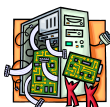


I/O = Input/Output Devices



Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

1

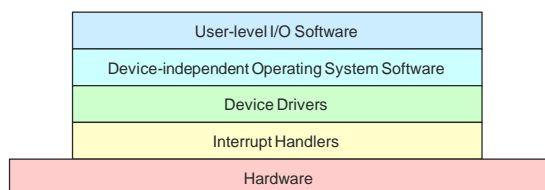
Overview

- Basic I/O hardware
 - ports, buses, devices and controllers
- I/O Software
 - Interrupt Handlers, Device Driver, Device-Independent Software, User-Space I/O Software
- Important concepts
 - Three ways to perform I/O operations
 - Programmed I/O, Interrupt and DMAs

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

2

I/O Software Layers



Layers of the I/O Software System

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

3

Devices

- Storage devices
 - Disk, tapes
- Transmission/Communication devices
 - Network card, modem
- Human interface devices
 - Screen, keyboard, mouse
- Specialized devices
 - Joystick

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

4

[Input/Output Problems]

- Wide variety of peripherals (external devices)
 - Delivering different amounts of data
 - At different speeds
 - In different formats
- All slower than CPU and RAM
 - Need I/O modules

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

5



[I/O Device Characteristics]

- Application usage
 - Disk for storing files or virtual memory pages
- Complexity of control
 - Simple vs. complex
- Data representation
 - Diversity of encoding schemes
- Error conditions
 - Devices respond to errors differently

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

6



[I/O Device Characteristics]

- Unit of transfer
 - Data may be transferred as a stream of bytes for a terminal or in larger blocks for a disk
 - Block devices
 - Disk drives
 - Commands include read, write, seek
 - Raw I/O or file-system access
 - Memory-mapped file access possible
 - Character devices
 - Keyboards, mice, serial ports
 - Commands include get, put
 - Libraries layered on top allow line editing

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

7



[I/O Device Characteristics]

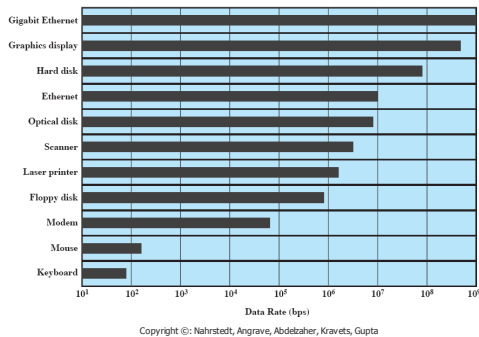
- Data rate
 - May be differences of several orders of magnitude between the data transfer rates

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

8



Typical I/O Device Data Rates



9



Hardware or Software?

- Is the following component software or hardware?
 - Device controller
 - A hardware element
 - Accepts simple device hardware instructions into registers to read and write data
 - Device driver
 - Part of the OS that runs in software on the CPU
 - Makes calls to the device controller

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

10



Device Controller

- I/O units typically consist of
 - **Mechanical** component
 - The device itself
 - **Electronic** component
 - The device controller or adapter
- Interface between controller and device is a very low level interface
- Example: Disk controller
 - Take serial bit streams coming off the drive
 - Convert into a block of bytes
 - Perform error correction

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

11



I/O Controller

- Disk controller
 - The disk side of the protocol
 - Error mapping, prefetching, buffering, caching
- Controller has I/O registers/ports for data and control
- CPU and controllers communicate via
 - I/O instructions and registers
 - Memory-mapped I/O

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

12



I/O Registers/Ports

- 4 registers - 1 to 4 bytes
 - Status
 - 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

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

13



I/O Registers/Ports

- Instructions and Data
 - Format is device-dependent
 - Device driver code needs to be aware of this format
 - Each device from each vendor typically needs a separate device driver

