

CSE 120

Summer, 2005

Day 6
Input/Output

Instructor: Neil Rhodes

Overview

Hardware

- How hardware works

Operating Systems Layer

- What the Kernel does

API

- What the programmer does

2

Hardware

Kinds

- Block devices: read/write a block independent of all others
 - Disk drive
 - Floppy drive
 - USB pendrive
- Character devices: read/write one or more bytes
 - Keyboard
 - Mouse
 - Serial
 - Network
 - Printer
- Miscellaneous
 - Memory-mapped video
 - Video chips
 - Clock

3

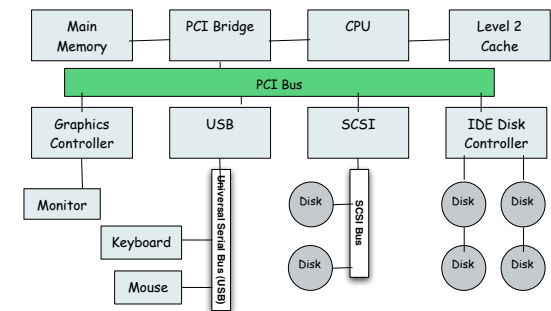
Hardware

Bus: set of wires and protocol for communicating

- ISA
- PCI
- Memory bus

Daisy chain: bus with cables going from one device to the next

- USB
- SCSI
- Firewire



4

Hardware Controllers

Simple

- Serial controller:
 - Has buffer of several bytes

Complex

- SCSI controller:
 - reads data from drive serially into block of memory inside controller
 - Returns that block of data once block is completely read

Communication between CPU and controller

- Registers:
 - Data in
 - Data out
 - Status
 - Control
- How to access registers?
 - *Memory-mapped I/O*
 - Registers in a portion of memory space. Device controller looks for memory in its own area of space
 - *Using I/O ports*
 - Separate instructions for reading and writing port address space. (Separate bit on bus specifies I/O space rather than memory space).

5

Communicating with a Hardware Controller

Polling write (busy-wait):

- **CPU:**
 - loop while busy bit in status register is set
 - Set write bit in control register. Write data byte into data out register.
 - Set command-ready bit in control register
 - loop while busy bit in status register is set

Controller:

loop while command-ready bit in control register is not set

Since write bit in control register set, read data-out register

do I/O

Clear command-ready bit in control register
Clear error bit in status register
Clear busy bit in status register

- Continue processing

6

Communicating with a Hardware Controller

Interrupt-driven I/O

- **CPU:**
 - loop while busy bit in status register is set
 - Set write bit in control register. Write data byte into data out register.
 - Set command-ready bit in control register
 - Continue processing

Controller:

loop while command-read bit in control register is not set

Since write bit in control register set, read data-out register

do I/O

Clear command-ready bit in control register
Clear error bit in status register
Clear busy bit in status register
Issue interrupt

- Process interrupt

7

Interrupts

Interrupt Levels

- Specifies priority of an interrupt: higher level interrupts are serviced before lower-level ones
- Non-Maskable Interrupt (NMI)
 - One that the processor can't ignore. Debugging button, for example

Interrupt Controller will:

- Be notified by hardware controller that it wishes to interrupt
- Check to see whether there's an already-pending interrupt of same or higher priority
- If not, put interrupt address (small number) on bus, and generate interrupt signal to CPU

CPU will:

- Check for interrupt signal after every instruction
- On interrupt (if not masked), save minimal state
 - Use *interrupt vector* (table at well-known location) to jump to *interrupt service routine* (ISR)
 - ISR will acknowledge the interrupt when it is ready to handle another interrupt
 - Handle condition that caused interrupt (wake sleeping process)
 - Restore state

8

Direct Memory Access (DMA)

Without DMA

- CPU must transfer information from device controller to main memory by reading/writing data one byte/word at a time (*programmed I/O*)

Idea: Have DMA controller that will do the transfer on behalf of the CPU

- Sometimes built into device controller (must be a *bus master*)
- CPU programs the DMA controller to specify source, destination and amount
- DMA controller loops:
 - Seize memory bus (*cycle stealing*)
 - Tell device controller to write to memory address
 - Release memory bus
- When complete, issue interrupt to CPU

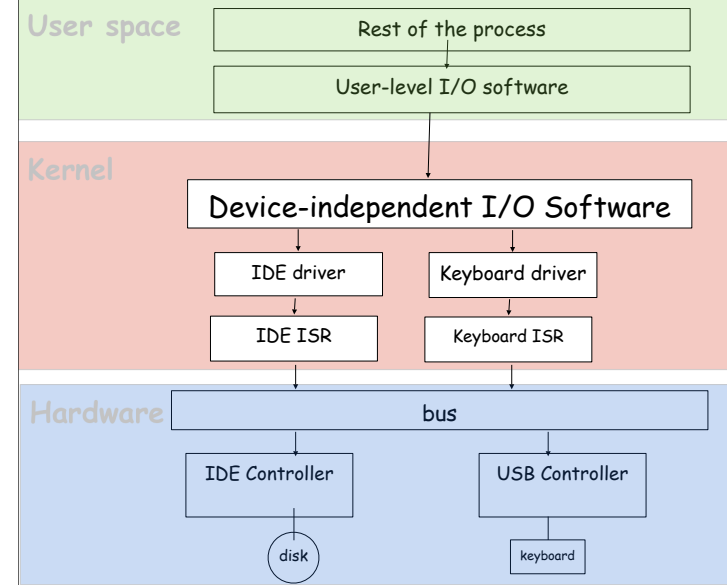
Burst mode: do a series of transfers while holding the bus

