I/O Systems

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

Motivation

Overview

- I/O management is a major component of OS design and operation.
  - Important aspect of computer operation
  - I/O devices vary greatly
  - Various methods to control them
  - Performance management
  - New types of devices frequent

Overview

- I/O management is a major component of OS design and operation.
    - Important aspect of computer operation
    - I/O devices vary greatly
    - Various methods to control them
    - Performance management
    - New types of devices frequent

- Ports, busses, device controllers connect to various devices.

Overview

- I/O management is a major component of OS design and operation.
  - Important aspect of computer operation
  - I/O devices vary greatly
  - Various methods to control them
  - Performance management
  - New types of devices frequent

- Ports, busses, device controllers connect to various devices.

- Device drivers encapsulate device details.
  - Present uniform device-access interface to I/O subsystem.

I/O Hardware

I/O Hardware

- Incredible variety of I/O devices
  - Storage, e.g., disks, tapes
  - Transmission, e.g., network connections, bluetooth
  - Human-interface, e.g., screen, keyboard, mouse, audio in and out

I/O Hardware

- Incredible variety of I/O devices
    - Storage, e.g., disks, tapes
    - Transmission, e.g., network connections, bluetooth
    - Human-interface, e.g., screen, keyboard, mouse, audio in and out

- We only need to understand how the devices are attached and how the software can control the hardware.

Common Concepts in I/O Hardware

- Common concepts: signals from I/O devices interface with computer.

Common Concepts in I/O Hardware

- Common concepts: signals from I/O devices interface with computer.
- Port: connection point for device

Common Concepts in I/O Hardware

- Common concepts: signals from I/O devices interface with computer.

- Port: connection point for device

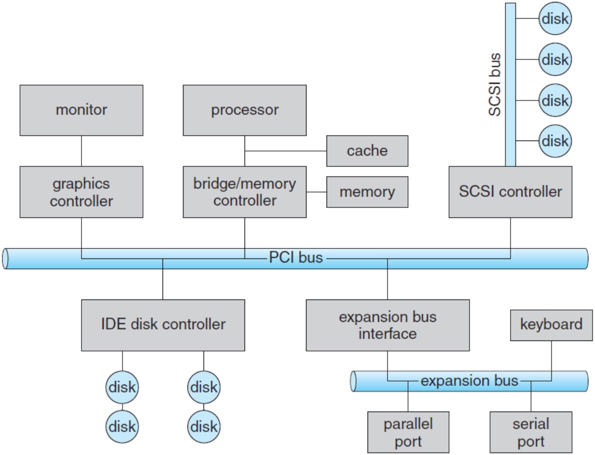
- Bus: set of wires and a protocol that specifies a set of messages that can be sent on the wires.

Common Concepts in I/O Hardware

- Common concepts: signals from I/O devices interface with computer.
- Port: connection point for device
- Bus: set of wires and a protocol that specifies a set of messages that can be sent on the wires.
- Controller: a collection of electronics that can operate a port, a bus, or a device.
  - Sometimes integrated and sometimes separate circuit board (host adapter)
  - Contains processor, microcode, private memory, bus controller, etc

Bus

- PCI bus: connects the processor-memory subsystem to fast devices.
- Expansion bus: connects relatively slow devices.



Processor/Controller Interaction

- How can the processor give commands and data to a controller to accomplish an I/O transfer?

Processor/Controller Interaction

- How can the processor give commands and data to a controller to accomplish an I/O transfer?

- Devices usually have registers where device driver places commands, addresses, and data.

Processor/Controller Interaction

- How can the processor give commands and data to a controller to accomplish an I/O transfer?
- Devices usually have registers where device driver places commands, addresses, and data.
- Processor/controller interaction:

Processor/Controller Interaction

- How can the processor give commands and data to a controller to accomplish an I/O transfer?

- Devices usually have registers where device driver places commands, addresses, and data.

- Processor/controller interaction:
  - Direct I/O instructions: triggers bus lines to select the proper device and to move bits into or out of a device register.

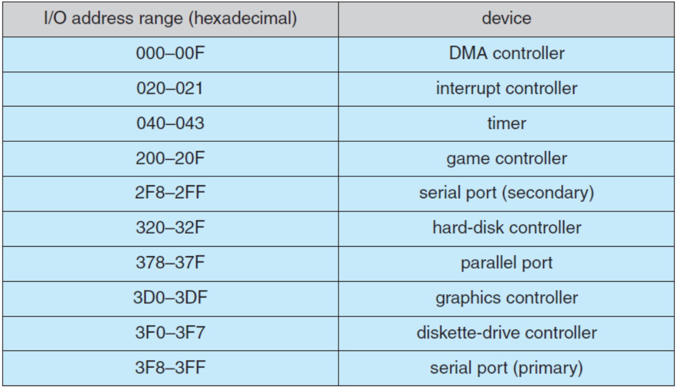
Processor/Controller Interaction

- How can the processor give commands and data to a controller to accomplish an I/O transfer?

