Operating Systems (A)
(Honor Track)

Lecture 20: I/O

Yao Guo (郭耀)

Peking University

Fall 2021

Acknowledgements: Prof. Xiangqun Chen & Tao Wang at PKU and Prof. Yuanyuan Zhou at UCSD

Buzz Words

- **Programmed I/O (PIO)**
- **Directed memory access (DMA)**
- **Single buffer**
- **Double buffer**
- **Circular buffer**
- **RAID**

This Lecture

I/O

I/O Devices

I/O Software Design

I/O Examples

RAID

Categories of I/O Devices

- A difficult area of OS design
  - Difficult to develop a consistent solution due to a wide variety of devices and applications

Input/output is an often neglected, but important, topic.

- Categories of I/O devices:
  - Human readable
  - Machine readable
  - Communications

Human Readable

- Devices used to communicate with the user
- Printers and terminals
  - Video display
  - Keyboard
  - Mouse
  - Touch screen

Machine Readable

- Devices used to communicate with electronic equipment
  - Disk drives
  - USB keys
  - Sensors
  - Controllers
  - Actuators

Communication

- Devices used to communicate with remote devices
  - Digital line drivers
  - Network adaptors
  - Modems
  - Routers

A Different Classification

- Block devices
  - A block device stores information in fixed-size blocks, each one with its own address
  - Examples: Hard disks, Blu-ray discs, and USB sticks

- Character devices
  - A character device delivers or accepts a stream of characters, without regard to any block structure
  - Examples: Printers, network interfaces, mice (for pointing), rats (for psychology lab experiments), and most other devices that are not disk-like

Data Rate

- Data rate difference between different devices

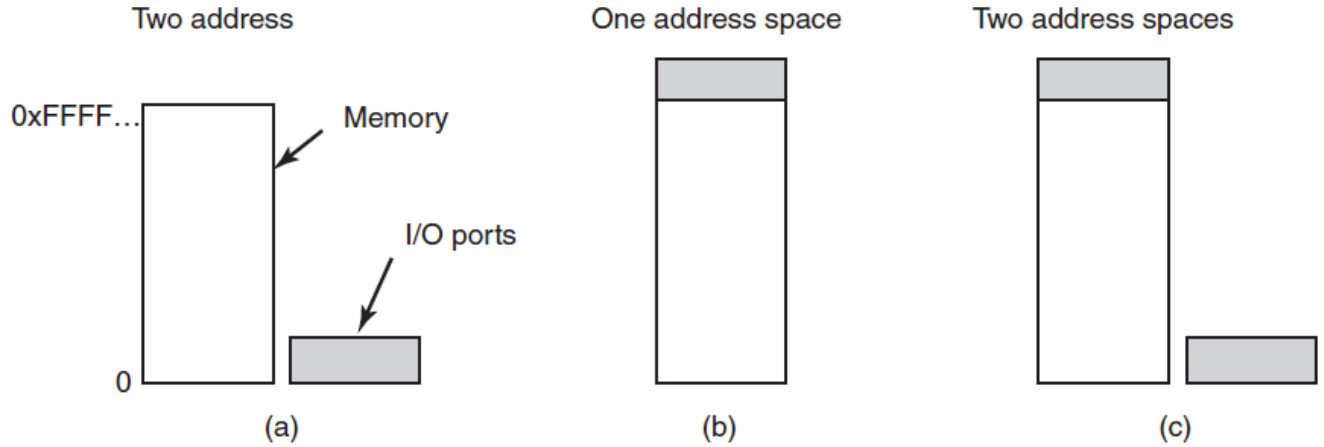
Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner at 300 dpi	1 MB/sec
Digital camcorder	3.5 MB/sec
4x Blu-ray disc	18 MB/sec
802.11n Wireless	37.5 MB/sec
USB 2.0	60 MB/sec
FireWire 800	100 MB/sec
Gigabit Ethernet	125 MB/sec
SATA 3 disk drive	600 MB/sec
USB 3.0	625 MB/sec
SCSI Ultra 5 bus	640 MB/sec
Single-lane PCIe 3.0 bus	985 MB/sec
Thunderbolt 2 bus	2.5 GB/sec
SONET OC-768 network	5 GB/sec

Device Controllers

- I/O units often consist of a mechanical component and an electronic component.
  - It is possible to separate the two portions to provide a more modular and general design.
- The electronic component is called the device controller or adapter.
  - The controller's job is to convert the serial bit stream into a block of bytes and perform error correction if needed
  - The interface between the controller and device often has some standards
    - For example, hard drive interfaces include : SATA, SCSI, USB, Thunderbolt, or FireWire (IEEE 1394) interfaces

Addressing the Device Controllers

- How to use device controllers?
  - Each controller has a few registers that are used for communicating with the CPU
  - Many devices have a data buffer that the OS can read and write
- How to communicate with device controllers?



Separate I/O and memory space. Memory-mapped I/O. Hybrid.

Memory-Mapped I/O

- Map all the control registers into the memory space
  - Introduced with the PDP-11
  - X86 (IBM PC) uses the hybrid approach
- Treat I/O reads as memory reads
  - Every I/O device compares the address lines to the range of addresses that it services, then responds accordingly
- Advantages of memory-mapped I/O
  - An I/O device driver can be written entirely in C
  - No special protection mechanism is needed to keep user processes from performing I/O
  - Every instruction that can reference memory can also reference control registers
- Drawbacks: caching might be dangerous
  - Needs to disable caching selectively

Dedicated Memory Bus

- Single-bus: only one address space, memory modules and all I/O devices must examine all memory references
- Dual-bus: dedicated high speed memory bus, how to distinguish between memory and I/O accesses in memory mapped-I/O?
  - Needs special hardware/mechanism to check/route accesses to I/O controllers

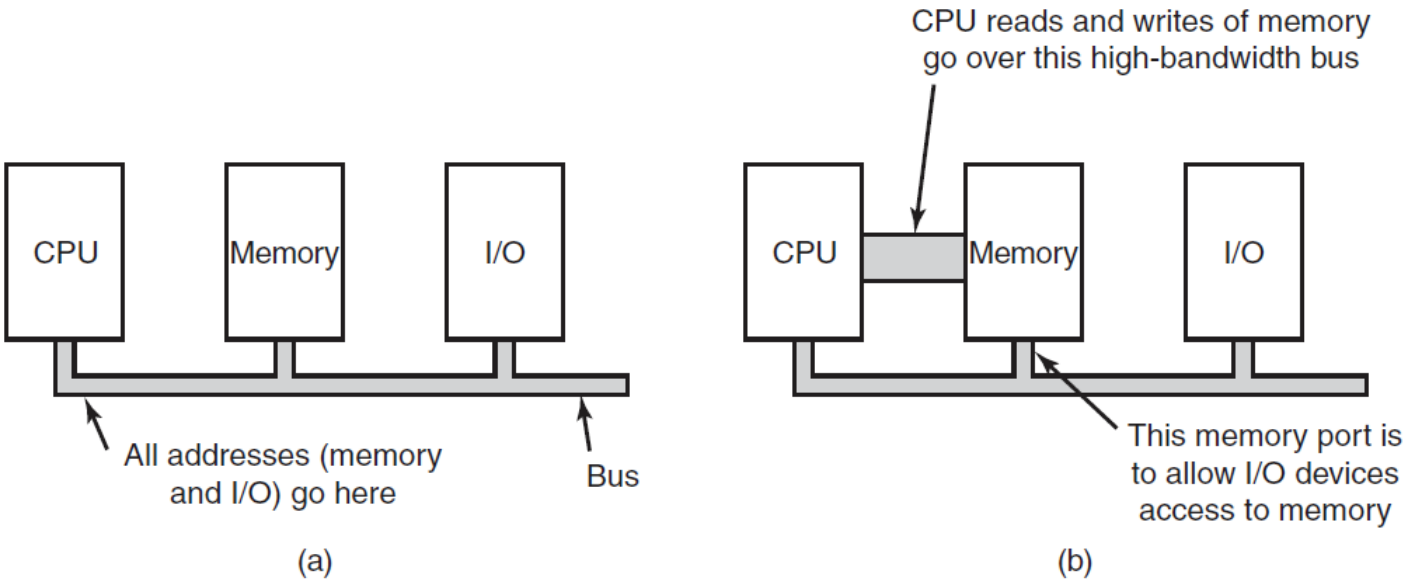


Figure 5-3. (a) A single-bus architecture. (b) A dual-bus memory architecture.

