DM510: I/O Systems and Networks

Lars Rohwedder

Disclaimer

These slides contain (modified) content and media from the official Operating System Concepts slides: https://www.os-book.com/OS10/slide-dir/index.html

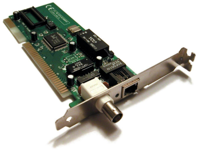
Today's lecture

- Chapter 12 of course book
- Chapter 19 of course book

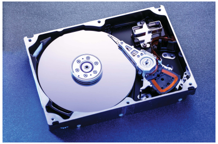
I/O Systems

Devices

- Huge number of different types of devices, for example:



network card



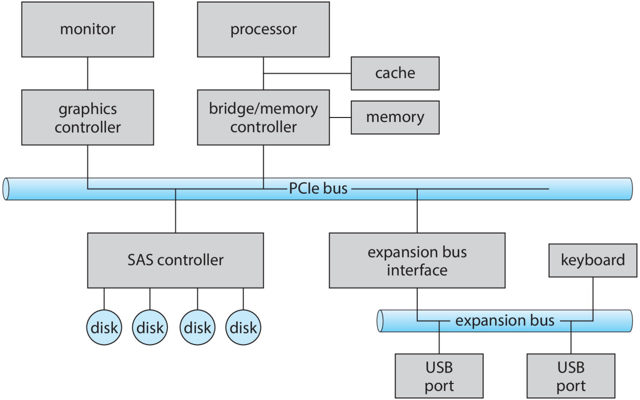
hard disk

GPU

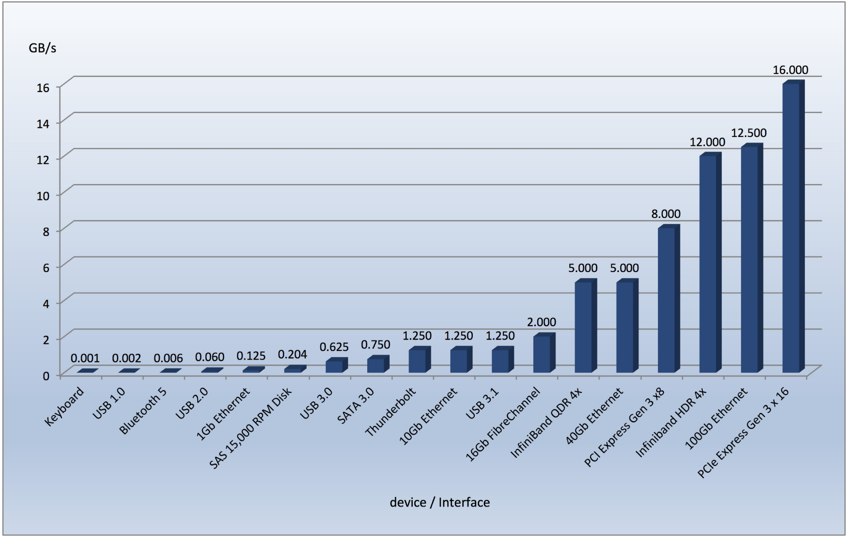
- **Controller:** small processor on device operating a device asynchronously from CPU, usually with local memory

Connections to devices

- **Port:** Connection point, for example: serial port, USB port
- **Bus:** wires connecting devices, often capable of more than two endpoints. For example: PCIe bus



Speed of busses



Low-level I/O interface

Memory-mapped I/O: Devices have registers, which are linked to CPU's address space. CPU writes/reads to them as it would to main memory. Typical registers:

- **data-in register** to read from device
- **data-out register** to write to device
- **status register** e.g. completion of task, data availabile to read, error
- **control register** to start a command

Example: host

Host

- while (control.command-ready = 1) ;
- while (status.busy = 1) ; // busy wait
- data-out = produce()
- control.command-ready = 1

Example: controller

- while (control.command-ready = 0) ;
  // busy wait
- status.busy = 1
- control.command-ready = 0
- consume(data-out)
- status.error = 0
- status.busy = 0

Low-level I/O interface

Memory-mapped I/O: Devices have registers, which are linked to CPU's address space. CPU writes/reads to them as it would to main memory. Typical registers:

- **data-in register** to read from device
- **data-out register** to write to device
- **status register** e.g. completion of task, data availabile to read, error
- **control register** to start a command