- Devices usually have registers where device driver places commands, addresses, and data.

- Processor/controller interaction:
  - Direct I/O instructions: triggers bus lines to select the proper device and to move bits into or out of a device register.
  - Memory-mapped I/O: device data and command registers mapped to processor address space.

Device I/O Port Locations on PCs



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)

I/O Port Registers

- The data-in register: read by the host to get input.

I/O Port Registers

- The data-in register: read by the host to get input.
- The data-out register: written by the host to send output.

I/O Port Registers

- The data-in register: read by the host to get input.
- The data-out register: written by the host to send output.
- The status register: read by the host, and indicates states.
    - whether the current command has completed.
    - whether a byte is available to be read from the data-in register.
    - whether a device error has occurred.

I/O Port Registers

- The data-in register: read by the host to get input.
- The data-out register: written by the host to send output.
- The status register: read by the host, and indicates states.
  - whether the current command has completed.
  - whether a byte is available to be read from the data-in register.
  - whether a device error has occurred.
- The control register: written by the host to start a command or to change the mode of a device.

Host Device Interaction

- Polling
- Interrupt
- Direct memory access (DMA)

Polling (1/2)

- A handshake between the host and a controller.

Polling (1/2)

- A handshake between the host and a controller.
- Assume 2 bits for coordination. For each byte of I/O:

Polling (1/2)

- A handshake between the host and a controller.
- Assume 2 bits for coordination. For each byte of I/O:
  1. Host reads the busy bit from the status register until 0.

Polling (1/2)

- A handshake between the host and a controller.

- Assume 2 bits for coordination. For each byte of I/O:
  1. Host reads the busy bit from the status register until 0.
  2. Host sets the read or write bit and if write copies data into the data-out register.

Polling (1/2)

- A handshake between the host and a controller.
- Assume 2 bits for coordination. For each byte of I/O:
  1. Host reads the busy bit from the status register until 0.
  2. Host sets the read or write bit and if write copies data into the data-out register.
  3. Host sets the command-ready bit.

Polling (1/2)

- A handshake between the host and a controller.
- Assume 2 bits for coordination. For each byte of I/O:
  1. Host reads the busy bit from the status register until 0.
  2. Host sets the read or write bit and if write copies data into the data-out register.
  3. Host sets the command-ready bit.
  4. Controller sets the busy bit, executes transfer.

Polling (1/2)

- A handshake between the host and a controller.
- Assume 2 bits for coordination. For each byte of I/O:
  1. Host reads the busy bit from the status register until 0.
  2. Host sets the read or write bit and if write copies data into the data-out register.
  3. Host sets the command-ready bit.
  4. Controller sets the busy bit, executes transfer.
  5. Controller clears the busy bit, error bit, and command-ready bit when transfer done.

Polling (2/2)

- Step 1 is busy-wait cycle (polling) to wait for I/O from device.
  - Reasonable if device is fast.
  - But inefficient if device slow.
  - CPU switches to other tasks? but if miss a cycle data overwritten/lost.

Interrupts

- Polling can happen in 3 instruction cycles.
    - `read` status, `logical-and` to extract status bit, and `branch` if not zero.
    - Inefficient, but more efficient way?

Interrupts

- Polling can happen in 3 instruction cycles.
  - `read` status, `logical-and` to extract status bit, and `branch` if not zero.
  - Inefficient, but more efficient way?

- CPU interrupt-request line is triggered by I/O device.
  - Checked by processor after each instruction.
  - Saves state and jumps to interrupt-handler routine at a fixed address in memory.

