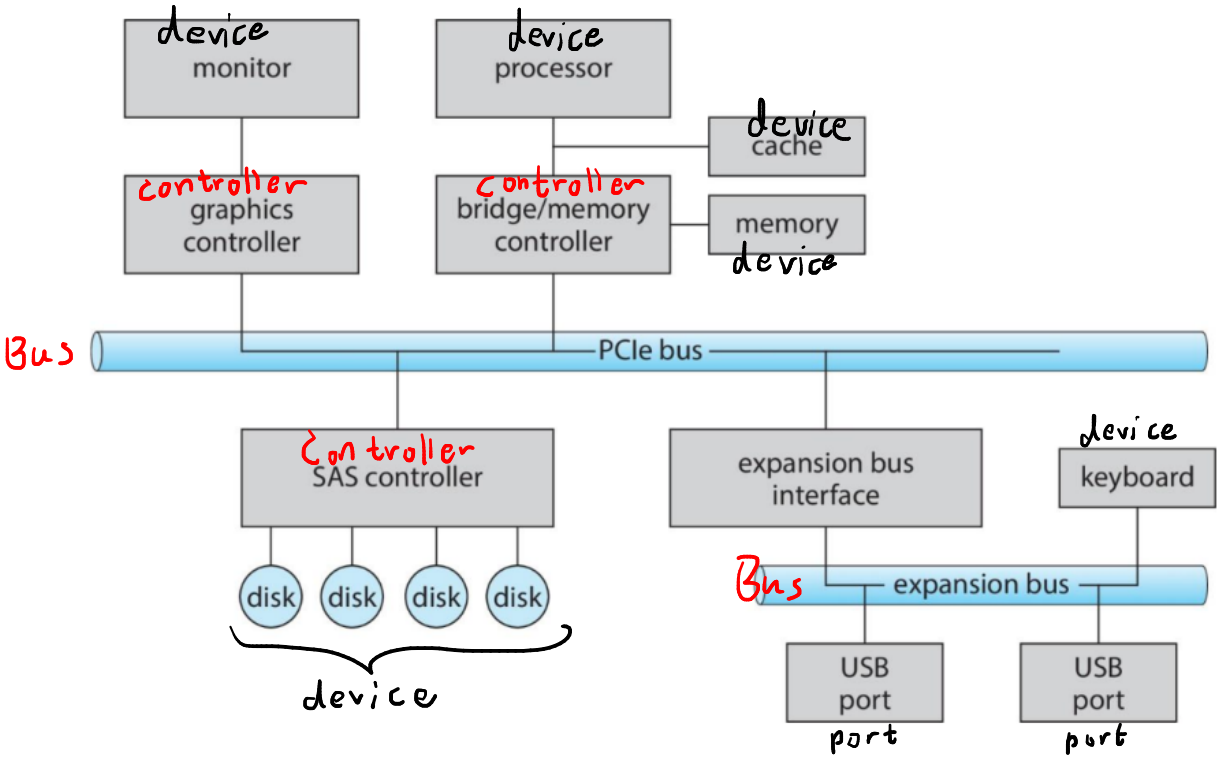
# I/O Systems

ประกอบด้วย

- Storage ( HDD, SSD, flash Drive)
- Transmission ( LAN, Wi-fi, Bluetooth)
- Human - Interface ( Printer, Mouse, keyboards)