Techniques for Performing I/O

- Programmed I/O (PIO)
  - Processor issues an I/O command, on behalf of a process, to an I/O module;
  - May need busy waiting for the I/O operation to complete
- Interrupt-driven I/O
  - CPU can be released while waiting for I/O
  - An interrupt will be generated when I/O operation is completed
- Direct memory access (DMA)

	No Interrupts	**Use of Interrupts**
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

Direct Memory Access

- Processor delegates I/O operation to the DMA module
- DMA module transfers data directly to or from memory
- When complete DMA module sends an interrupt signal to the processor

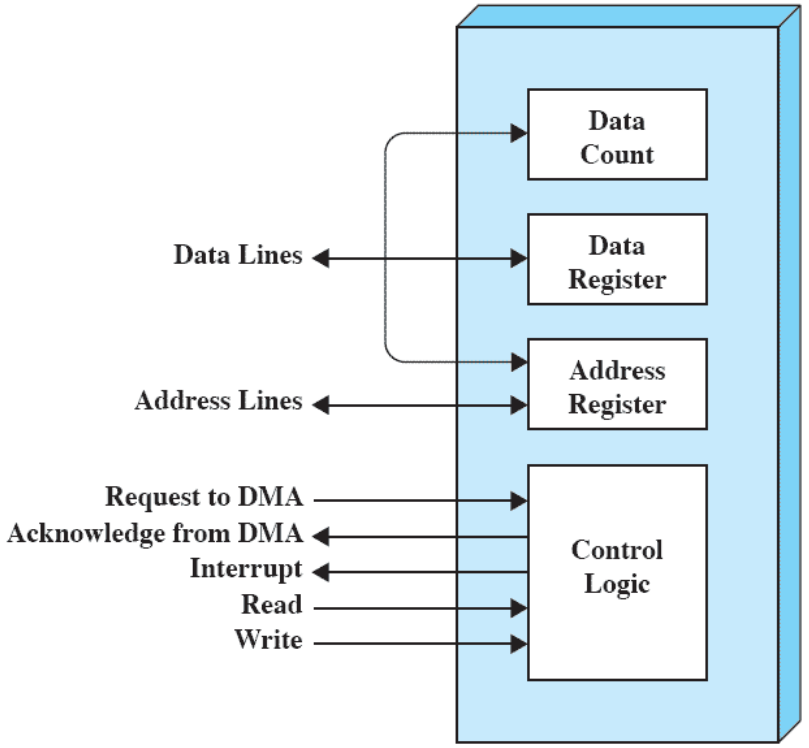
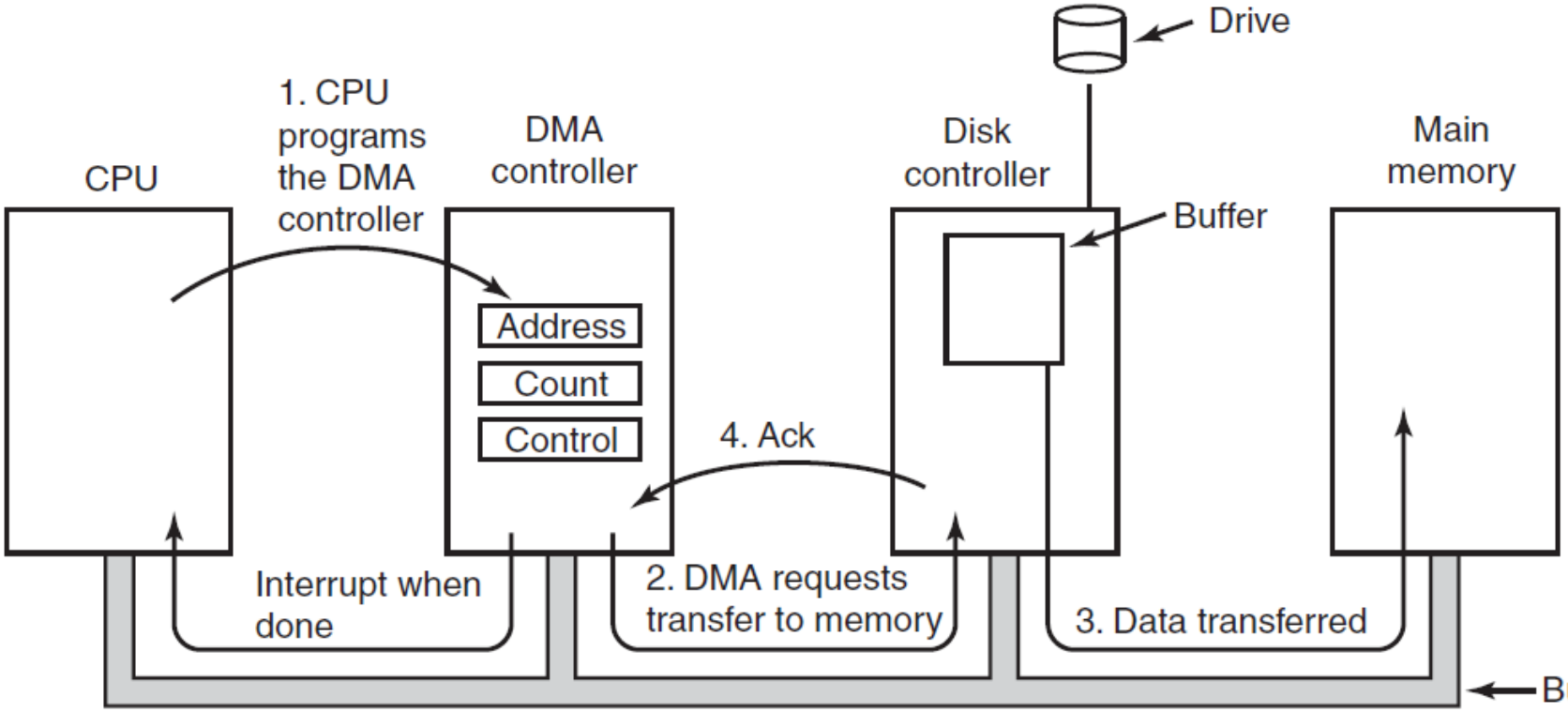


Figure 11.2 Typical DMA Block Diagram

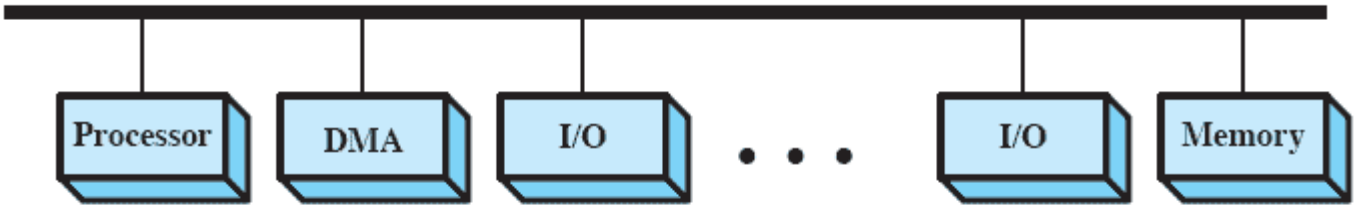
DMA in Action

- CPU programs the DMA controller by setting its registers (step 1)
- DMA controller initiates the transfer by issuing a read request (step 2)
- Data transferred and written to memory (step 3)
- Disk controller sends an acknowledgement signal to the DMA (step 4)



DMA Configurations: Single Bus

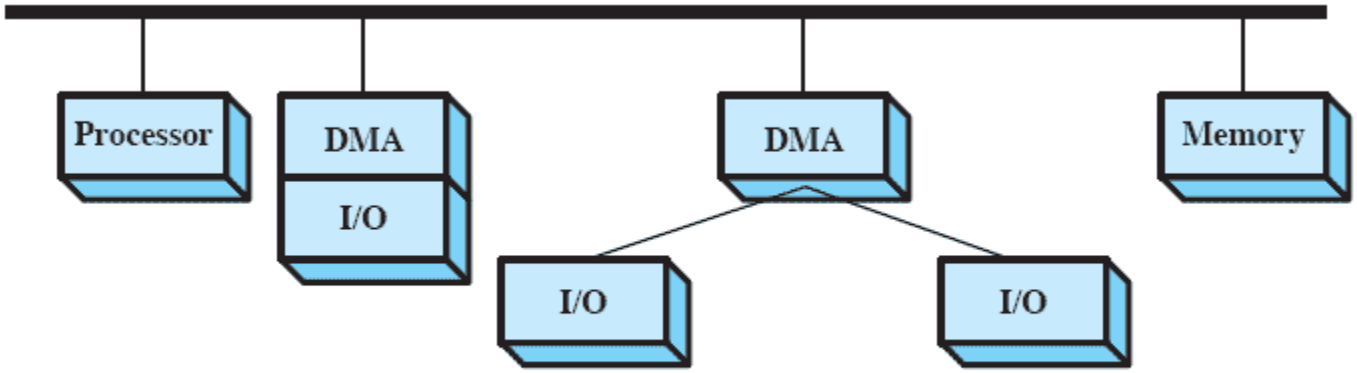
- DMA can be configured in several ways
- Shown here, **all modules share the same system bus**
  - The DMA module, acting as a surrogate processor, uses programmed I/O to exchange data between memory and an I/O module through the DMA module.
  - This is clearly **inefficient**: As with processor-controlled programmed I/O, each transfer of a word consumes two bus cycles (transfer request followed by transfer).