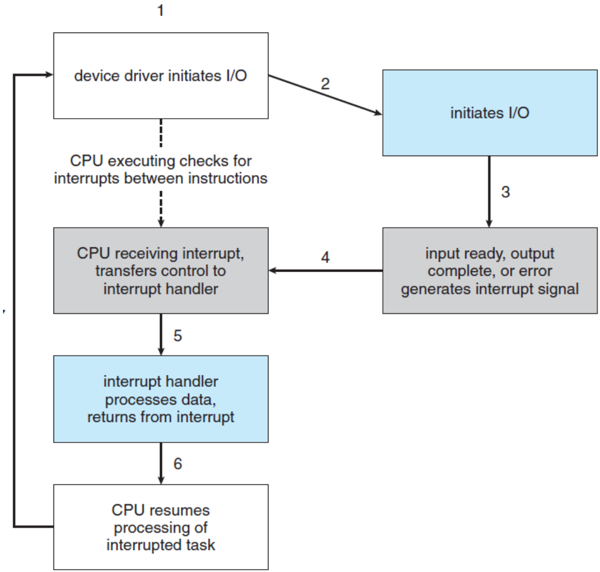
Interrupts

- Polling can happen in 3 instruction cycles.
  - `read` status, `logical-and` to extract status bit, and `branch` if not zero.
  - Inefficient, but more efficient way?

- CPU interrupt-request line is triggered by I/O device.
  - Checked by processor after each instruction.
  - Saves state and jumps to interrupt-handler routine at a fixed address in memory.

- Two interrupt request lines:
  - Nonmaskable: reserved for events such as unrecoverable memory errors.
  - Maskable: it can be turned off by the CPU.

Interrupt-Driven I/O Cycle



1

device driver initiates I/O

2

initiates I/O

CPU executing checks for interrupts between instructions

3

CPU receiving interrupt, transfers control to interrupt handler

4

input ready, output complete, or error generates interrupt signal

5

interrupt handler processes data, returns from interrupt

6

CPU resumes processing of interrupted task

Interrupt Vector

- The interrupt mechanism accepts an address: a number that selects a specific interrupt-handling routine.

Interrupt Vector

- The interrupt mechanism accepts an address: a number that selects a specific interrupt-handling routine.

- This address is an offset in a table called the interrupt vector.

Interrupt Vector

- The interrupt mechanism accepts an address: a number that selects a specific interrupt-handling routine.
- This address is an offset in a table called the interrupt vector.
- The interrupt vector contains the memory addresses of specialized interrupt handlers.

Interrupt Vector

- The interrupt mechanism accepts an address: a number that selects a specific interrupt-handling routine.

- This address is an offset in a table called the interrupt vector.

- The interrupt vector contains the memory addresses of specialized interrupt handlers.

- Computers have more devices than they have address elements in the interrupt vector.

Interrupt Vector

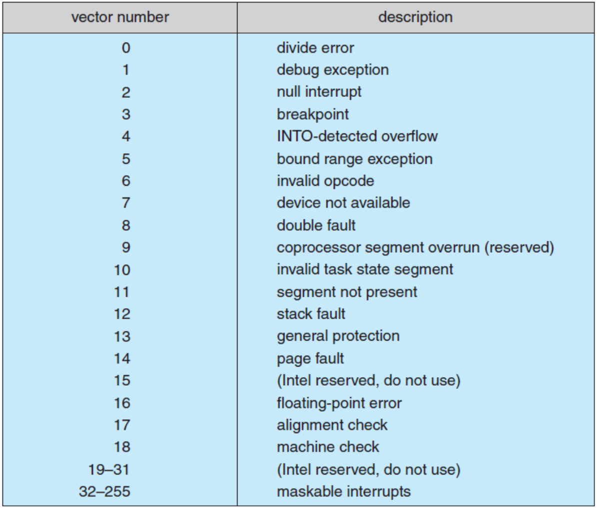
- The interrupt mechanism accepts an address: a number that selects a specific interrupt-handling routine.

- This address is an offset in a table called the interrupt vector.

- The interrupt vector contains the memory addresses of specialized interrupt handlers.

- Computers have more devices than they have address elements in the interrupt vector.
    - Use interrupt chaining: each element in the interrupt vector points to the head of a list of interrupt handlers.

Intel Pentium Processor Event-Vector Table



vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Other Interrupt Usages

- Interrupt mechanism also used for exceptions.
    - Divide by zero, terminate process, hardware error.

Other Interrupt Usages

- Interrupt mechanism also used for exceptions.
  - Divide by zero, terminate process, hardware error.
- Page fault executes when memory access error.

Other Interrupt Usages

- Interrupt mechanism also used for exceptions.
  - Divide by zero, terminate process, hardware error.
- Page fault executes when memory access error.
- System call executes via trap to trigger kernel to execute request.

Other Interrupt Usages

- Interrupt mechanism also used for exceptions.
    - Divide by zero, terminate process, hardware error.
- Page fault executes when memory access error.
- System call executes via trap to trigger kernel to execute request.
- Multi-CPU systems can process interrupts concurrently.

