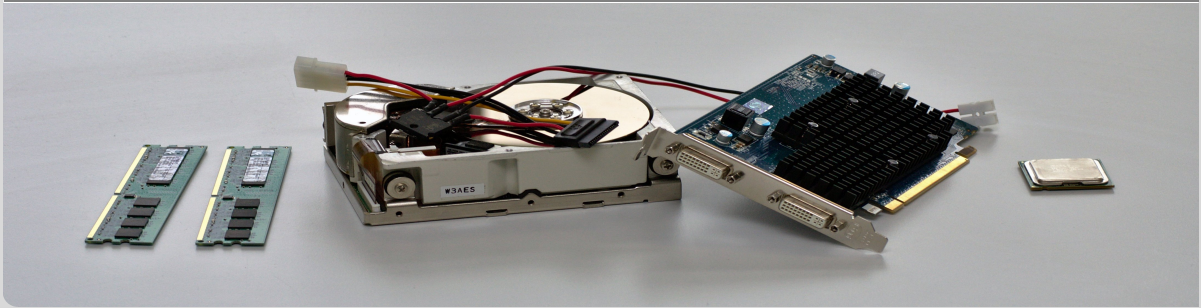
Karlsruhe Institute of Technology

Operating Systems

15. Device Managment

Prof. Dr. Frank Bellosa | WT 2020/2021



Device Management Objectives

- Abstraction from details of physical devices
- Uniform Naming that does not depend on HW details
- Serialization of I/O-operations by concurrent applications
- Protection of standard-devices against unauthorizes accesses
- Buffering, if data from/to a device cannot be stored in the final destination
- Error Handling of sporadic device errors
- Virtualizing physical devices via memory and time multiplexing (e.g. pty, RAM disk)

Device Management Objectives

- Abstraction from details of physical devices
- Uniform Naming that does not depend on HW details
- Serialization of I/O-operations by concurrent applications
- Protection of standard-devices against unauthorizes accesses
- Buffering, if data from/to a device cannot be stored in the final destination
- Error Handling of sporadic device errors
- Virtualizing physical devices via memory and time multiplexing (e.g. pty, RAM disk)

Device Management Objectives

- Abstraction from details of physical devices
- Uniform Naming that does not depend on HW details
- Serialization of I/O-operations by concurrent applications
- Protection of standard-devices against unauthorizes accesses
- Buffering, if data from/to a device cannot be stored in the final destination
- Error Handling of sporadic device errors
- Virtualizing physical devices via memory and time multiplexing (e.g. pty, RAM disk)

Device Management Objectives

- Abstraction from details of physical devices
- Uniform Naming that does not depend on HW details
- Serialization of I/O-operations by concurrent applications
- Protection of standard-devices against unauthorizes accesses
- Buffering, if data from/to a device cannot be stored in the final destination
- Error Handling of sporadic device errors
- Virtualizing physical devices via memory and time multiplexing (e.g. pty, RAM disk)

Device Management Objectives

- Abstraction from details of physical devices
- Uniform Naming that does not depend on HW details
- Serialization of I/O-operations by concurrent applications
- Protection of standard-devices against unauthorizes accesses
- Buffering, if data from/to a device cannot be stored in the final destination
- Error Handling of sporadic device errors
- Virtualizing physical devices via memory and time multiplexing (e.g. pty, RAM disk)

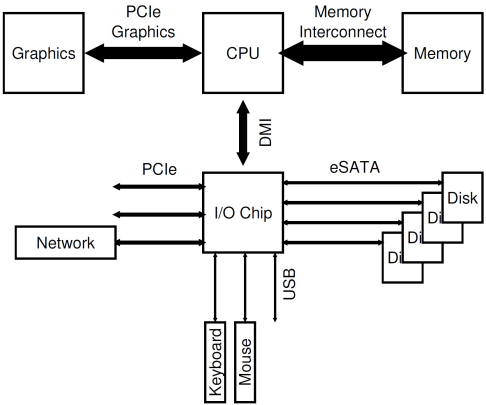
Device Management Objectives

- Abstraction from details of physical devices
- Uniform Naming that does not depend on HW details
- Serialization of I/O-operations by concurrent applications
- Protection of standard-devices against unauthorizes accesses
- Buffering, if data from/to a device cannot be stored in the final destination
- Error Handling of sporadic device errors
- Virtualizing physical devices via memory and time multiplexing (e.g. pty, RAM disk)

Device Management Objectives

- Abstraction from details of physical devices
- Uniform Naming that does not depend on HW details
- Serialization of I/O-operations by concurrent applications
- Protection of standard-devices against unauthorizes accesses
- Buffering, if data from/to a device cannot be stored in the final destination
- Error Handling of sporadic device errors
- Virtualizing physical devices via memory and time multiplexing (e.g. pty, RAM disk)

