the system
  - Devices appear to be simple files or streams
  - Allows programmatic access via standard file handling system calls
  - Device driver requests an entry in the file system when device is enabled
  - Linux uses the devtmpfs virtual disk format with the udev device manager

- Some pseudo-devices are also created

Device	Reading	Writing
/dev/null	Sends EOF character (end of file)	Discards all data
/dev/zero	Stream of NUL characters (zeroes)	Discards all data
/dev/full	Stream of NUL characters (zeroes)	Generates 'disk full' error
/dev/random	Stream of pseudo-random bytes (from environmental noise)	Adds noise to entropy pool

Devices in Linux

- Device files start with certain letters to show what type of device they are
  - lp – printer
  - sd – disk
  - pp – parallel port
  - tty – terminal
  - pty – pseudo-terminal (pipes between processes)
  - and many more…