(a) Single-bus, detached DMA

DMA Configurations: Integrated DMA & I/O

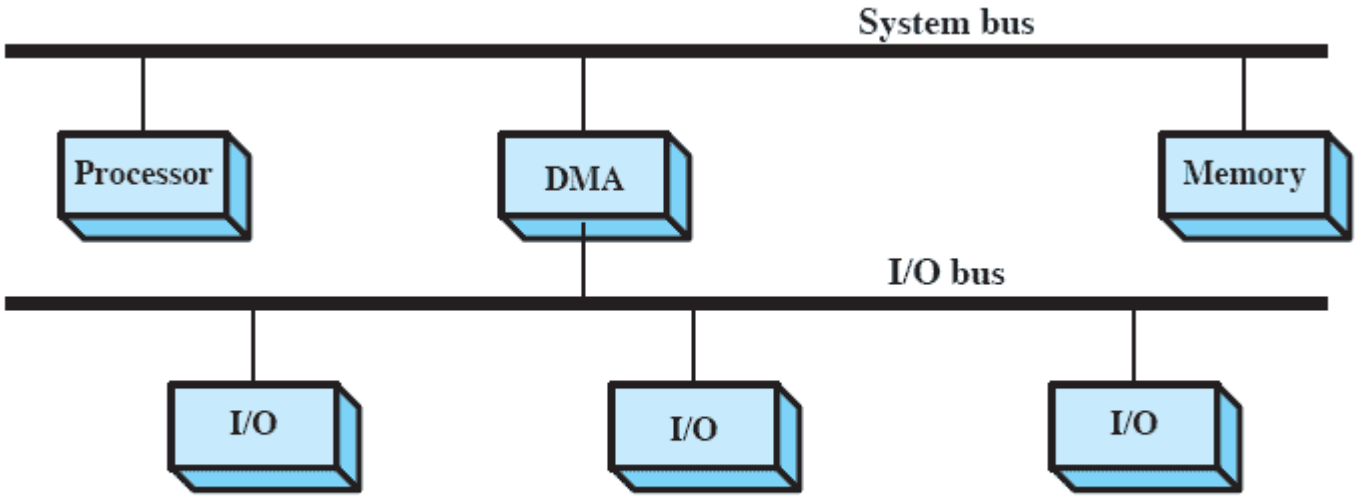
- **Direct Path** between DMA and I/O modules
  - This substantially cuts the required bus cycles
  - The DMA logic may actually be a part of an I/O module, or it may be a separate module that controls one or more I/O modules.



(b) Single-bus, Integrated DMA-I/O

DMA Configurations: I/O Bus

- **Dedicated I/O Bus**
  - This reduces the number of I/O interfaces in the DMA module to one
  - provides for an easily expandable configuration.



(c) I/O bus

Evolution of the I/O Function

1. Processor directly controls a peripheral device
2. Controller or I/O module is added (PIO)
   - Processor uses programmed I/O without interrupts
   - Processor does not need to handle details of external devices
3. Controller or I/O module with interrupts
   - Efficiency improves as processor does not spend time waiting for an I/O operation to be performed
4. Direct Memory Access
   - Blocks of data are moved into memory without involving the processor
   - Processor involved at beginning and end only
5. I/O module as a separate processor
   - CPU directs the I/O processor to execute an I/O program in main memory
6. I/O module as a computer
   - I/O module has its own local memory
   - Commonly used to control communications with interactive terminals

This Lecture

I/O

I/O Devices

I/O Software Design

I/O Examples

RAID

Design Goals

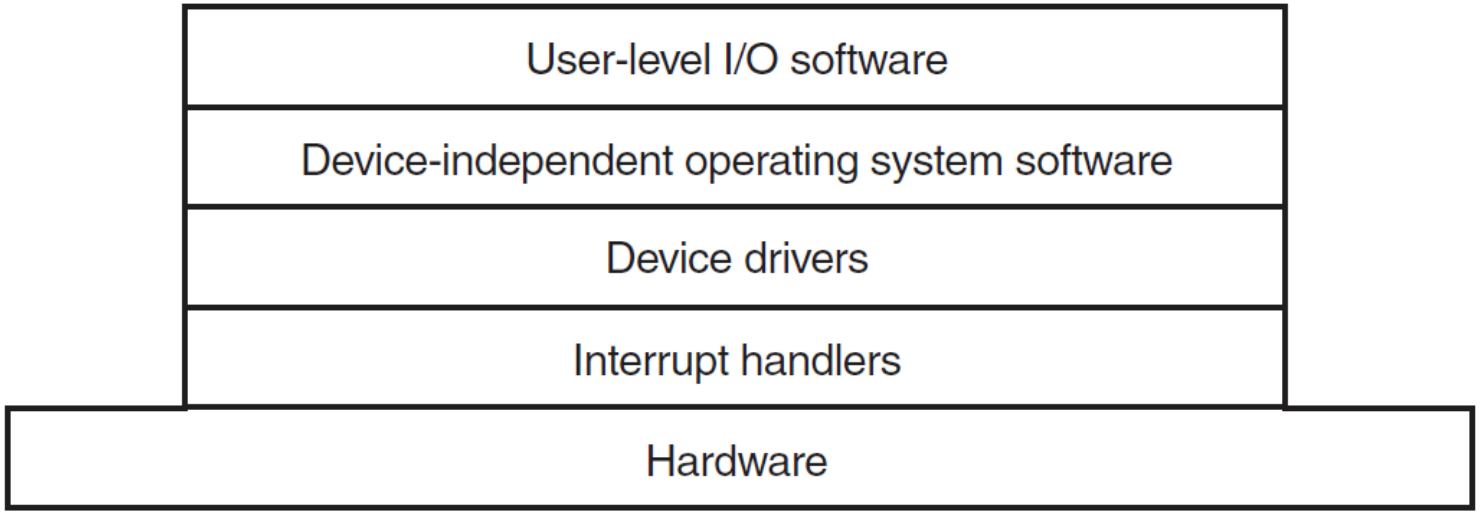
- Generality
- Efficiency

Goals: Generality

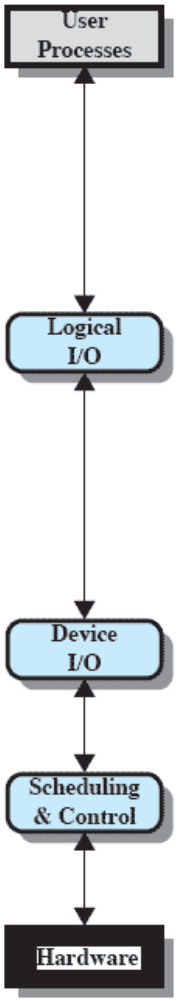
- For simplicity and freedom from error, it is desirable to handle all devices in a uniform manner.
  - This applies both to the way in which processes **view** I/O devices and the way in which the OS **manages** I/O devices and operations.
- Because of the diversity of device characteristics, it is difficult in practice to achieve true generality.
- What can be done is to use a hierarchical, modular approach to the design of the I/O function.
  - This hides most of the details of device I/O in lower-level routines
  - User processes and upper levels of the operating system see devices in terms of general functions, such as read, write, open, close, lock, unlock.

Hierarchical Design

- I/O software is typically organized in four layers
  - Each layer has a well-defined function to perform and a well-defined interface to the adjacent layers
  - Each layer relies on the next lower layer to perform more primitive functions



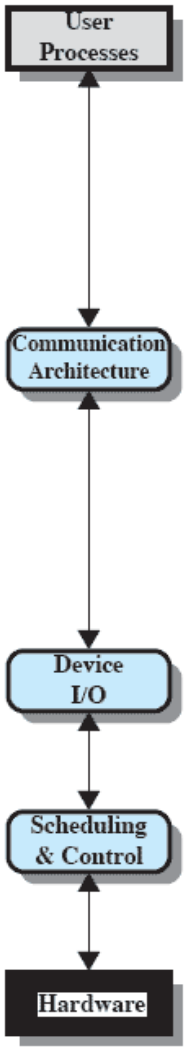
Local Peripheral Device



(a) Local peripheral device

- Logical I/O:
  - Deals with the device as a logical resource
- Device I/O:
  - Converts requested operations into sequence of I/O instructions
- Scheduling and Control
  - Performs actual queuing and control operations

Communications Port



(b) Communications port

- Similar to previous but the logical I/O module is replaced by a communications architecture
  - This consist of a number of layers
  - An example is TCP/IP