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

14



Memory-Mapped I/O

Key Idea: Each port of each device is assigned a unique main-memory address



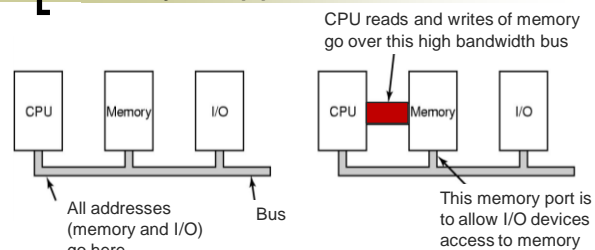
- Separate I/O ports and memory space
 - Requires extra commands to access registers
- Memory-mapped I/O
 - Easy access, protection, memory and registers are treated the same
 - Poor interoperation with caches
- Hybrid
 - Memory-mapped data buffers
 - Separate I/O ports

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

15



Memory-Mapped I/O



- Single-bus architecture
 - All memory modules and all I/O devices must examine all memory references
- Dual-bus memory architecture
 - Memory bus is not accessible by I/O devices

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

16



I/O Functions

- Programmed I/O
 - CPU issues I/O command
 - CPU directly writes instructions into registers
 - CPU busy waits for completion
- Interrupt-driven I/O
 - CPU issues I/O command
 - CPU directly writes instructions into registers
 - CPU continues operation until interrupt
- Direct Memory Access (DMA)
 - CPU asks DMA to perform I/O to memory transfer
 - DMA issues I/O command and transfers new item into memory
 - DMA module is interrupted after completion

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

17



Programmed I/O - Polling

- Programmed I/O sequence:
 1. CPU requests I/O operation
 2. I/O module performs operation
 3. I/O module sets status bits
 4. CPU checks status bits periodically
- I/O module does not inform CPU directly
- I/O module does not interrupt CPU
- CPU may wait or come back later

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

18



Programmed I/O - Polling

- Driver operation to input sequence of chars

```
i = 0;
while (...) {
    write_reg(opcode, read);
    while (busy_flag == true);
    mm_in_area[i] = data_buffer;
    increment i;
    compute;
}
```

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

19



Programmed I/O - Polling

- Expensive for large transfers
 - What devices make large transfers?
 - What devices make small transfers?
- Acceptable only if
 - Small dedicated system
 - Not too few processes
 - Character devices (as opposed to block devices)

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

20



Interrupt-driven I/O

- Approach
 1. CPU issues read command
 2. I/O module gets data from peripheral while CPU does other work
 3. I/O module interrupts CPU
 4. CPU requests data
 5. I/O module transfers data
- Advantage: Overcomes CPU busy waiting loops
 - No repeated CPU checking of device
- I/O module interrupts when ready
 - Event-driven!

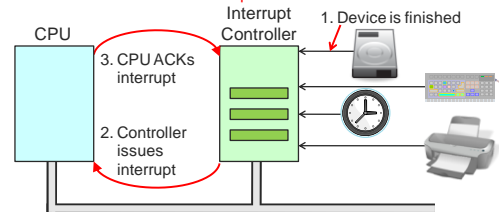
Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

21



Interrupt-driven I/O

Key Idea: Inform device controller of I/O request, go to blocked state, wait for device to finish request



- Connections between devices and interrupt controller use shared interrupt lines on the bus rather than dedicated wires

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

22



Interrupt-driven I/O

- Driver operation to input sequence of chars

```
i = 0;
while (...) {
    write_reg(opcode, read);
    block to wait for interrupt;
    mm_in_area[i] = data_buffer;
    increment i;
    compute;
}
```

was:

```
while (busy_flag == true);
```

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

23



Host-controller interface: Interrupts

- CPU hardware has the interrupt report line that the CPU tests after executing every instruction
 - If a(ny) device raises an interrupt by setting interrupt report line
 - CPU catches the interrupt and saves the state of current running process into PCB
 - CPU dispatches/starts the interrupt handler
 - Interrupt handler determines cause, services the device and clears the interrupt report line
- Real life analogy for interrupts
 - An alarm sets off when the food/laundry is ready
 - So you can do other things in between