A Modern System Architecture



[ADAD18]

Characteristics of I/O Devices (1)

- **Block devices** include disk drives
  - Commands include `read`, `write`, `seek`
  - Raw I/O or file-system access
  - Memory-mapped file access possible
- **Character devices** include keyboards, mice, serial ports
  - Commands include `get, put`
  - Libraries layered on top allow line editing
- **Network devices** Specific communication patterns and addressing schemes require own interface
  - Usually addressed with socket interface
    - Separates network protocol from network operation

Characteristics of I/O Devices (1)

- **Block devices** include disk drives
  - Commands include `read`, `write`, `seek`
  - Raw I/O or file-system access
  - Memory-mapped file access possible
- **Character devices** include keyboards, mice, serial ports
  - Commands include `get, put`
  - Libraries layered on top allow line editing
- **Network devices** Specific communication patterns and addressing schemes require own interface
  - Usually addressed with socket interface
    - Separates network protocol from network operation

Characteristics of I/O Devices (1)

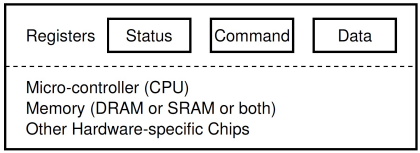
- **Block devices** include disk drives
  - Commands include `read`, `write`, `seek`
  - Raw I/O or file-system access
  - Memory-mapped file access possible
- **Character devices** include keyboards, mice, serial ports
  - Commands include `get, put`
  - Libraries layered on top allow line editing
- **Network devices** Specific communication patterns and addressing schemes require own interface
  - Usually addressed with socket interface
    - Separates network protocol from network operation

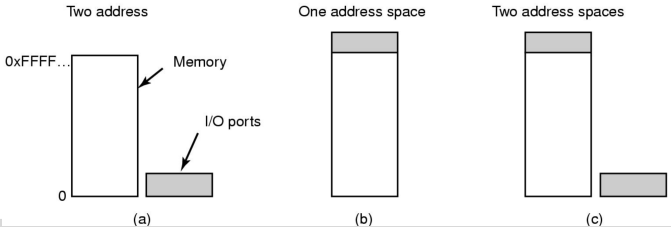
Characteristics of I/O Devices (2)

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	printer optical drive
transfer schedule	synchronous asynchronous	3D printer keyboard
sharing	dedicated sharable	tape network interface
device speed	latency / seek time transfer rate	
I/O direction	read only write only read-write	video capture printer disk

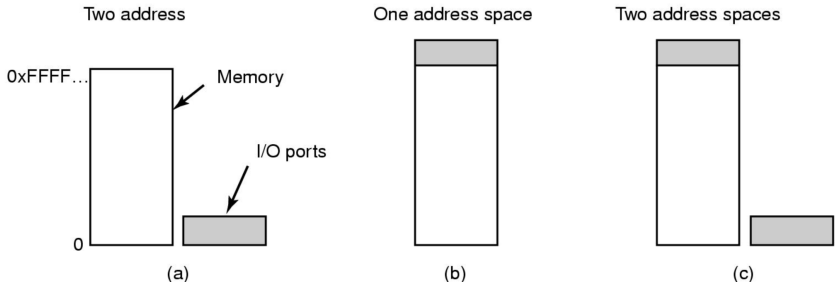
I/O Hardware

- Canonical device interface [ADAD18]
  - Status register
  - Control register
  - Data-in register
  - Data-out register
- Devices are referenced by addresses [TB15]
  - Direct I/O instructions (e.g. to access x86 I/O ports)
  - Memory-mapped I/O





Ports v. Memory-Mapped I/O



- Separate I/O-address space and memory address space
  - `IN R0, 4` `//<port 4> → R0`
  - `MOV R0, 4` `//<4> → R0`
- Memory-mapped I/O
- Hybrid (Intel)

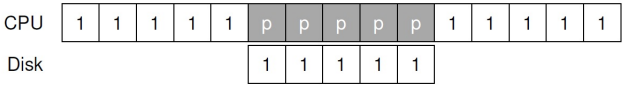
Device I/O Port Locations on PCs (partial)

I/O address range (hexadecimal)	device
$000 - 00F$	DMA controller
$020 - 021$	interrupt controller
$040 - 043$	timer
$200 - 20F$	game controller
$2F8 - 2FF$	serial port (secondary)
$320 - 32F$	hard-disk controller
$378 - 37F$	parallel port
$3D0 - 3DF$	graphics controller
$3F0 - 3F7$	diskette-drive controller
$3F8 - 3FF$	serial port (primary)