File System

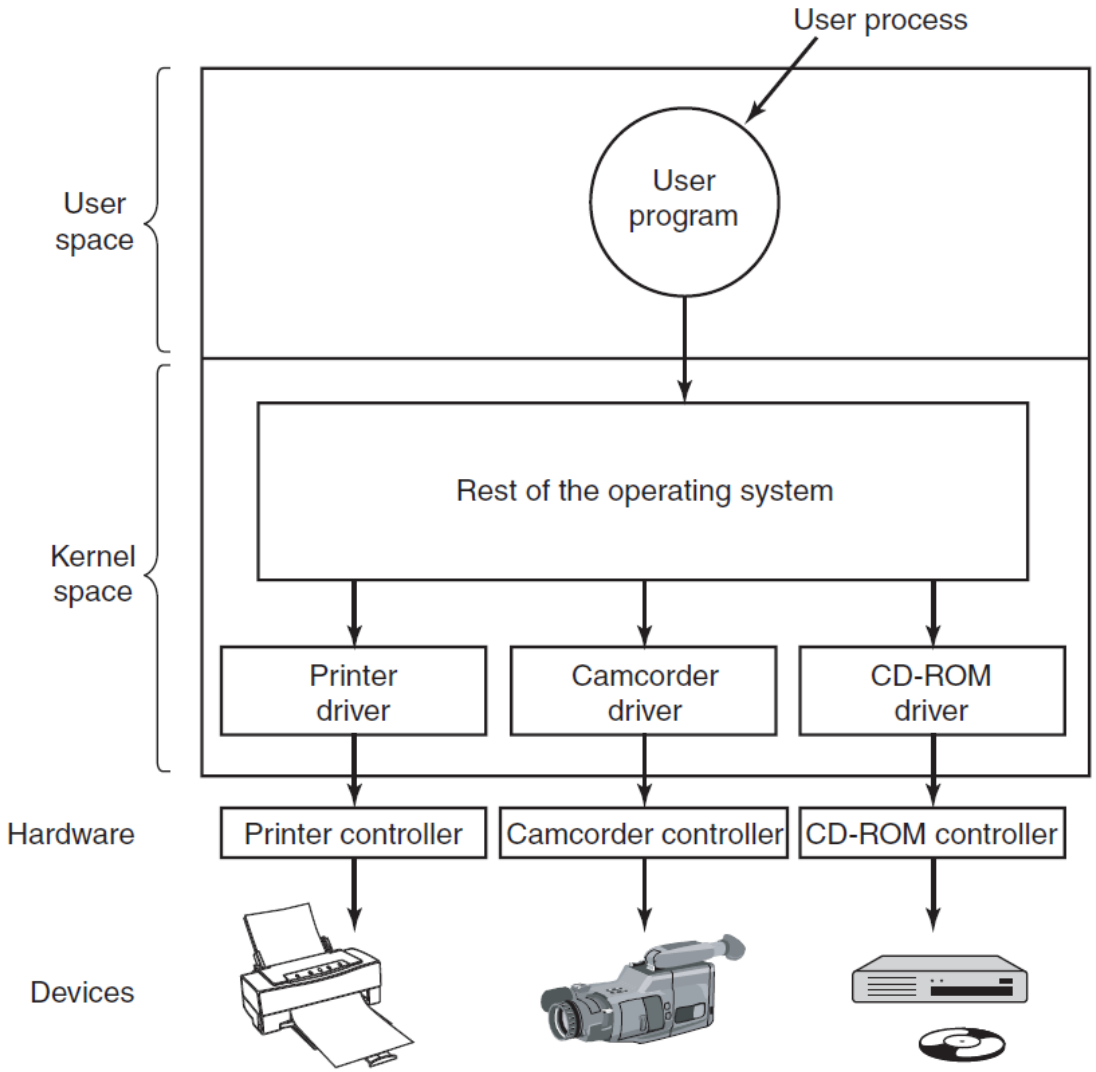
(c) File system

- Directory management
  - Concerned with user operations affecting file directory
- File System
  - Logical structure and operations
- Physical organization
  - Converts logical names to physical addresses

Device Drivers

- Device drivers:
  - Each I/O device attached to a computer needs some device-specific code for controlling it
  - Generally written by the device's manufacturer and delivered along with the device
- The device driver normally has to be part of the operating system kernel
  - Buggy device drivers are often the reasons why an OS crashes
- Many device drivers have standard interfaces
  - For example, a standard interface that all block drivers must support and a second standard interface that all character drivers must support

Device Driver Organization



Device-Independent I/O Software

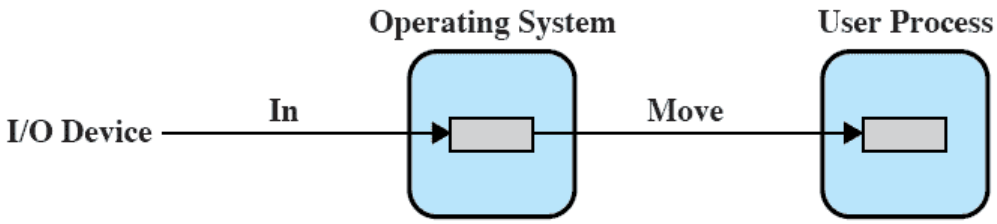
- Provides device-independent functions on top of the device driver
  - Uniform interfacing for device drivers
  - Buffering
  - Error reporting
  - Allocating and releasing dedicated devices
  - Providing a device-independent block size
- Some functions that could be done in a device-independent way may actually be done in the drivers, for efficiency or other reasons

Goals: Efficiency

- Most I/O devices extremely slow compared to main memory
- Use of multiprogramming allows for some processes to be waiting on I/O while another process executes
- I/O cannot keep up with processor speed
  - **Swapping** is used to bring in ready processes
  - But this is an I/O operation itself

I/O Buffering

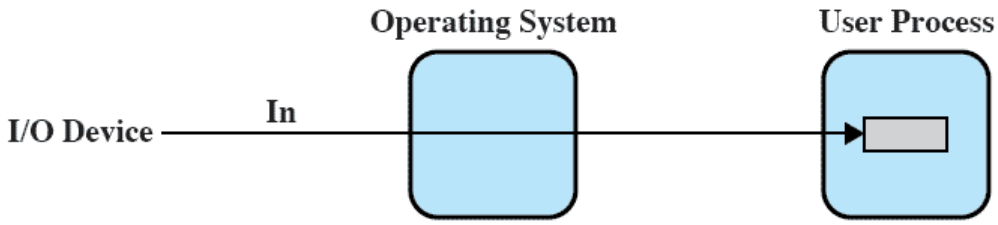
- It may be more efficient to perform input transfers in advance of requests being made and to perform output transfers some time after the request is made



(b) Single buffering

No Buffer

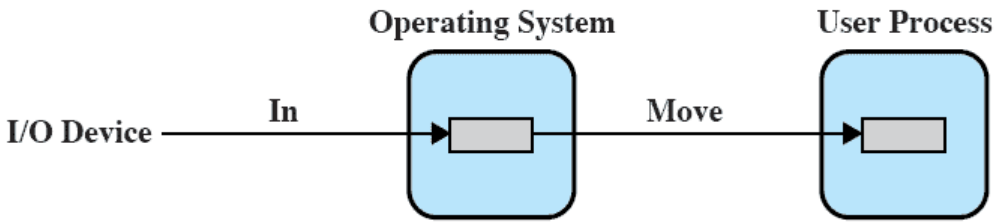
- Without a buffer, the OS directly access the device as and when it needs



(a) No buffering

Single Buffer

- Operating system assigns a buffer in main memory for an I/O request



(b) Single buffering

- Can be used for block-oriented devices and stream-oriented devices

Block-oriented Device

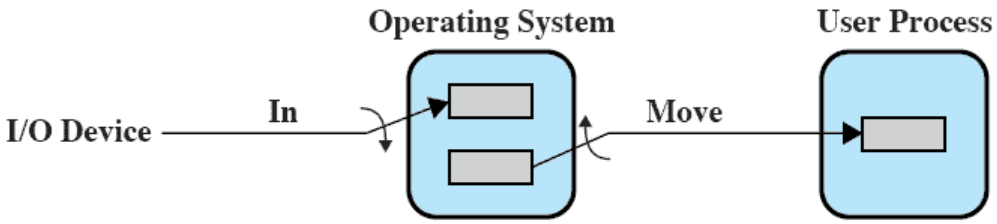
- Information is stored in fixed sized blocks
- Transfers are made a block at a time
  - Can reference data by block number
  - Example: disks and USB keys
- Input transfers made to buffer
- Block moved to user space when needed
- The next block is moved into the buffer
  - *Read ahead* or *Anticipated input*
  - Often a reasonable assumption as data is usually accessed sequentially

Stream-oriented Single Buffer

- Transfer information as a stream of bytes
  - Example: terminals, printers, communication ports, mouse and other pointing devices
- Line-at-time or Byte-at-a-time
  - Terminals often deal with one line at a time with carriage return signaling the end of the line
  - Byte-at-a-time suites devices where a single keystroke may be significant
    - Also sensors and controllers

Double Buffer

- Use two system buffers instead of one
- A process can transfer data to or from one buffer while the operating system empties or fills the other buffer

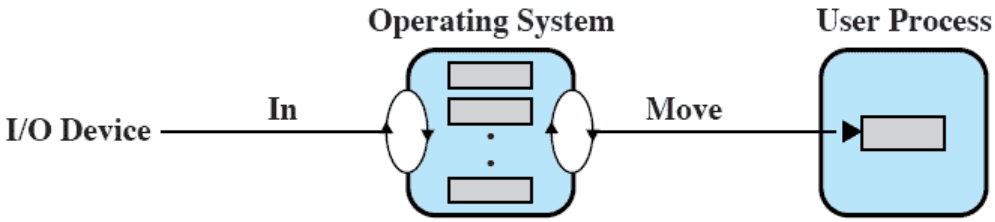


(c) Double buffering

Double buffering may be inadequate if the process performs rapid bursts of I/O.

Circular Buffer

- More than two buffers are used
- Each individual buffer is one unit in a circular buffer
- Used when I/O operation must keep up with process



(d) Circular buffering

Buffer Limitations

- Buffering smoothes out peaks in I/O demand
  - But with enough demand, eventually all (output) buffers become full and their advantage is lost

- When there is a variety of I/O and process activities to service, buffering can increase the efficiency of the OS and the performance of individual processes

User-Space I/O Software

- Typically library procedure linked with user programs
  - System calls
  - Formatting of input and output (i.e., printf)
- Spooling systems
  - Spooling is a way of dealing with dedicated I/O devices in a multiprogramming system (printer as example)
  - A process first generates the entire file to be printed and puts it in the spooling directory.
  - A daemon, which is the only process having permission to use the printer's special file, determines when to print the files in the directory.