Usually used for device generating lots of data

- Disk, video
- Not keyboard, for example

9

Layers of I/O Software



Device Driver Functionality

Initialize

- Probe whether device is there
- Figure out interrupt address (usually known by device controller)
- Allocate data structures

Read

- Read data from device

Write

- Write data to device

ioctl

- Device-specific functionality
 - Set baud rate, for example

ISR

- Not called directly, but only in response to an interrupt

11

Uniformity

Uniform interface between OS and drivers

- Unix character devices:
 - read
 - write
 - open
 - close
 - ioctl
- Unix block devices (buffered in kernel memory)
 - open
 - close
 - strategy
 - Given a buffer header with: address, read/write bit, block number, word count, major/minor device number

Uniform interface between user programs and devices

- Unix, for example
 - inode specifies block vs. character and major/minor device number
 - Major number used as index into table of drivers
 - Minor number used to specify which device
 - which partition on a disk
 - which serial port

12

Buffering

Unbuffered

- Driver reads directly into user space, one byte at a time
 - `read(fd, &ch, 1);`
- Disadvantage: must wake up user process on every byte

Buffered in user space

- Driver reads directly into a buffer in user space
 - `read(fd, buffer, sizeof(buffer));`
- Disadvantage: must lock the page in memory

Buffered in kernel

- Driver reads into a buffer in the kernel
- When buffer is full, copied to user space
- Disadvantage: copying takes time. Buffer may overflow

Double-buffering in kernel

- Driver reads into a buffer in the kernel
- When buffer is full, copied to user space
- While copying (or waiting for user space page to be paged in), use separate buffer for incoming data
- Disadvantage: more memory used in the kernel

13

Buffering and Performance

Write a network packet

- Assemble packet in user space
- Write:
 - Copies to kernel buffer
- Driver:
 - copies to controller buffer
- Controller
 - moves onto network
- On remote end:
 - controller assembles into controller buffer
- Driver
 - Copies from controller buffer to kernel buffer
- Read
 - Copies from kernel buffer to user space

14

User-space I/O software

Buffering in user-space

- all FILE * routines: `fputc`, `putchar`, `getchar`

Formatting routines:

- `sprintf`, `printf`, etc.

Library interfaces to system calls

- `write`
 - Small assembly-language stub that marshals parameters and issue system call
- `read`
- etc.

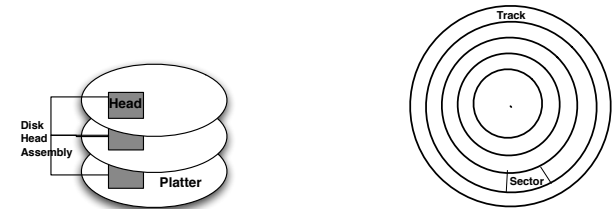
Spooling

- printing a file
- UUCP (Unix to Unix Copy)

15

Disks

Hardware



Cylinder: set of same tracks on multiple platters

Time to access a block (sector)

- Seek time (time to move the head in or out to the appropriate track)
- Rotational latency (time for the disk to spin so that the beginning of the sector is under the head)
- Transfer time (time for the data to be read from the sector).

16

Disk Specs

	Western Digital Raptor WD740GD	Hitachi Deskstar 7250
Capacity	72 GB	250 MB
Rotational speed	10000 RPM	7200 RPM
Average rotational latency	3 ms	4 ms
Average seek time	5 ms	8.5 ms
Average sustained transfer rate	72 MB/s	40 MB/s
Buffer size	8 MB	8 MB

17

RAID

Redundant Array of Independent Disks (compared to SLED: Single Large Expensive Disk)

- Bunch of disks controlled by a RAID card
- Looks like SLED to operating system, but provides better performance and better reliability

Level 0

- Striping
- Parallel reads (for large read requests)
- Worse reliability

	Disk 1	Disk 2	Disk 3
Block 1-k	LBN 1-k	LBN k-2k	LBN 2k-3k
Block k-2k	LBN 3k-4k	LBN 4k-5k	LBN 5k-6k
Block 2k-3k	LBN 6k-7k	LBN 7k-8k	LBN 8k-9k

Level 1

- Every strip written twice
- Read from one disk while busy with another

	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5	Disk 6
Block 1-k	LBN 1-k	LBN k-2k	LBN 2k-3k	LBN 1-k	LBN k-2k	LBN 2k-3k
Block k-2k	LBN 3k	LBN 4k	LBN 5k-6k	LBN 3k-4k	LBN 4k-5k	LBN 5k-6k
Block 2k	LBN 6k	LBN 7k	LBN 8k-9k	LBN 6k-7k	LBN 7k-8k	LBN 8k-9k