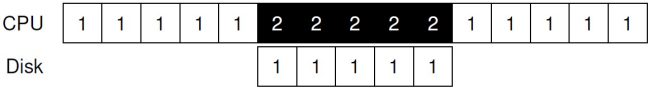
[SGG12]

Polling vs. Interrupt-driven I/O [ADAD18]

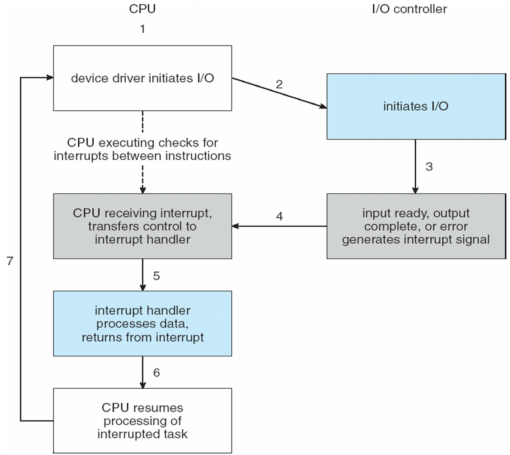
- Programmed I/O
  - Thread is busy-waiting for the I/O-operation
  - Kernel thread is Polling the state of an I/O device
    - device ready
    - busy
    - error



- Interrupt-driven I/O
  - I/O-command is issued
  - processor continues executing instructions
  - I/O-device sends an interrupt when I/O-command is done



Interrupt-Driven I/O Cycle



[SGG12]

Steps for Handling an Interrupt (1)

1. Save registers not already saved by HW-interrupt mechanism
2. Set up context (address space) for interrupt service procedure
   - Typically, handler runs in the context of the currently running process/task
     $\Rightarrow$ no expensive context switch
3. Set up stack for interrupt service procedure
   - Handler usually runs on the kernel stack of the current process/kernel-level thread
   - Handler cannot block, otherwise the unlucky interrupted process/kernel-thread would also be blocked, might lead to starvation or even to a deadlock
4. Acknowledge/mask interrupt controller, thus re-enable other interrupts

Steps for Handling an Interrupt (2)

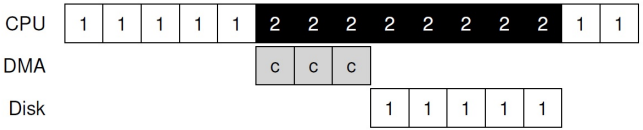
5. Run interrupt service procedure
   - Acknowledge interrupt at device level
   - Figures out what caused the interrupt, e.g.
     - Received a network packet
     - Disk read has properly finished, . . .
   - If needed, it signals the blocked device driver
6. In some cases, we have to wake up a higher priority process/kernel level thread
   - Potentially schedule another process/kernel-level thread
   - Set up MMU context for process to run next
7. Load new/original process' registers
8. Return from interrupt, start running new/original process

Polling vs. Interrupts [ADAD18]

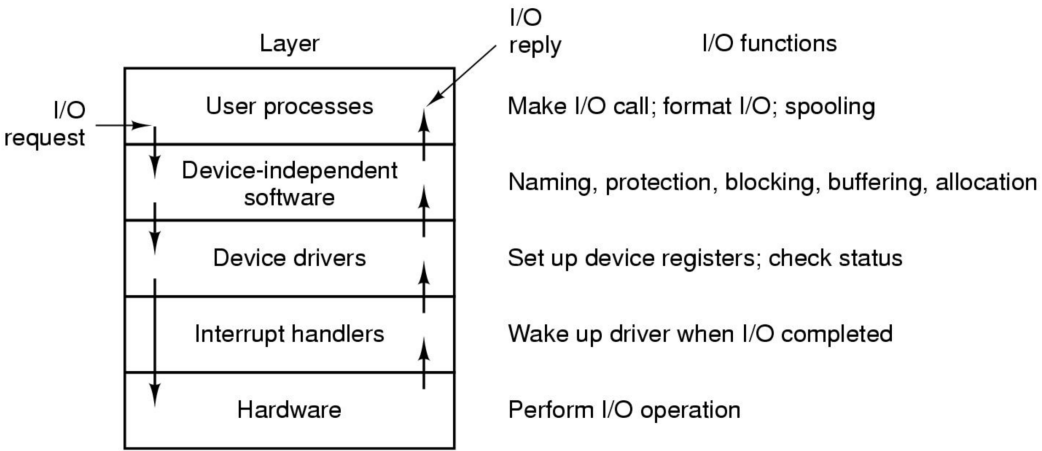
- Fast device: better to spin than take interrupt overhead
- Device time unknown: Hybrid approach (spin then use interrupts)
- Flood of interrupts arrive
  - Can lead to livelock (always handling interrupts)
  - Better to ignore interrupts while make some progress handling them
- Interrupt coalescing (batch together several interrupts)