Summary: I/O Software Design

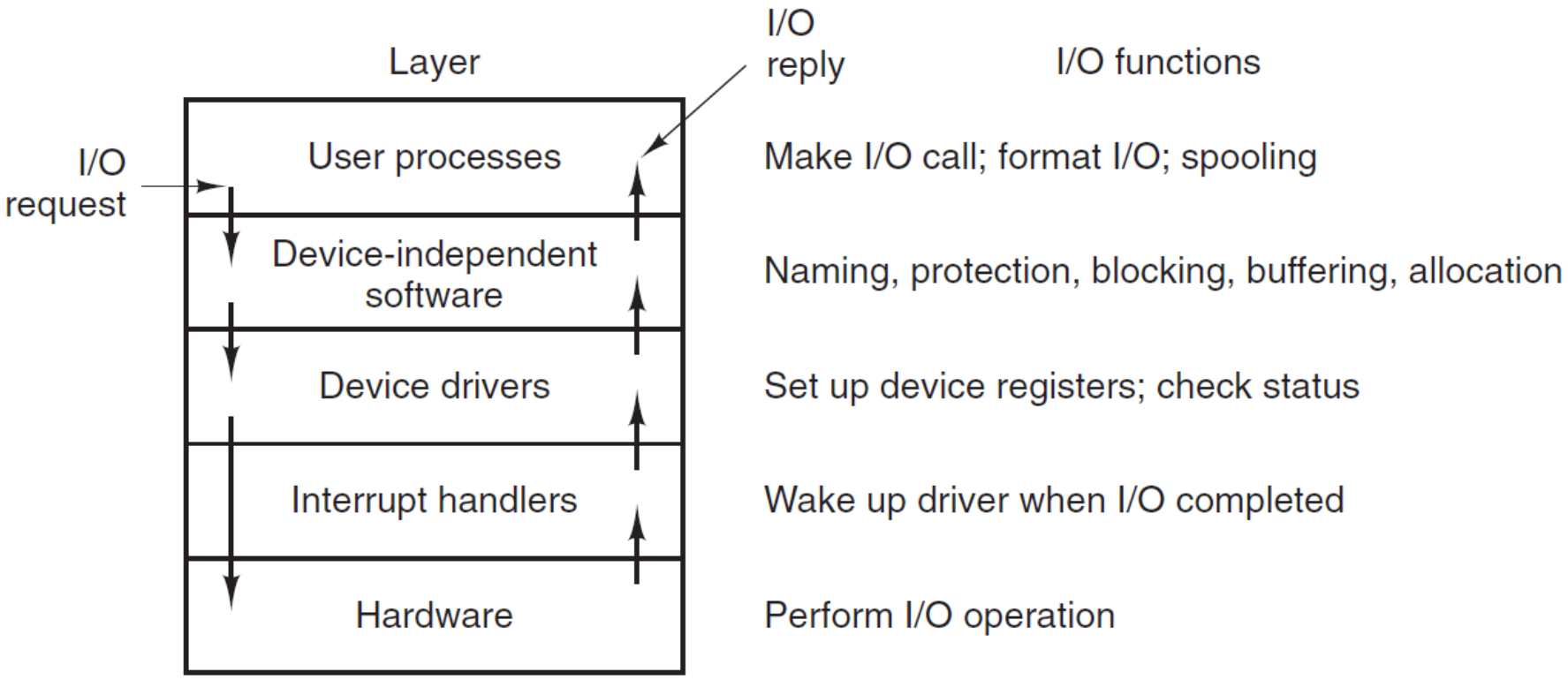


Figure 5-17. Layers of the I/O system and the main functions of each layer.

This Lecture

I/O

I/O Devices

I/O Software Design

I/O Examples

RAID

Devices are Files

- Each I/O device is associated with a special file
  - Managed by the file system
  - Provides a clean uniform interface to users and processes.

- To access a device, read and write requests are made for the special file associated with the device

UNIX SVR4 I/O

- Each individual device is associated with a special file

- Two types of I/O
  - Unbuffered
  - Buffered

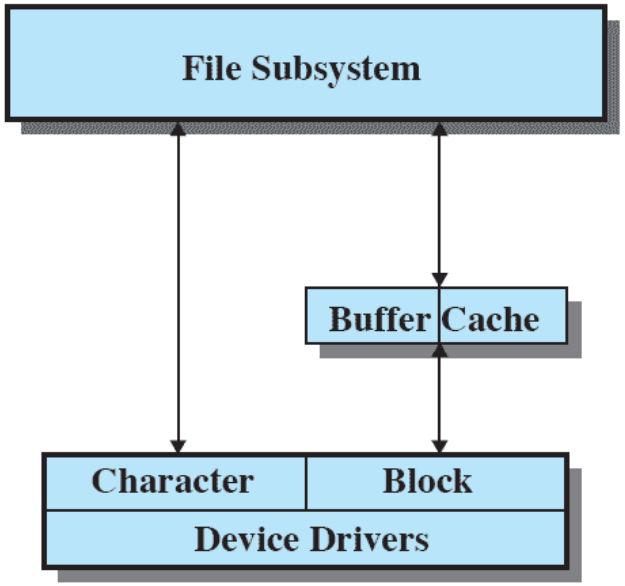


Figure 11.12 UNIX I/O Structure

Unbuffered I/O

- Unbuffered I/O is simply DMA between device and process
  - Fastest method
  - Process is locked in main memory and can't be swapped out
  - Device is tied to process and unavailable for other processes

Buffer Cache

- Essentially a disk cache
  - Buffer in main memory for disk sectors
    - Contains a copy of some of the sectors on the disk
  - When an I/O request is made for a particular sector
    - a check is made to determine if the sector is in the disk cache
- Three lists are maintained
  - Free List
  - Device List
  - Driver I/O Queue

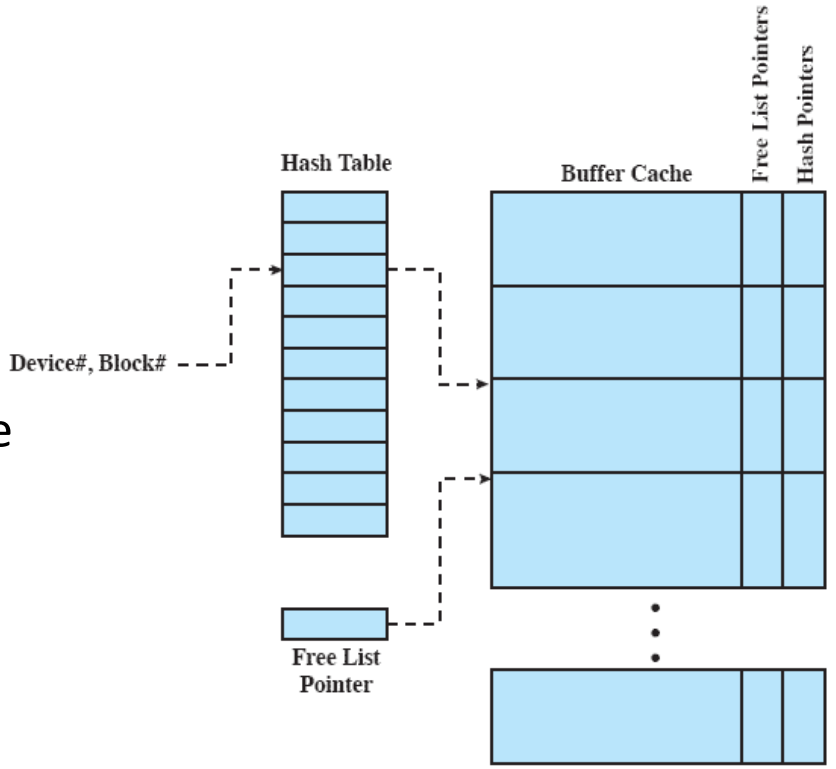


Figure 11.13 UNIX Buffer Cache Organization

Character Queue

- Used by character oriented devices
  - E.g. terminals and printers
- Either written by the I/O device and read by the process or vice versa
  - Producer/consumer model used
- Any differences with the buffer cache mechanism?
  - Typically can be read only once

I/O for Device Types

Table 11.5 Device I/O in UNIX

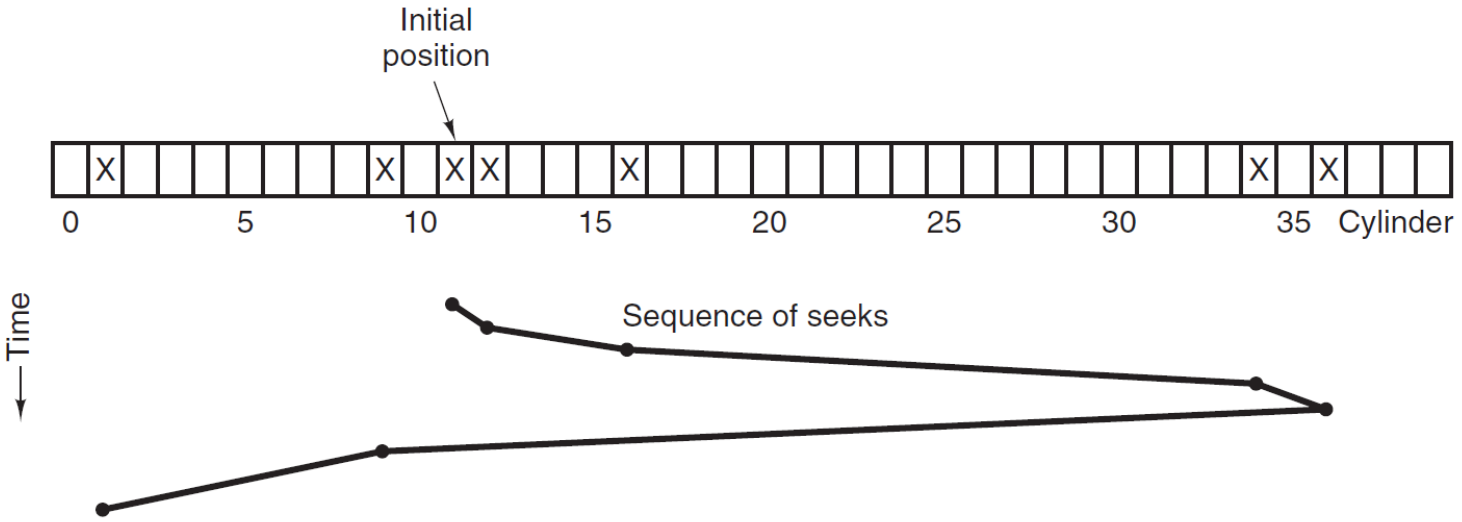
	Unbuffered I/O	Buffer Cache	Character Queue
Disk drive	X	X	
Tape drive	X	X	
Terminals			X
Communication lines			X
Printers	X		X