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

24



Support for Interrupts

- Need the ability to defer interrupt handling during critical processing
 - Why?
- Need efficient way to dispatch the proper interrupt handler
 - Interrupt comes with an id
 - Interrupt vector maintains addresses of interrupt handler functions (one per device) – an array of function pointers
 - Id is index into vector of device driver functions
- Need multilevel interrupts - interrupt priority level
 - Some interrupts more important than others, e.g., clock more than network

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

25



Interrupt Handler

- Discovery:
 - At boot time, OS probes the hardware buses to
 - Determine what devices are present
 - Install corresponding interrupt handlers into the interrupt vector
- During I/O interrupt
 - Device controller implicitly signals that device is ready for next request

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

26



Other Uses of Interrupts

- Besides I/O devices
- Interrupt mechanisms are used to handle a wide variety of exceptions:
 - Division by zero, wrong address
 - System calls (software interrupts/signals, trap)
 - Multi-threaded systems
 - Examples?
 - Virtual memory paging

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

27



Direct Memory Access (DMA)

- Means what it says!
 - Special hardware element
 - Assists in direct exchange of data between main memory and I/O controller
 - More efficient than CPU requesting data from I/O controller byte by byte, e.g., for block devices like disks
 - DMA controller
- Real life analogy
 - Hire housekeeper to help arrange your stuff

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

28



The DMA-CPU Protocol

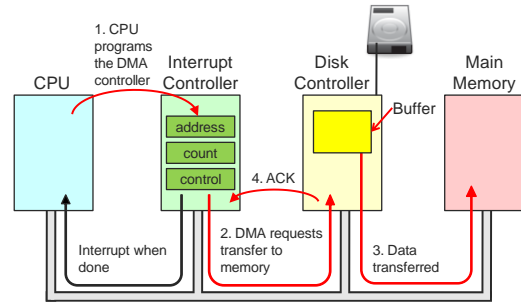
1. CPU programs DMA controller
 - Sets registers to specify source/destination addresses, byte count and control information (e.g., read/write). Then continues with other work.
2. DMA controller proceeds to operate the memory bus directly without help of main CPU
 - Requests I/O controller directly to move data to memory
3. Disk controller transfers data to main memory
4. Disk controller ACKs transfer to DMA controller
5. DMA controller sends interrupt back to CPU

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

29



Direct Memory Access (DMA)



Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

30



DMA

- Driver operation to input sequence of chars

```
write_reg(mm_buf, m);
write_reg(count, n);
write_reg(opcode, read);
block to wait for interrupt;
```

- Writing opcode triggers DMA controller
- DMA controller issues interrupt after n chars in memory

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

31



What's the Catch?

- Handshaking between DMA controller and the device controller
- Cycle stealing
 - Bus is shared by DMA controller and CPU
 - DMA controller takes away CPU cycles when it uses bus, hence blocks CPU from accessing memory
 - Not an interrupt: CPU does not switch context
 - Causes the CPU to execute more slowly
- In general DMA controller improves the total system performance

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

32



Discussion

- Tradeoffs between
 - Programmed I/O
 - Interrupt-driven I/O
 - I/O using DMA
- Which is fastest for a single I/O request that takes a very short time?
- Which is fastest for a single I/O request that takes a very long time?
- Which one gives the highest throughput?

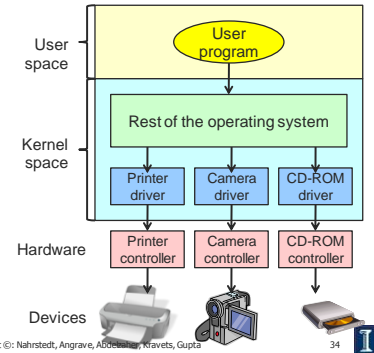
Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

33



Device Drivers

- Logical position of device drivers
- Communications between device driver and device controllers goes over the bus



Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

34



Device Drivers

- Device-specific code to control an IO device
 - Typically written by device's manufacturer
 - Controller has some device registers used to give it commands.
 - Number of device registers and the nature of commands vary from device to device
 - Mouse driver accepts information from the mouse about how far it has moved
 - Disk driver has to know about sectors, tracks, heads, etc).