Other Interrupt Usages

- Interrupt mechanism also used for exceptions.
  - Divide by zero, terminate process, hardware error.
- Page fault executes when memory access error.
- System call executes via trap to trigger kernel to execute request.
- Multi-CPU systems can process interrupts concurrently.
- Used for time-sensitive processing, frequent, must be fast.

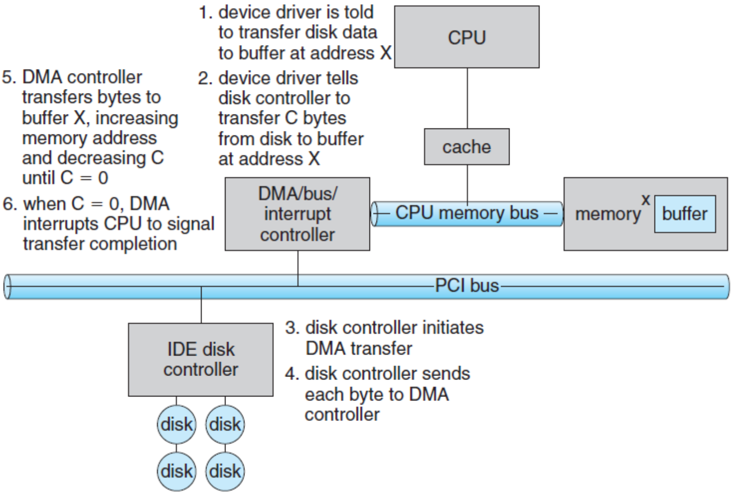
Direct Memory Access (1/2)

- Used to avoid programmed I/O (one byte at a time) for large data movement.
- Requires Direct Memory Access (DMA) controller
- Bypasses CPU to transfer data directly between I/O device and memory.
- Version that is aware of virtual addresses can be even more efficient.

Direct Memory Access (2/2)

- OS writes DMA command block into memory.
  - Source and destination addresses
  - Read or write mode
  - Count of bytes
  - Writes location of command block to DMA controller
  - Bus mastering of DMA controller - grabs bus from CPU: cycle stealing from CPU but still much more efficient
  - When done, interrupts to signal completion

Six Step Process to Perform DMA Transfer



1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion

Application I/O Interface

Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes.

Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes.
- Device-driver layer hides differences among I/O controllers from kernel.

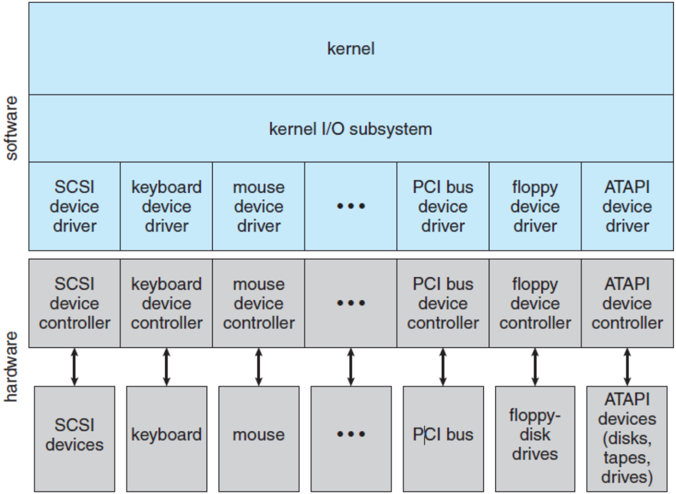
Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes.
- Device-driver layer hides differences among I/O controllers from kernel.
- New devices talking already-implemented protocols need no extra work.

Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes.
- Device-driver layer hides differences among I/O controllers from kernel.
- New devices talking already-implemented protocols need no extra work.
- Each OS has its own I/O subsystem structures and device driver frameworks.

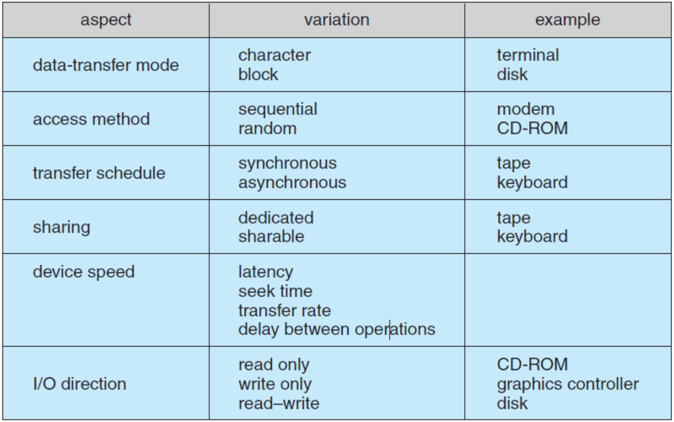
A Kernel I/O Structure



Characteristics of I/O Devices (1/2)

- Devices vary in many dimensions
    - Data-transfer mode: character or block
    - Access method: sequential or random-access
    - Transfer schedule: synchronous or asynchronous (or both)
    - Sharing: sharable or dedicated
    - Device speed: speed of operation
    - I/O direction: read-write, read only, or write only

Characteristics of I/O Devices (2/2)



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

Character Devices

- Character devices include keyboards, mouse, serial ports.