Linux/Unix Similarities

- Linux and Unix (e.g. SVR4) are very similar in I/O terms
  - The Linux kernel associates a special file with each I/O device driver
  - Block, character, and network devices are recognized

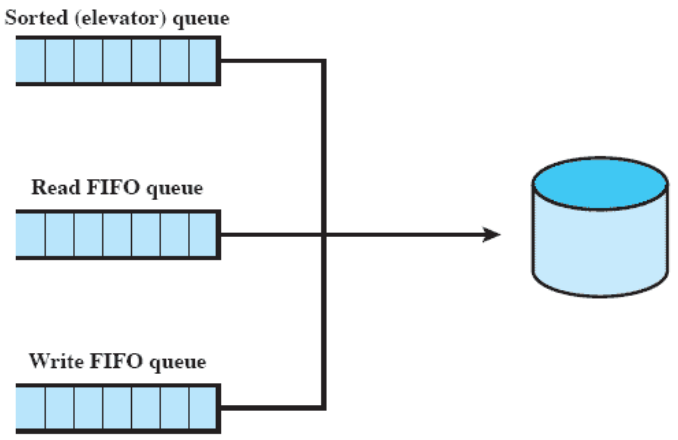
The Elevator Scheduler

- The **Elevator Scheduler** is used to improve disk arm scheduling
  - Maintains a single queue for disk read and write requests
  - Keeps list of requests sorted by block number
  - Drive moves in a single direction to satisfy each request
- Example: comparing to FCFS or SSF (Shorted Seek Frist)



Deadline Scheduler

- Uses three queues
  - Incoming requests
  - Read requests go to the tail of a FIFO queue
  - Write requests go to the tail of a FIFO queue



- Each request has an expiration time
- Ordinarily, the scheduler dispatches from the sorted queue.
  - When a request is satisfied, it is removed from the head of the sorted queue and also from the appropriate FIFO queue.
  - However, when the item at the head of one of the FIFO queues becomes older than its expiration time, then the scheduler next dispatches from that FIFO queue, taking the expired request, plus the next few requests from the queue.

Anticipatory I/O scheduler

- Elevator and deadline scheduling can be counterproductive if there are numerous synchronous read requests
- When a read request is dispatched, **the anticipatory scheduler** causes the scheduling system to delay for up to 6 milliseconds, depending on the configuration.
- During this small delay, there is a good chance (**principal of locality**) that the application that issued the last read request will issue another read request to the same region of the disk.
  - If so, that request will be serviced immediately.
  - If no such read request occurs, the scheduler resumes using the deadline scheduling algorithm.

Linux Page Cache

- In Linux 2.2 and earlier, the kernel maintained a page cache for reads and writes from regular file system files and for virtual memory pages, and a separate buffer cache for block I/O.

- In Linux 2.4 and later, a single unified page cache for all traffic between disk and main memory

- Benefits of page cache:
  - Dirty pages can be collected and written out efficiently
  - Pages in the page cache are likely to be referenced again due to temporal locality

Windows I/O Manager

- The I/O manager is responsible for all I/O for the operating system
- It provides a uniform interface that all types of drivers can call

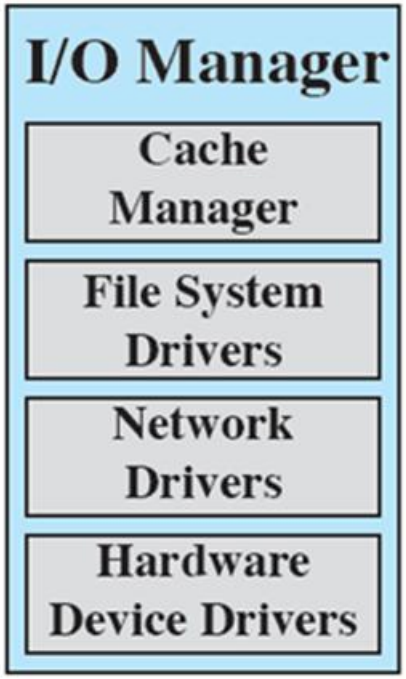


Figure 11.15 Windows I/O Manager

Windows I/O

- The I/O manager works closely with:
  - Cache manager – handles all file caching
    - A kernel thread, the lazy writer, periodically batches the updates together to write to disk
  - File system drivers - routes I/O requests for file system volumes to the appropriate software driver for that volume
  - Network drivers - implemented as software drivers rather than part of the Windows Executive.
  - Hardware device drivers

Asynchronous and Synchronous I/O

- Windows offers two modes of I/O operation:
  - asynchronous and synchronous.

- Asynchronous mode is used whenever possible to optimize application performance

Software RAID

- Windows implements RAID functionality as part of the operating system and can be used with any set of multiple disks.
  - RAID 0, 1 and RAID 5 are supported.

- In the case of RAID 1 (disk mirroring), the two disks containing the primary and mirrored partitions may be on the same disk controller or different disk controllers

Volume Shadow Copies

- Shadow copies are implemented by a software driver that makes copies of data on the volume before it is overwritten
  - From Windows XP
- Designed as an efficient way of making consistent snapshots of volumes to that they can be backed up.
  - Also useful for archiving files on a per-volume basis
- Shadow copies are implemented by a software driver that makes copies of data on the volume before it is overwritten.

This Lecture

I/O

I/O Devices

I/O Software Design

I/O Examples

RAID

Multiple Disks

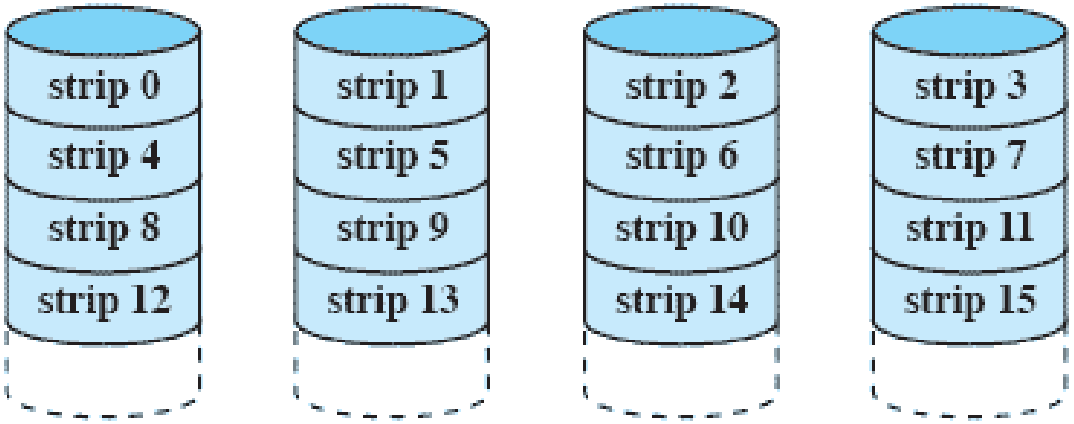
- Disk I/O performance may be increased by spreading the operation over multiple read/write heads
  - Or multiple disks
- **Disk failures** can be recovered if parity information is stored

RAID

- RAID: Redundant Array of Independent Disks
  - Set of physical disk drives viewed by the operating system as a single logical drive
  - Original paper: Patterson, David; Gibson, Garth A.; Katz, Randy. A Case for Redundant Arrays of Inexpensive Disks (RAID). SIGMOD 1988.
- Data are distributed across the physical drives of an array
- **Redundant** disk capacity is used to store **parity** information which provides recoverability from disk failure

RAID o - Stripped

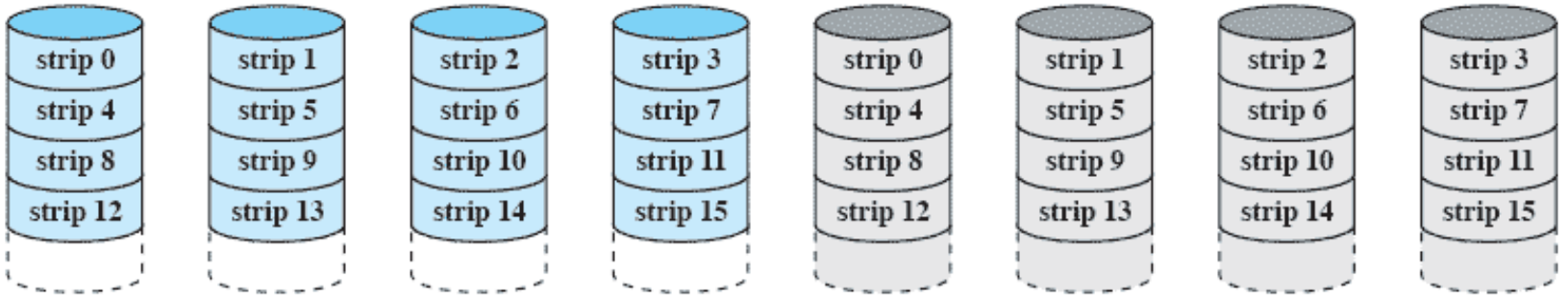
- Not a true RAID – no redundancy
- Disk failure is catastrophic
- Very fast due to **parallel read/write**



(a) RAID 0 (non-redundant)

RAID 1 - Mirrored

- Redundancy through **duplication** instead of parity
- A read request can be serviced by either of the two disks
  - Choose the one involving the minimum seek time plus rotational latency.
- A write request requires that both corresponding strips be updated, but this can be done in parallel.
  - Thus, the write performance is dictated by the slower of the two writes
- Simple recovery from one disk failure