Details

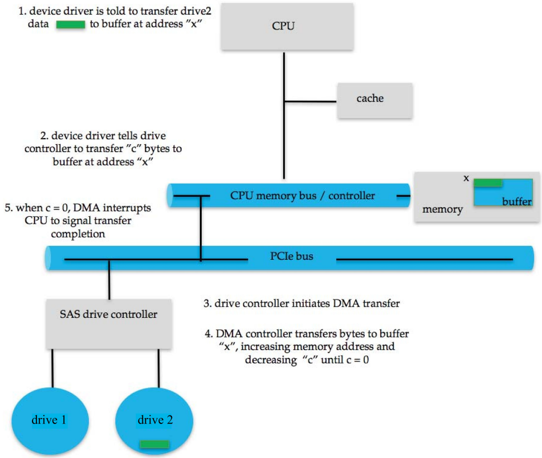
- Some devices support FIFO buffers for several commands at a time
- Notification to CPU can be via **interrupts**
- Low level device I/O **privileged** (only from kernel mode) for protection

Direct memory access (DMA)

Programmed I/O (CPU copies byte by byte) on large data can be performance hit

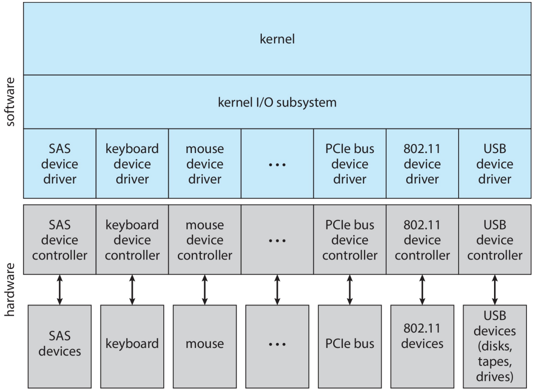
DMA

- CPU sends only command and address of data in main memory, but not data itself
- Controller directly reads and writes from main memory
- Controller sends interrupt when done
- Device may occupy memory bus, delaying CPU-memory communication



Driver

- **Drivers** hide most of the low level details of device communication
- In Unix: drivers expose devices in file system (`/dev`) and with **major/minor numbers** (type of device and instance)



High-level (application) I/O interface

Interfaces provided by drivers usually standardized, with some variations:

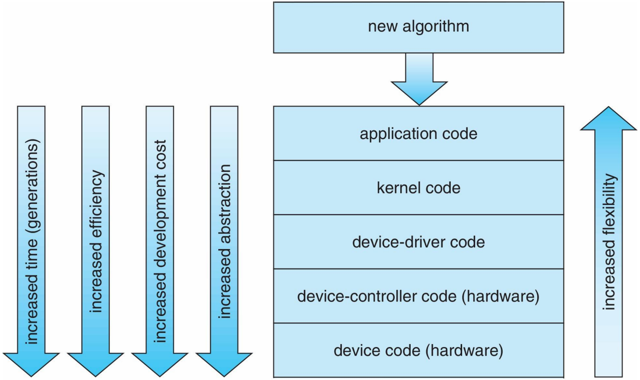
aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

Device access in Linux

- Most devices are exposed as file-like objects in file directory, can be opened, closed, read from, written to as files
- `mmap` can map device to address space for random access (if device capable)
- Socket API for network connections (TCP/UDP)

Abstraction

Often decision has to be made where to implement functionality. Tradeoffs:



Networks

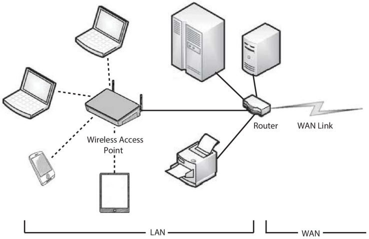
Environment and goals

Goals and challenges

- Allow communication between different hosts (computers or other devices)
- Applications can establish connection to other applications on (usually) different host and use it as reliable bi-directional stream of bytes (see TCP later).
- How does data reach correct LAN, host, application? How to achieve reliability?

Types of networks

- **Local-area network (LAN):** hosts all within small geographical area, e.g. home, office, university
- **Wide-area network (WAN):** hosts span large geographical area, e.g., **Internet / World Wide Web**



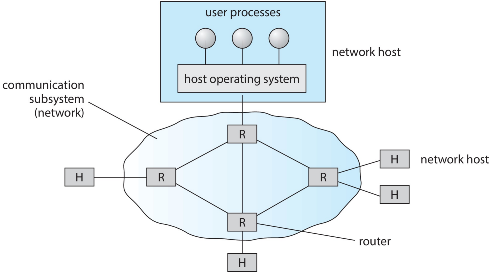
Environment and goals

Goals and challenges

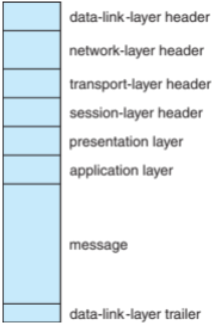
- Allow communication between different hosts (computers or other devices)
- Applications can establish connection to other applications on (usually) different host and use it as reliable bi-directional stream of bytes (see TCP later).
- How does data reach correct LAN, host, application? How to achieve reliability?