Character Devices

- Character devices include keyboards, mouse, serial ports.
- A character device transfers bytes one by one.

Character Devices

- Character devices include keyboards, mouse, serial ports.
- A character device transfers bytes one by one.
- Commands include `get()` and `put()`.

Block Devices

- Block devices include disk drives.

Block Devices

- Block devices include disk drives.
- Commands include `read()` and `write()` and `seek()` for random-access devices.

Block Devices

- Block devices include disk drives.
- Commands include `read()` and `write()` and `seek()` for random-access devices.
- Raw I/O: access a block device as a simple linear array of blocks.

Block Devices

- Block devices include disk drives.
- Commands include `read()` and `write()` and `seek()` for random-access devices.
- Raw I/O: access a block device as a simple linear array of blocks.
- Direct I/O: disable buffering and locking.

Block Devices

- Block devices include disk drives.
- Commands include `read()` and `write()` and `seek()` for random-access devices.
- Raw I/O: access a block device as a simple linear array of blocks.
- Direct I/O: disable buffering and locking.
- Memory-mapped file access: file mapped to virtual memory and clusters brought via demand paging.

Network Devices

- Varying enough from block and character to have own interface.

Network Devices

- Varying enough from block and character to have own interface.
- Linux, Unix, Windows and many others include socket interface.
    - Separates network protocol from network operation.
    - Includes `select()` functionality.

Clocks and Timers

- Provide current time, elapsed time, and timer (trigger operation X at time T)

Clocks and Timers

- Provide current time, elapsed time, and timer (trigger operation X at time T)

- Programmable interval timer, the hardware used for timings, and periodic interrupts.

Clocks and Timers

- Provide current time, elapsed time, and timer (trigger operation X at time T)
- Programmable interval timer, the hardware used for timings, and periodic interrupts.
- Normal resolution about 1/60 second.

Clocks and Times

- Provide current time, elapsed time, and timer (trigger operation X at time T)
- Programmable interval timer, the hardware used for timings, and periodic interrupts.
- Normal resolution about 1/60 second.
- Some systems provide higher-resolution timers.

Blocking, Nonblocking and Asynchronous I/O

- Blocking: process suspended until I/O completed
  - Insufficient for some needs

Blocking, Nonblocking and Asynchronous I/O

- Blocking: process suspended until I/O completed
  - Insufficient for some needs

- Nonblocking: I/O call returns as much as available
  - User interface, data copy (buffered I/O)

Blocking, Nonblocking and Asynchronous I/O

- Blocking: process suspended until I/O completed
  - Insufficient for some needs
- Nonblocking: I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading

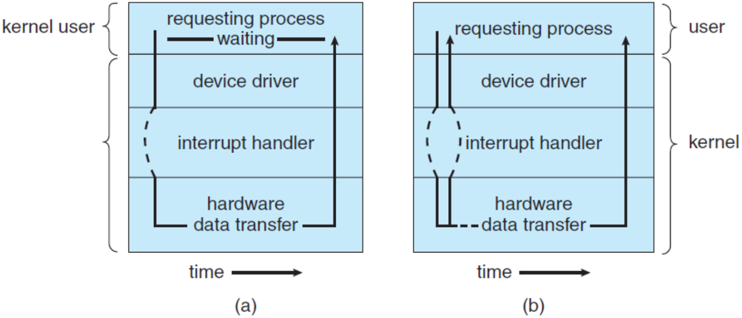
Blocking, Nonblocking and Asynchronous I/O

- Blocking: process suspended until I/O completed
  - Insufficient for some needs
- Nonblocking: I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - `select()` to find if data ready then `read()` or `write()` to transfers.

Blocking, Nonblocking and Asynchronous I/O

- Blocking: process suspended until I/O completed
  - Insufficient for some needs
- Nonblocking: I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - `select()` to find if data ready then `read()` or `write()` to transfers.
- Asynchronous: process runs while I/O executes
  - I/O subsystem signals process when I/O completed.