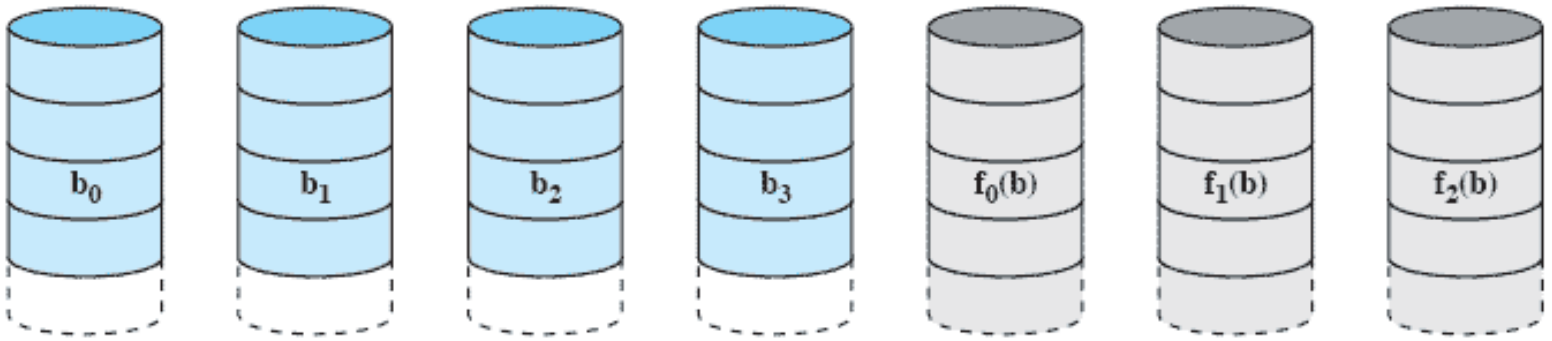


(b) RAID 1 (mirrored)

Actually, this figure depicts RAID 0+1, instead of RAID 1.

RAID 2 (Using Hamming Code)

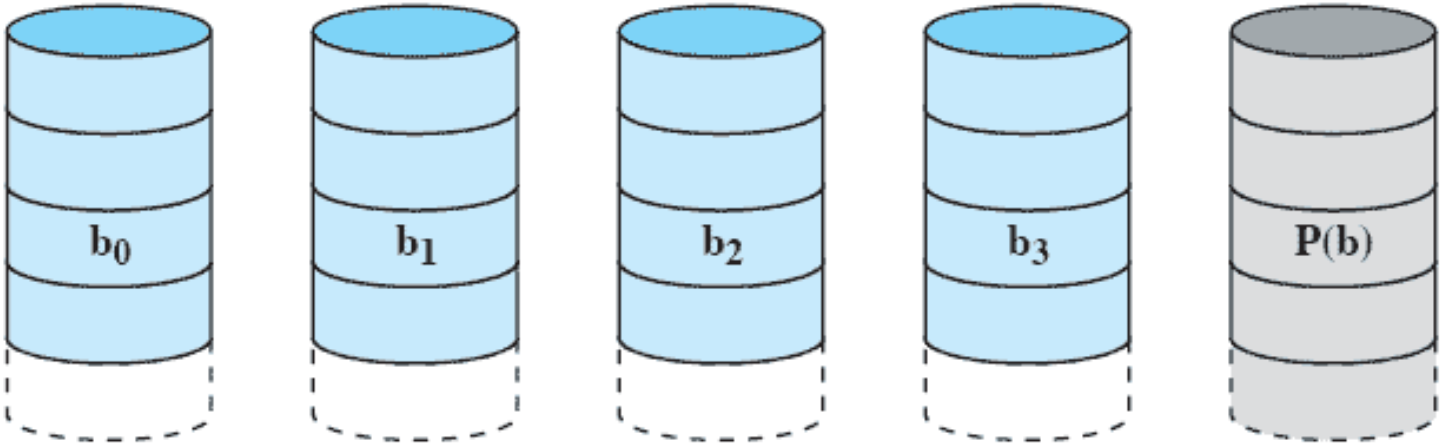
- Synchronized disk rotation
- Data striping is used (extremely small)
  - often as small as a single byte or word.
- **Hamming code** used to correct single bit errors and detect double-bit errors



(c) RAID 2 (redundancy through Hamming code)

RAID 3 Bit-interleaved Parity

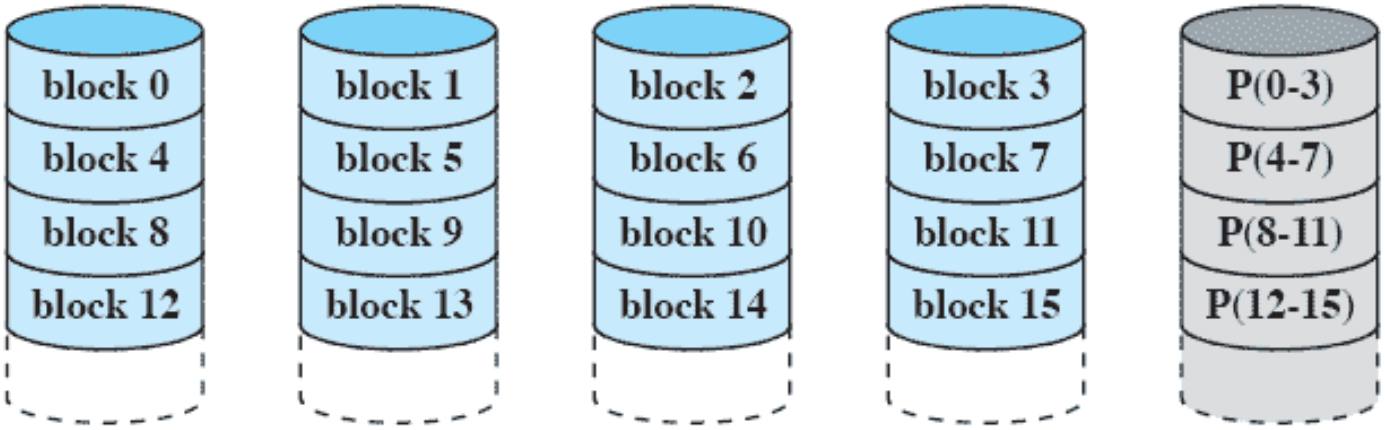
- Similar to RAID-2 but all parity bits are stored on a single drive
- RAID 3 employs parallel access, with data distributed in small strips.
  - Instead of an error-correcting code, a simple parity bit is computed for the set of individual bits in the same position on all of the data disks.



(d) RAID 3 (bit-interleaved parity)

RAID 4 Block-level Parity

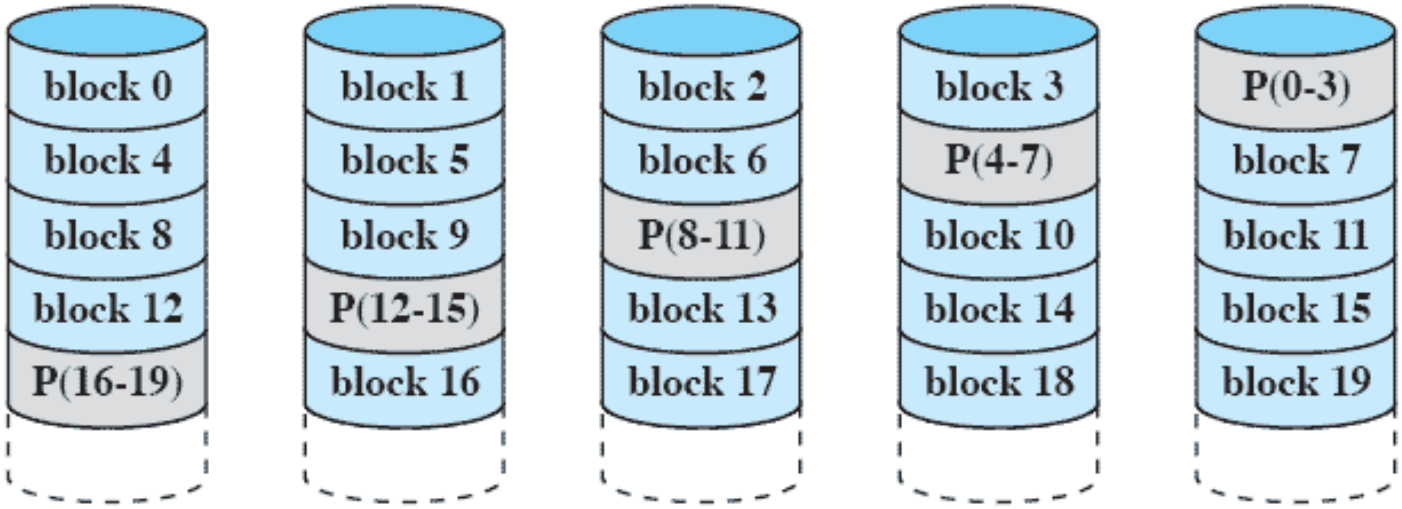
- A bit-by-bit parity strip is calculated across corresponding strips on each data disk
- The parity bits are stored in the corresponding strip on the parity disk



(e) RAID 4 (block-level parity)