18

RAID

Level 2

- Break each word into bits
- Add Hamming code
 - can correct any 1-bit error
 - can detect any 2-bit error
- Can lose any one drive
- Disks must be synchronized (rotational position and head location)
- Throughput increased by 16x

	Disk 1	Disk 2	Disk 3	...	Disk 21
Bit 1	Err bit	Err bit	Bit 1	...	Bit 16
Bit 2	Err bit	Err bit	Bit 1	...	Bit 16
Bit 3	Err bit	Err bit	Bit 1	...	Bit 16

Level 3

- One extra parity drive
- Similar to level 2
- Can't correct silent errors

	Disk 1	Disk 2	...	Disk	Disk 17
Bit 1	bit 1	bit 1	...	Bit 16	parity
Bit 2	bit 1	bit 1	...	Bit 16	parity
Bit 3	bit 1	bit 1	...	Bit 16	parity

19

RAID

Level 4

- Like level 0, but with extra parity drive
- On write, must re-read all strips to recalculate parity
 - Or, can pre-read old parity and data to compute new parity
- Parity drive may become bottleneck (used on every write)

	Disk 1	Disk 2	Disk 3	Disk 4
Block 1-k	LBN 1-k	LBN k-2k	LBN 2k-3k	Parity 1-3k
Block k-2k	LBN 3k-4k	LBN 4k-5k	LBN 5k-6k	Parity 3k-6k
Block 2k-3k	LBN 6k-7k	LBN 7k-8k	LBN 8k-9k	Parity 6k-9k

Level 5

- Like level 4, but with parity strip spread across drives

	Disk 1	Disk 2	Disk 3	Disk 4
Block 1-k	LBN 1-k	LBN k-2k	LBN 2k-3k	Parity 1-3k
Block k-2k	LBN 3k-4k	LBN 4k-5k	Parity 3k-6k	LBN 5k-6k
Block 2k-3k	LBN 6k-7k	Parity 6k-9k	LBN 7k-8k	LBN 8k-9k

20

Disk-arm Scheduling Algorithms

Reducing seek time will increase system performance

First Come First Serve (FCFS)

- Can require lots of seeking (imagine queue containing cylinder 1, cylinder n, cylinder 2, cylinder n-5)

Shortest Seek First (SSF)

- From among the cylinders in the queue, go to the one that's nearest
- Can increase throughput
- Disadvantage:
 - Imagine queue contains request for block in cylinder 1, but other requests keep coming in that are closer to the disk arm. A request can be indefinitely delayed.

Elevator (or SCAN)

- Keep state of moving up or down. Move the disk arm to next-closest requested *higher cylinder* (if moving up). If at end, switch direction and start moving down.

C-SCAN

- Always go up. When at end, go to other end.

21

Anticipatory Scheduling

Scenario:

- Imagine two process p and q each writing to 10 MB disk files. If running alone, each takes 5 seconds.
- How long to run together?
 - p and q will issue requests
 - If p is satisfied first, scheduling algorithm will try to satisfy q's request.
 - When p runs, it'll issue another write request
 - By that time, the disk head has moved way off.

You can be faster by being idle!

Solution:

- Add to any disk-scheduling algorithm *anticipation*.
- Simple explanation:
 - The last process that issued a request will probably issue another one soon (in same general location)
 - If request we'd otherwise do is not close, instead of moving the disk head, delay and do nothing for a short time (500 milliseconds?).
 - If request comes in from process during that time, stop waiting
 - and, presumably, read/write that requested disk block
 - If timer expires, continue with disk scheduling algorithm

Benefits

- Decreased latency, Increased bandwidth

22

Anticipatory Scheduling

Reference

- Sitaram Ayer, 2001, <<http://www.cs.rice.edu/~ssiyer/r/antsched/>>

Implementation: Linux

- Streaming read while streaming write taking place
 - Without anticipatory: 42, 48, 47 seconds
 - With anticipatory: 3.8 seconds
- Reading many small files while streaming write taking place
 - Without anticipatory: >15:00, 7:27, 9:55 minutes:seconds
 - With anticipatory: 17 seconds

More formally

- benefit = CalculateSeekTime(candidate) - Expected seek time of process
- cost = max(0, expected median think time of process - elapsed time)
- duration = max(0, expected 95th percentile think time of process - elapsed time)
- if (benefit > cost) time_to_wait = duration
else don't wait

23

Bad Blocks

Low-level format (at factory) reserves blocks

- Not all at beginning or end, but spread out across disk

Controller will detect bad blocks and either:

- Spare a block (for example, if block 17 is bad, remap it to block 120)
- Slip a contiguous section of blocks
 - If block 17 is bad and first reserved block is 120, remap blocks 17-119 to 18-120 (and move blocks 18-119 up one).

Drive controller can report information on bad blocks, etc. to OS

- SMART

24