Synchronous vs. Asynchronous I/O Methods



Vectored I/O

- Vectored I/O allows one system call to perform multiple I/O operations.

Vectored I/O

- Vectored I/O allows one system call to perform multiple I/O operations.
- The scatter-gather method is better than multiple individual I/O calls.

Vectored I/O

- Vectored I/O allows one system call to perform multiple I/O operations.
- The scatter-gather method is better than multiple individual I/O calls.
  - Decreases context switching and system call overhead.

Vectored I/O

- Vectored I/O allows one system call to perform multiple I/O operations.

- The scatter-gather method is better than multiple individual I/O calls.
  - Decreases context switching and system call overhead.
  - Some versions provide atomicity: avoid for example worry about multiple threads changing data as reads/writes occurring.

Vectored I/O

- Vectored I/O allows one system call to perform multiple I/O operations.
- The scatter-gather method is better than multiple individual I/O calls.
  - Decreases context switching and system call overhead.
  - Some versions provide atomicity: avoid for example worry about multiple threads changing data as reads/writes occurring.
- For example, Unix `readve()` accepts a vector of multiple buffers to read into or write from.

Kernel I/O Subsystem

Kernel I/O Subsystem

- Kernels provide many services related to I/O:
  - Scheduling
  - Buffering
  - Caching
  - Spooling
  - Device reservation
  - Error handling

Scheduling (1/2)

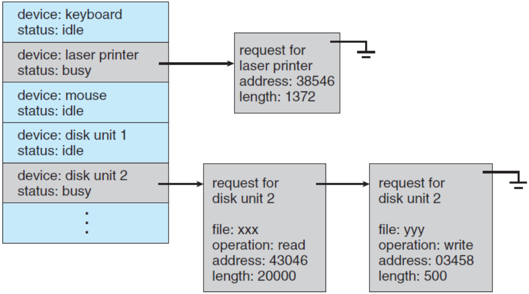
- Determine a good order in which to execute I/O requests.
- Some I/O request ordering via per-device queue.
- Some OSs try fairness.

Scheduling (2/2)

- In asynchronous I/O the kernel must be able to keep track of many I/O requests at the same time.

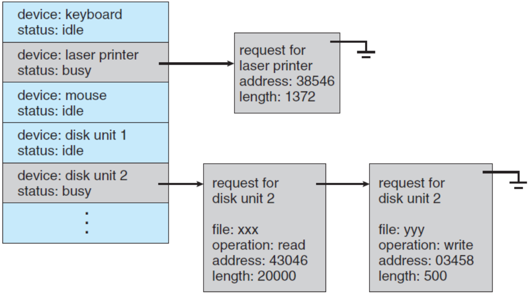
Scheduling (2/2)

- In asynchronous I/O the kernel must be able to keep track of many I/O requests at the same time.
  - The OS attaches the wait queue to a device-status table.



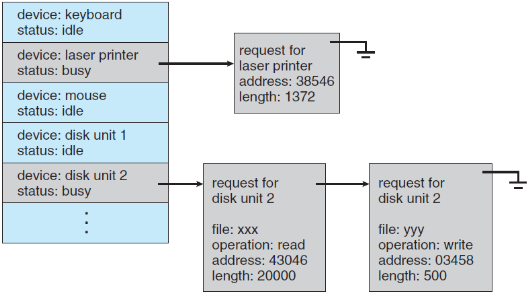
Scheduling (2/2)

- In asynchronous I/O the kernel must be able to keep track of many I/O requests at the same time.
  - The OS attaches the wait queue to a device-status table.
  - The table contains an entry for each I/O device.



Scheduling (2/2)

- In asynchronous I/O the kernel must be able to keep track of many I/O requests at the same time.
  - The OS attaches the wait queue to a device-status table.
  - The table contains an entry for each I/O device.
  - If the device is busy with a request, the type of request and other parameters will be stored in the table entry for that device.



Buffering and Caching

- Buffering: stores data in memory while transferring between devices.

Buffering and Caching

- Buffering: stores data in memory while transferring between devices.
    - To cope with device speed mismatch.