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

35



Device Drivers

- Typically part of the OS kernel
 - Compiled with the OS
 - Dynamically loaded into the OS during execution
- Each device driver handles
 - One device type
 - Mouse, Disk, etc.
 - One class of closely related devices
 - SCSI disk driver to handle multiple disks of different sizes and different speeds
- Categories
 - Block devices
 - Character devices

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

36



Functions of Device Drivers

- Accept abstract read and write requests from the device-independent layer above;
- Initialize the device
- Manage power requirements and log events
- Check input parameters if they are valid
- Translate valid input from abstract to concrete terms
 - e.g., convert linear block number into the head, track, sector and cylinder number for disk access
- Check the device if it is in use (i.e., check the status bit)
- Control the device by issuing a sequence of commands. The driver determines what commands will be issued.

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

37



Goals of I/O Software

- Device independence
 - Programs can access any I/O device
 - Without specifying device in advance
 - (floppy, hard drive, or CD-ROM)
- Uniform naming
 - Name of a file or device a string or an integer
 - Not depending on which machine
- Error handling
 - Handle as close to the hardware as possible

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

38



Goals of I/O Software

- Synchronous vs. asynchronous transfers
 - Blocked transfers vs. interrupt-driven
- Buffering
 - Data coming off a device cannot be stored in final destination
- Sharable vs. dedicated devices
 - Disks are sharable
 - Tape drives would not be

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

39



Device-Independent I/O Software

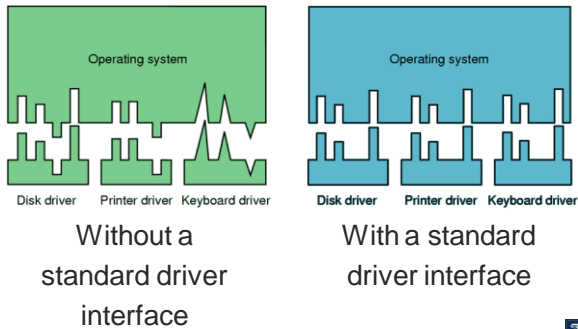
- Goals
 - Uniform interfacing for device drivers
 - Buffering
 - Error reporting
 - Allocating and releasing dedicate devices
 - Providing a device-independent block size

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

40



Device-Independent I/O Software



Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

41



Buffer and Cache

- Buffer
 - Memory area that stores data while it is being transferred between two devices or between a device and an application.
- Motivation
 - Adapt between devices with different data-transfer sizes
 - Adapt between devices with different data-transfer speeds
 - Support of copy semantics for application I/O - application writes to an application buffer and the OS copies it to the kernel buffer and then writes it to the disk
 - Buffering may be needed both ways

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

42



Buffer and Cache

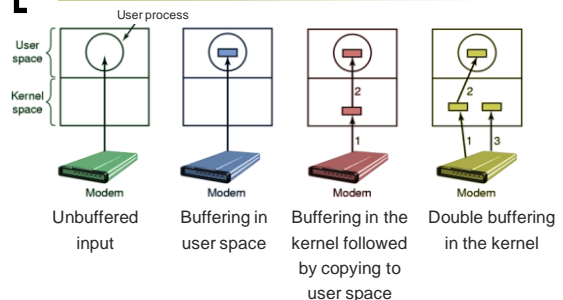
- Caching
 - Cache is region of fast memory that holds copies of recently-accessed data
 - Improves performance when there is locality of interest: processes accessing same data items repeatedly
 - E.g., CPU cache (located near CPU), Disk cache (located in disk)

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

43



Buffering strategies



Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

44



Error Reporting

- Programming I/O Errors
 - occur when a process asks for something impossible
 - e.g., write to an input device such as keyboard, or read from output device such as printer
- Actual I/O Errors
 - occur at the device level
 - e.g., read disk block that has been damaged, or try to read from video camera which is switched off
- The device-independent I/O software detects these errors and responds to them by reporting to the process

