# Operating Systems Design (CS 423)

Elsa L Gunter
2112 SC, UIUC
http://www.cs.illinois.edu/class/cs423/

Based on slides by Roy Campbell, Sam King, and Andrew S Tanenbaum

## Devices and Drivers: Overview

- Basic I/O hardware
  - ports, buses, devices and controllers
- I/O Software
  - Interrupt Handlers, Device Driver, Device-Independent Software, User-Space I/O Software
- Driver abstractions
  - Char devices, block devices

## Devices

- Devices
  - Storage devices (disk, tapes)
  - Transmission devices (network card, modem)
  - Human interface devices (screen, keyboard, mouse)
  - Specialized devices (A/D converter)

## Device-Computer / Device-Device Communication

- Physically: via signals over cable or through air
- Multiple devices are connected via a bus
  - Bus: common set of wires and rigidly defined protocol that specifies set of messages that can be sent on the wires

## Device I/O Controller

- I/O units typically consist of
  - Mechanical component
    - device itself
  - Electronic component
    - device controller / adapter, e.g., circuit board

## Example: Disk controller

- Implements disk side of protocol that does:
  - Bad error mapping, prefetching, buffering, caching
  - Converts the serial bitstream, coming off the drive into a block of bytes,
  - Performs error correction
  - Assembles block of bytes in a buffer
  - Verifies checksum
  - Copies error-free block to main memory

## Controller components

- Registers for data and control
- Communication protocol between CPU and controllers via I/O instructions and registers

## I/O Ports

- Typically at least 4 registers: status, control, data-in, data-out
  - **Status:** states whether the current command is completed, byte is available, device has an error, etc
  - **Control:** host determines to start a command or change the mode of a device
  - **Data-in:** host reads to get input
  - **Data-out:** host writes to send output
- Other possibilities as well, depends on device

## Interrupt-Driven I/O (Host-controller interface)

- CPU hardware has interrupt report line that CPU checks after executing each instruction
- Device raises an interrupt
- CPU catches interrupt and saves the state (e.g.,Instruction pointer)
- CPU dispatches the interrupt handler
- Interrupt handler determines cause, services device and clears interrupt

## Support for Interrupts

- Need ability to defer interrupt handling during critical processing
- CPUs have two interrupt request lines
- Non-maskable interrupt (reserved for unrecoverable memory errors)
- Maskable interrupt (can be turned off by cpu before execution of critical instructions

## Support for Interrupts

- Need efficient way to dispatch proper handler
- Interrupt comes with address (offset in interrupt vector) that selects specific interrupt handling
- Need interrupt handler
  - At boot time, OS probes hardware buses to determine what devices are present and installs corresponding interrupt handlers into interrupt vector

## Programmed I/O (PIO)

- Use CPU to watch status bits and feed data into a controller register 1 byte at a time
- EXPENSIVE for large transfers
- Every IO operation programmed and controlled
- Example: printing a file
- Essential aspect of programmed IO: **polling**
- Programmed IO is simple but requires CPU full time until all IO is done

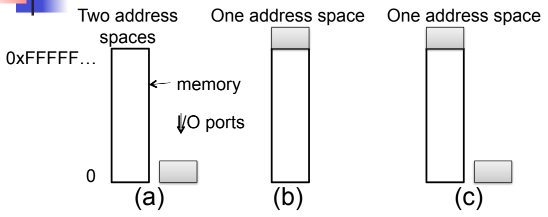
## Direct Memory Access (DMA)

- Direct memory access (DMA)
- Use special purpose processor, called DMA controller
- Traditional model: DMA controller separate hardware on bus between device, cpu, memory
- Modern reality: DMA controller embedded in device
  - Sometimes call a bus master

## DMA Issues

- Software must relinquish allocated memory
  - Physically contiguous - no breaks across non-contiguous physical pages
  - Typically set aside special memory region for I/O
- Cycle stealing
  - DMA controller takes away CPU cycles when it uses CPU memory bus, hence blocks CPU from accessing the memory
- In general DMA controller improves the total system performance

## Memory Mapped I/O



- (a) Separate I/O and memory space
- (b) Memory-mapped I/O
- (c) Hybrid

## Tradeoffs

- Interrupt-driven IO:
  - Pro: save CPU time for busy polling
  - Con: triggering interrupt takes time, latency
- Programmed I/O
  - Pro: requires no interrupt or DMA support, low latency, most predictable response time
  - Con: waste CPU time
- I/O using DMA:
  - Pro: relieve CPU from I/O operation
  - Con: more complex interface

## Recent trends

- Currently programmed I/O is uncommon
  - DMA by far most common interface
- Deep pipelines – interrupts expensive
- Multi-core – mostly idle
- Embedded systems – latency sensitive
  - Often prefer predictability over performance
- Are we going to see a resurgence in programmed IO?