Buffering and Caching

- Buffering: stores data in memory while transferring between devices.
  - To cope with device speed mismatch.
  - To cope with device transfer size mismatch, e.g., fragmentation and reassembly of messages

Buffering and Caching

- Buffering: stores data in memory while transferring between devices.
  - To cope with device speed mismatch.
  - To cope with device transfer size mismatch, e.g., fragmentation and reassembly of messages
  - To maintain copy semantics, e.g., copy semantics

Buffering and Caching

- Buffering: stores data in memory while transferring between devices.
  - To cope with device speed mismatch.
  - To cope with device transfer size mismatch, e.g., fragmentation and reassembly of messages
  - To maintain copy semantics, e.g., copy semantics

- Caching: faster device holding copy of data.
  - Always just a copy
  - Key to performance
  - Sometimes combined with buffering

Spooling and Device Reservation

- Spooling: a buffer that holds output for a device.
  - If device can serve only one request at a time, i.e., printing

Spooling and Device Reservation

- Spooling: a buffer that holds output for a device.
  - If device can serve only one request at a time, i.e., printing
- Device reservation: provides exclusive access to a device.
  - System calls for allocation and de-allocation
  - Watch out for deadlock

Error Handling

- OS can recover from disk read, device unavailable, transient write failures

Error Handling

- OS can recover from disk read, device unavailable, transient write failures
    - Retry a read or write.

Error Handling

- OS can recover from disk read, device unavailable, transient write failures
    - Retry a read or write.
    - Track error frequencies, stop using device with increasing frequency of retry-able errors.

Error Handling

- OS can recover from disk read, device unavailable, transient write failures
  - Retry a read or write.
  - Track error frequencies, stop using device with increasing frequency of retry-able errors.
- Most return an error number or code when I/O request fails.

Error Handling

- OS can recover from disk read, device unavailable, transient write failures
    - Retry a read or write.
    - Track error frequencies, stop using device with increasing frequency of retry-able errors.

- Most return an error number or code when I/O request fails.

- System error logs hold problem reports.

I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions.

I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions.
- All I/O instructions defined to be privileged.

I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions.

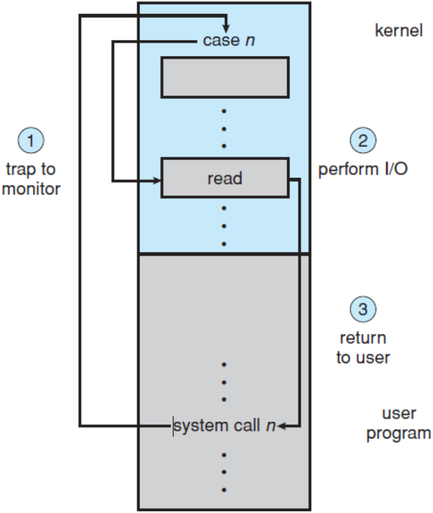
- All I/O instructions defined to be privileged.

- I/O must be performed via system calls.

I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions.
- All I/O instructions defined to be privileged.
- I/O must be performed via system calls.
- Memory-mapped and I/O port memory locations must be protected too.

Use of a System Call to Perform I/O



Kernel Data Structures

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state

Kernel Data Structures

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state
- Many complex data structures to track buffers, memory allocation, dirty blocks.

Kernel Data Structures

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state

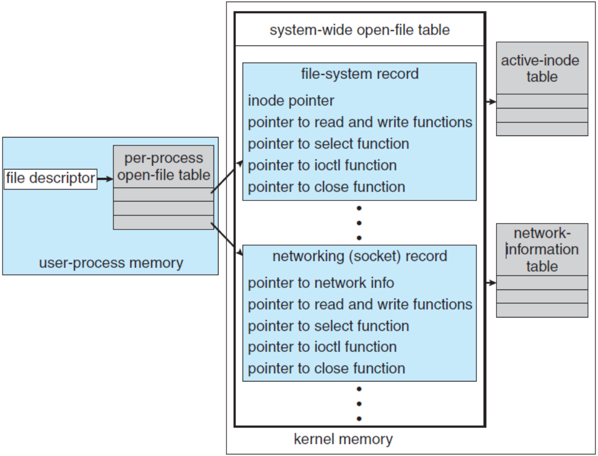
- Many complex data structures to track buffers, memory allocation, dirty blocks.

- Some use object-oriented methods and message passing to implement I/O

Kernel Data Structures

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state
- Many complex data structures to track buffers, memory allocation, dirty blocks.
- Some use object-oriented methods and message passing to implement I/O
- E.g., Windows uses message passing.
  - Message with I/O information passed from user mode into kernel.
  - Message modified as it flows through to device driver and back to process.