RAID 5 Block-level Distributed parity

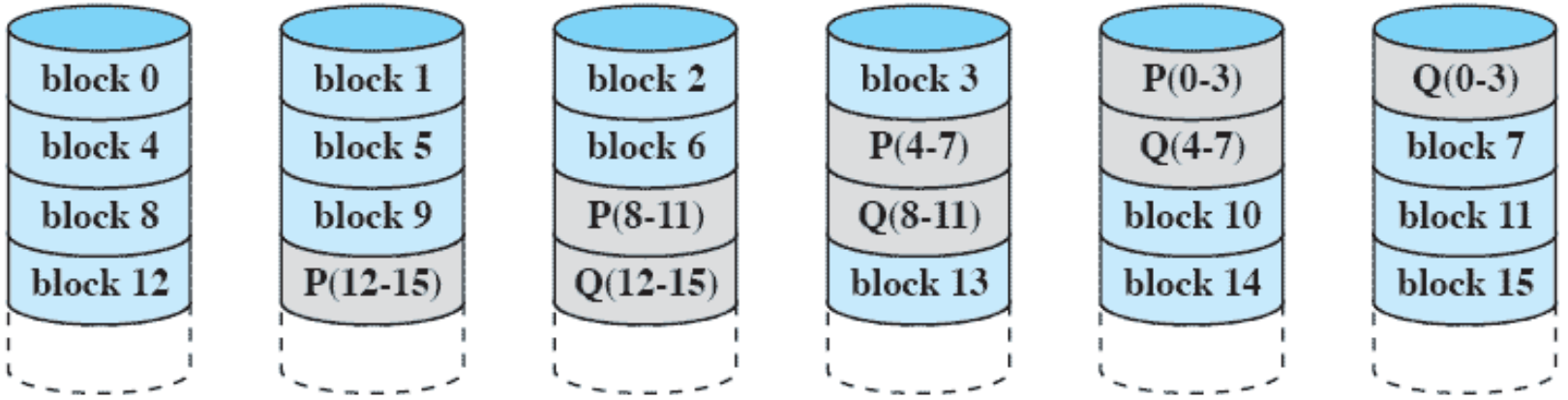
- Similar to RAID-4 but distributing the parity bits across all drives
  - A typical allocation is a round-robin scheme
  - For an n-disk array, the parity strip is on a different disk for the first n stripes, and the pattern then repeats.



(f) RAID 5 (block-level distributed parity)

RAID 6 Dual Redundancy

- Two different parity calculations are carried out
  - stored in separate blocks on different disks.
  - Can recover from two disks failing
- P and Q are two different data check algorithms.
  - One of the two is the exclusive-OR calculation used in RAID 4 and 5.
  - The other is an independent data check algorithm.

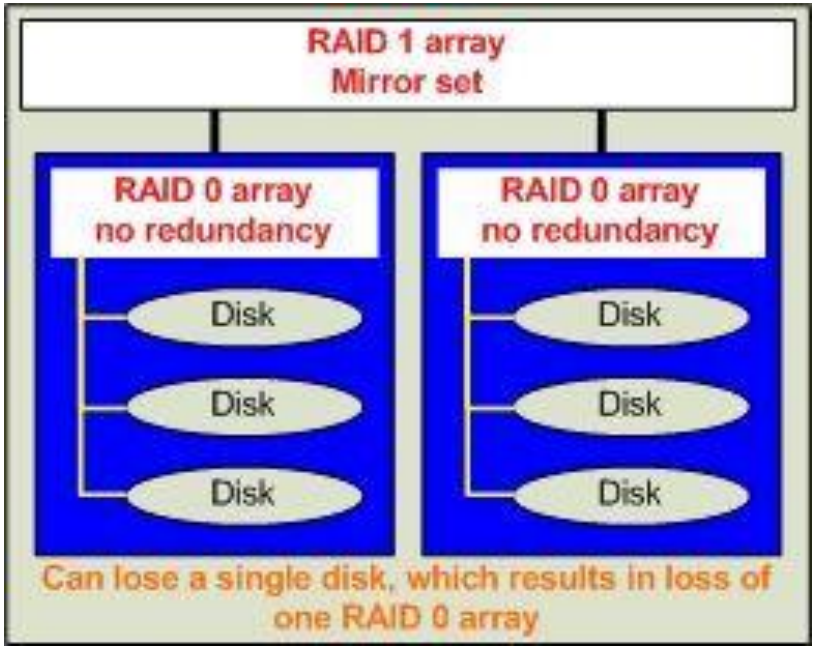


(g) RAID 6 (dual redundancy)

Multiple-Level RAID

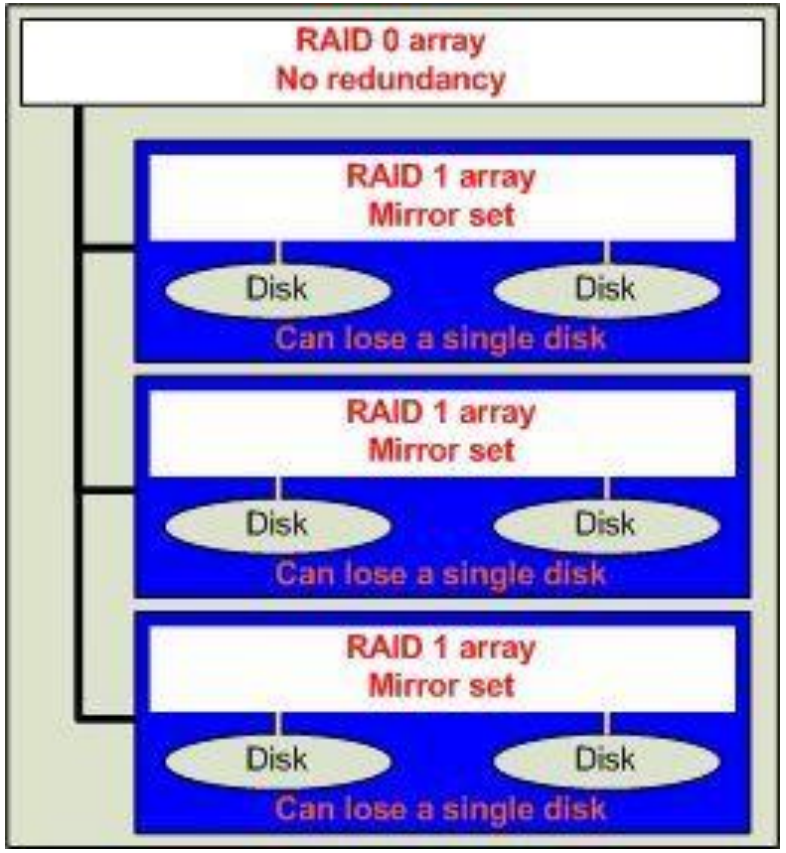
- We can combine different RAID mechanisms to achieve different goals within one system

- RAID 01 (RAID 0+1)
  - RAID 0 then RAID 1
  - A mirror (RAID 1) of a stripe set (RAID 0)
  - Minimum drive required?
    - Four
  - Pros?
  - Cons?

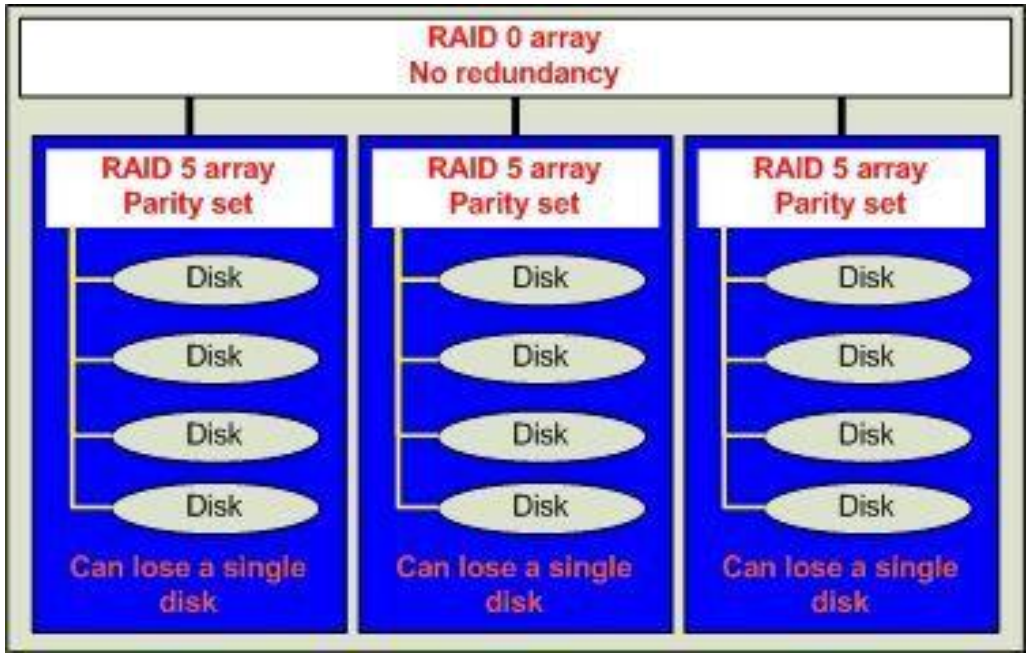


Multiple-Level RAID (cont.)

- RAID 10 (RAID 1+0)
  - RAID 1 then RAID 0
  - a stripe (RAID 0) of multiple mirror sets (RAID 1).
  - Minimum drive required?
    - Four
  - Pros?
  - Cons?



Multiple-Level RAID (cont.)



- RAID 50 (RAID 5+0, or RAID 5 then RAID 0)
  - Stripe of Parity Set
  - Minimum drive required?
    - Six

Summary

- I/O Devices
- I/O Software Design
- I/O Examples
- RAID

- Next lecture
  - Distributed systems