## Controller components

- Registers for data and control
- Communication protocol between CPU and controllers via I/O instructions and registers
  - **Status:** states whether the current command is completed, byte is available, device has an error, etc
  - **Control:** host determines to start a command or change the mode of a device
  - **Data-in:** host reads to get input
  - **Data-out:** host writes to send output

## Programming I/O Hardware

- Programmers look at hardware as interface to software
  - Commands hardware accepts
  - Functions it carries out
  - Errors that it reports back

## Programming I/O Hardware

- We consider programming of I/O devices, not building them in this class
- However, IO devices are closely connected with their internal operations hence we need to understand at least some of the basic principles

## Device driver interfaces

Physical reality:

- Single hardware device
- Specific manufacturer and device model
  - Device specific access
- Limited buffer space
- Asynchronous

Illusion:

- Single hardware device
- General type of device
- Generic access interface
- Lots of buffer space
- Synchronous

## Device Classification

- Character-stream or block devices
- Sequential or random access devices
- Synchronous or asynchronous devices
- Sharable or dedicated devices
- Speed of operation
- WO, RO, RW devices

## Block Devices

- Transfer blocks of data (e.g., Disk device)
- Commands: read, write, seek (if random access device)
- Memory-mapped files access can be layered on top of block-device drivers

## Character Devices

- Character device - transfers byte by byte (e.g., Keyboard, sound card, NIC)
- Commands: get, put one character
- Can block device be implemented using character device?

## I/O Software: Principles

- Device independence
- Possible to write programs that can access any Io device without having to specify device in advance
- Read file as input on a floppy, hard disk, or CD-ROM, without modifying program for each device

## I/O Software: Principles

- Uniform naming
  - Name of file or device should simply be a string or an integer and not depend on device in anyway
  - Example: Plan9
    - All drivers are files in file system name space

## I/O Software: Principles

- Device independence
  - Possible to write programs that can access any Io device without having to specify device in advance
  - Read file as input on a floppy, hard disk, or CD-ROM, without modifying program for each device

## I/O Software: Principles

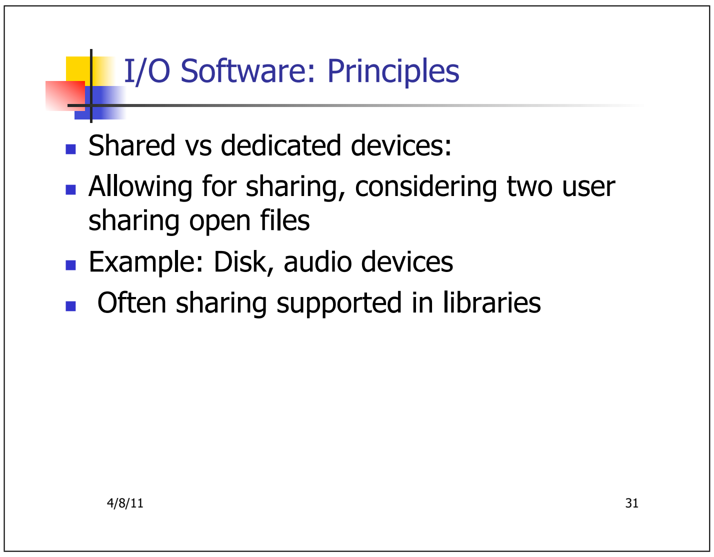
- Uniform naming
  - Name of file or device should simply be a string or an integer and not depend on device in anyway
  - Example: Plan9
  - All drivers are files in file system name space

## I/O Software: Principles

- Synchronous (blocking) vs asynchronous (interrupt-driven) transfers
  - Example: most devices are asynchronous. User programs are easier to write if the IO operations are synchronous. Hence, it is up to OS to make asynchronous operations look like synchronous to user programs

## I/O Software: Principles

- Buffering
- Systems are busy, devices have limited storage
- Keep extra data in software to make device appear to have more space
- Example: Network packets, audio



## I/O Software: Principles

- Shared vs dedicated devices:
- Allowing for sharing, considering two user sharing open files
- Example: Disk, audio devices
- Often sharing supported in libraries