Direct Memory Access (DMA)

- Direct Memory Access (DMA)
  - DMA module controls exchange of data between main memory and I/O device
  - processor interrupted after entire block has been transferred
  - bypasses CPU to transfer data directly between I/O device and memory



Layered I/O Subsystem

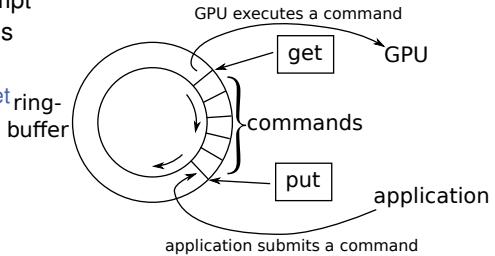


[TB15]

Functions of the Kernel I/O Subsystem (1)

- Device reservation – provides exclusive access to a device
  - System calls for allocation and deallocation

- Protection
  - I/O must be performed via system calls
    - Memory-mapped and I/O port memory locations must be protected
  - User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
  - Example: Nvidia General Purpose GPU
    - Command submission channel: Ring-buffer and put/get device registers are memory-mapped
    - Mapping can be exposed to application
      → Application can submit commands in user-mode



Functions of the Kernel I/O Subsystem (2)

- Spooling
  - Hold output for a device, if device can serve only one request at a time (i.e. printing)
- Uniform kernel interface for device code
  - Drivers use a defined interface to kernel service, e.g. `kmalloc`, install IRQ handler, etc.
  - Allows kernel to evolve without breaking device drivers

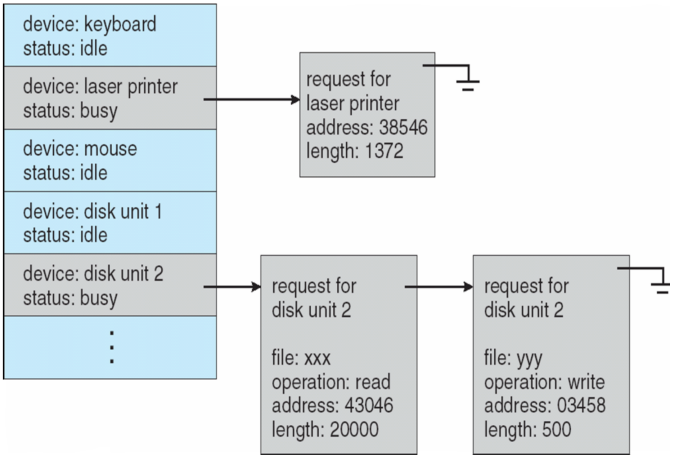
Functions of the Kernel I/O Subsystem (3)

- Request Scheduling
  - Some I/O request ordering via per-device queue
  - Some OSs try fairness
- Buffering – store data in memory while transferring between devices
  - To cope with device speed mismatch
  - To cope with device transfer size mismatch
  - To maintain “copy semantics”
- Error handling and reporting
  - OS can recover from errors e.g., disk read, device unavailable, transient write failures
  - Error reporting to upper levels, i.e. all errors the driver cannot resolve

Device Driver

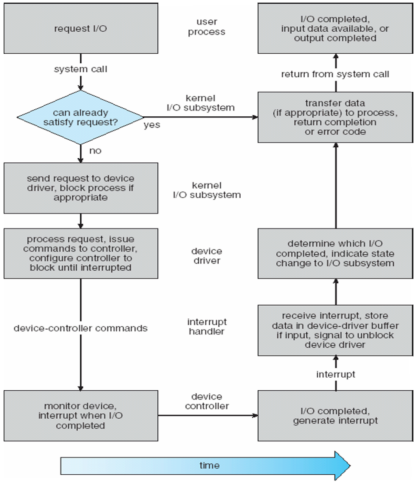
- After issue the command to the device, device either
  - completes immediately and the driver simply returns to the caller or it
  - processes request and the driver usually blocks waiting for an I/O (complete) interrupt signal
- Drivers are reentrant as they can be called by another process while a process is already blocked in the driver
  - Reentrant: code that can be executed by more than one thread (or CPU) at the same time
    - Manages concurrency using sync primitives

Device-status Table



[SGG12]

Life Cycle of an I/O Request



References I

[ADAD18] R.H. Arpaci-Dusseau and A.C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. 2018.

retrieved from `http://pages.cs.wisc.edu/~remzi/OSTEP/`.

[SGG12] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts*. Wiley Publishing, 9th edition, 2012.

[TB15] Andrew S Tanenbaum and Herbert Bos. *Modern operating systems*. Pearson, 4th edition, 2015.