UNIX I/O Kernel Structure

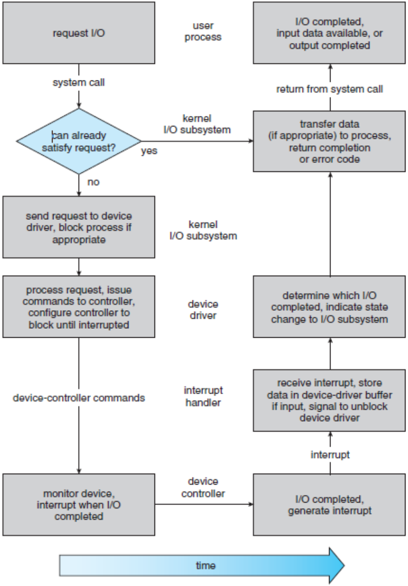


Transforming I/O Requests to Hardware Operations

I/O Requests to Hardware Operations

- Consider reading a file from disk for a process:
  - Determine device holding file
  - Translate name to device representation
  - Physically read data from disk into buffer
  - Make data available to requesting process
  - Return control to process

Life Cycle of An I/O Request



STREAMS

STREAMS (1/2)

- STREAM: a full-duplex communication channel between a user-level process and a device in Unix System V and beyond.

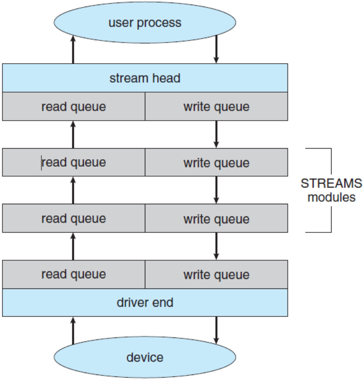
STREAMS (1/2)

- STREAM: a full-duplex communication channel between a user-level process and a device in Unix System V and beyond.

- A STREAM consists of:
  - STREAM head: interfaces with the user process.
  - Driver end: interfaces with the device.
  - Zero or more STREAM modules between them.

STREAMS (2/2)

- Each module contains a read queue and a write queue.
- Message passing is used to communicate between queues.
- Asynchronous internally, synchronous where user process communicates with stream head.

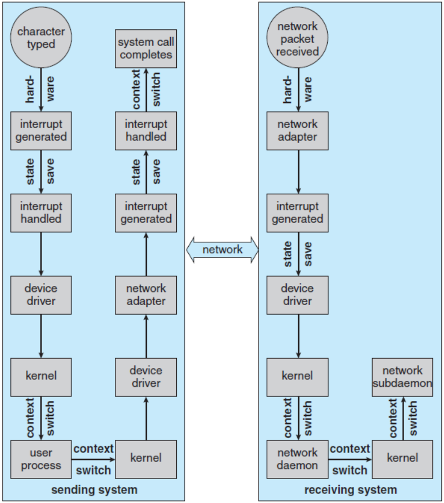


Performance

Performance

- I/O is a major factor in system performance:
  - Demands CPU to execute device driver, kernel I/O code
  - Context switches due to interrupts
  - Data copying
  - Network traffic especially stressful

Inter-computer Communications



Improving Performance

- Reduce number of context switches

Improving Performance

- Reduce number of context switches
- Reduce data copying

Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling

Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA

Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Use smarter hardware devices

Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Use smarter hardware devices
- Balance CPU, memory, bus, and I/O performance for highest throughput

Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Use smarter hardware devices
- Balance CPU, memory, bus, and I/O performance for highest throughput
- Move user-mode processes to kernel

Summary

Summary

- I/O hardware: port, bus, controller

Summary

- I/O hardware: port, bus, controller
- I/O port registers: data-in, data-out, status, control

Summary

- I/O hardware: port, bus, controller
- I/O port registers: data-in, data-out, status, control
- Host-device interaction: polling, interrupt, DMA

Summary

- I/O hardware: port, bus, controller
- I/O port registers: data-in, data-out, status, control
- Host-device interaction: polling, interrupt, DMA
- Devices: char, block, network

Summary

- I/O hardware: port, bus, controller
- I/O port registers: data-in, data-out, status, control
- Host-device interaction: polling, interrupt, DMA
- Devices: char, block, network
- Kernel I/O: schedulling, buffering, caching, spooling, device reservation, error handling

Summary

- I/O hardware: port, bus, controller
- I/O port registers: data-in, data-out, status, control
- Host-device interaction: polling, interrupt, DMA
- Devices: char, block, network
- Kernel I/O: schedulling, buffering, caching, spooling, device reservation, error handling
- STREAMS

Summary

- I/O hardware: port, bus, controller
- I/O port registers: data-in, data-out, status, control
- Host-device interaction: polling, interrupt, DMA
- Devices: char, block, network
- Kernel I/O: schedulling, buffering, caching, spooling, device reservation, error handling
- STREAMS
- Performance

Questions?

Acknowledgements

Some slides were derived from Avi Silberschatz slides.