Types of networks

- **Local-area network (LAN):** hosts all within small geographical area, e.g. home, office, university
- **Wide-area network (WAN):** hosts span large geographical area, e.g., **Internet / World Wide Web**



Protocol stacks



bits of a packet
and their role

- Typical network communication is implemented as stack of layered protocols that add more and more abstraction, each protocol making use of the next lower protocol
- **Header:** bits of a message reserved for necessary information of each protocol
- **OSI model** is a classical formalization for roles of layers, but not all are used in modern technology, details omitted
- We focus here on the most prominent TCP/IP + Ethernet/WiFi/mobile technology

Application view

Only application layer (e.g. TCP) is exposed to applications, lower layers (e.g., IP layer, Ethernet) almost never directly used

Ethernet (data-link layer)

- Network card sends and receives unreliable bit-stream via physical transmission
- Goal of Ethernet: send variable-size **packets** between hosts in same LAN
- WiFi or mobile networks have similar role to Ethernet

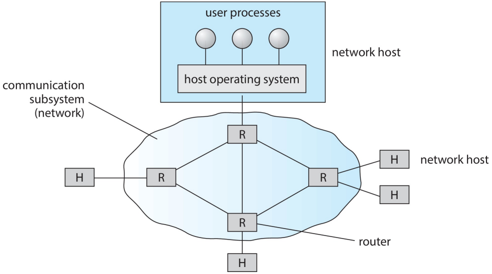
bytes		
7	preamble—start of packet	each byte pattern 10101010
1	start of frame delimiter	pattern 10101011
2 or 6	destination address	Ethernet address or broadcast
2 or 6	source address	Ethernet address
2	length of data section	length in bytes
0–1500	data	message data
0–46	pad (optional)	message must be > 63 bytes long
4	frame checksum	for error detection

- Every Ethernet-capable device has unique **Ethernet/MAC address**
- Packets have **source** and **destination** MAC addresses
- Special **broadcast** address to target all hosts in LAN, e.g. to discover other hosts
- **Checksum** to detect bit errors (discard packet if error detected)

Internet protocol / IP (network-layer)

- Goal of internet protocol: send packages to any host in internet (WAN)

- Each host has unique IP address (slight simplification; due to lack of addresses in IPv4 sometimes protocol abused)
- Routers (devices connecting different LANs) forward packages to correct direction using routing tables

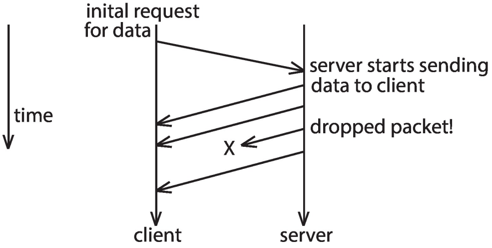


Domain name system (DNS)

- Instead of IP addresses we are often given **domain names**, e.g. sdu.dk, wikipedia.org
- **Name servers** are hosts that translate domain names into IP addresses

User datagram protocol / UDP (transport layer)

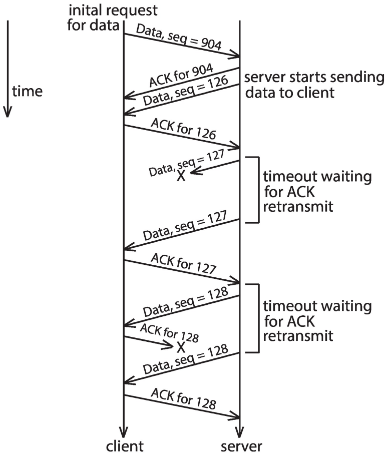
- Goal of UDP: Bring packets to correct application
- Each UDP packet contains a source and a destination port
- Applications can listen to specific ports to receive packets sent to this port (at most one application per port)
- Connectionless protocol, but source port can be used for response
- No guarantee or acknowledgement that packets arrive, no guarantee that they arrive in order



Internet protocol / TCP (transport layer)

- Goal of TCP: establish connection of reliable byte stream between applications on (usually) different hosts
- Used for vast majority of applications (much more than UDP)

- Similar to UDP, TCP packets contain a source and a destination port
- Applications can listen to specific ports to accept connections
- 3-way **handshake** to establish connection
- **Sequence numbers** and **acknoledgements** (for sequence numbers), possible retransmissions (if acknoledgement missing) to ensure in-order reliable stream



Other protocols

- Inside TCP, other protocols like HTTP (access to websites), FTP (file transmission), SMTP (email), SSH (remote terminal) etc. are used
- Security layer for end-to-end encryption may be used (e.g. in SSH or HTTPs)