## Concept หลัก ของ I/O

- Port
- Bus → PCIe
- Controller → อุปกรณ์ควบคุม port, bus



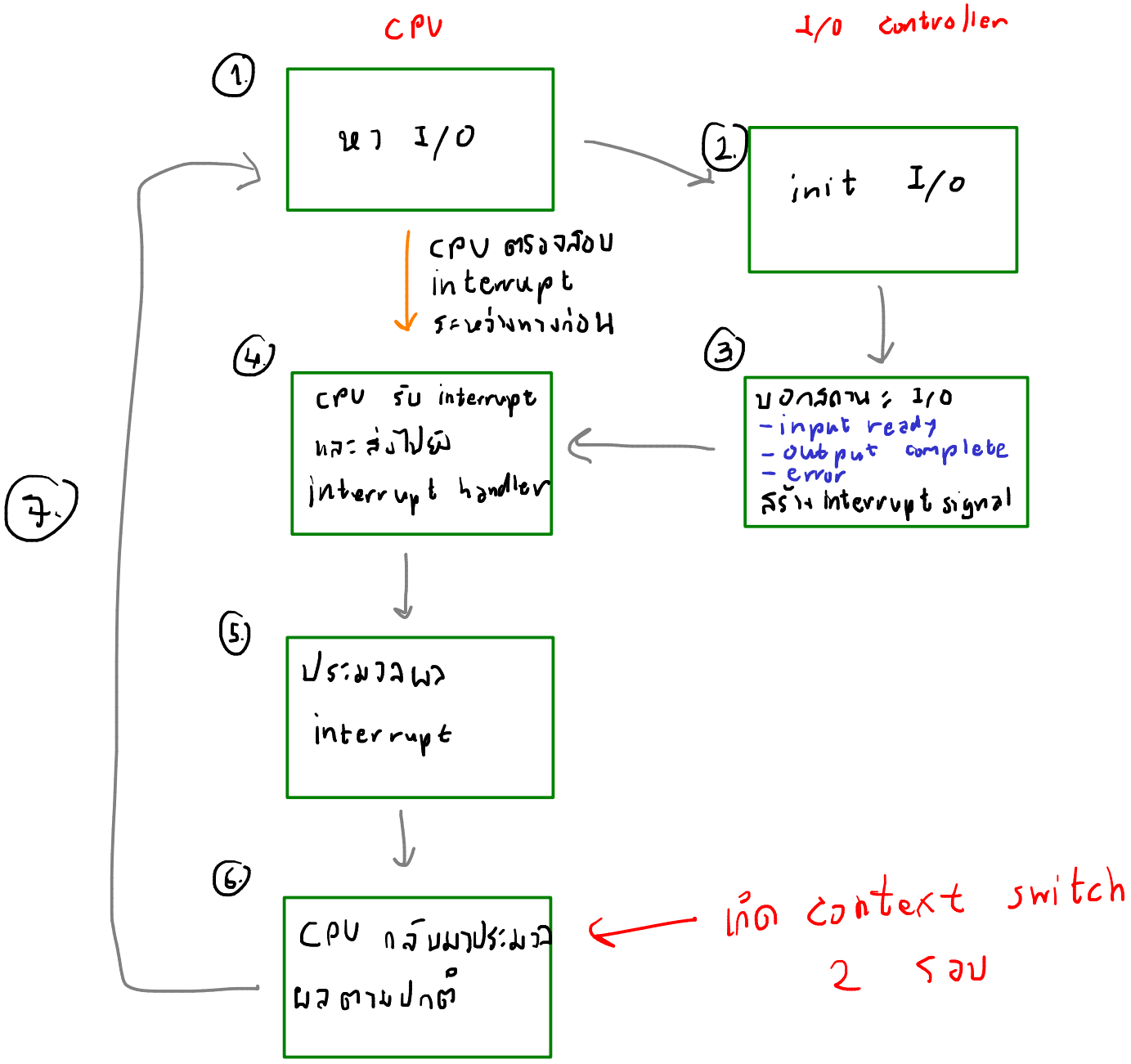

## Polling

1.) อ่าน busy-bit จาก status register จนเป็น 0
2.) Host กำหนด ว่า read/write
3.) Host set → ready-bit
4.) controller set → busy-bit และ execute transfer
5.) controller clear → busy-bit, error-bit, command-ready bit เมื่อ transfer-done

หลังการเรียก I/O → Interrupts

- Polling - สามารถเกิดขึ้นได้ใน 3 instruction cycle
  → อ่าน status-bit
- CPU Interrupt-request line ← จาก I/O
- Interrupt Handler (ใช้รับ Interrupt)
- Interrupt Vector (ใช้ประมวลผลให้ handler ทำงานถูกต้อง)
  - context switch
  - priority

Interrupt-Driven I/O cycle




← เกิด context switch 2 รอบ

- การทำ Interrupt ยังใช้สำหรับ exception
  - Page Fault
  - trap
  - Multi-CPU สามารถรับ interrupt พร้อมกันได้ (ถ้า OS รองรับ)
  - ต้องเร็ว

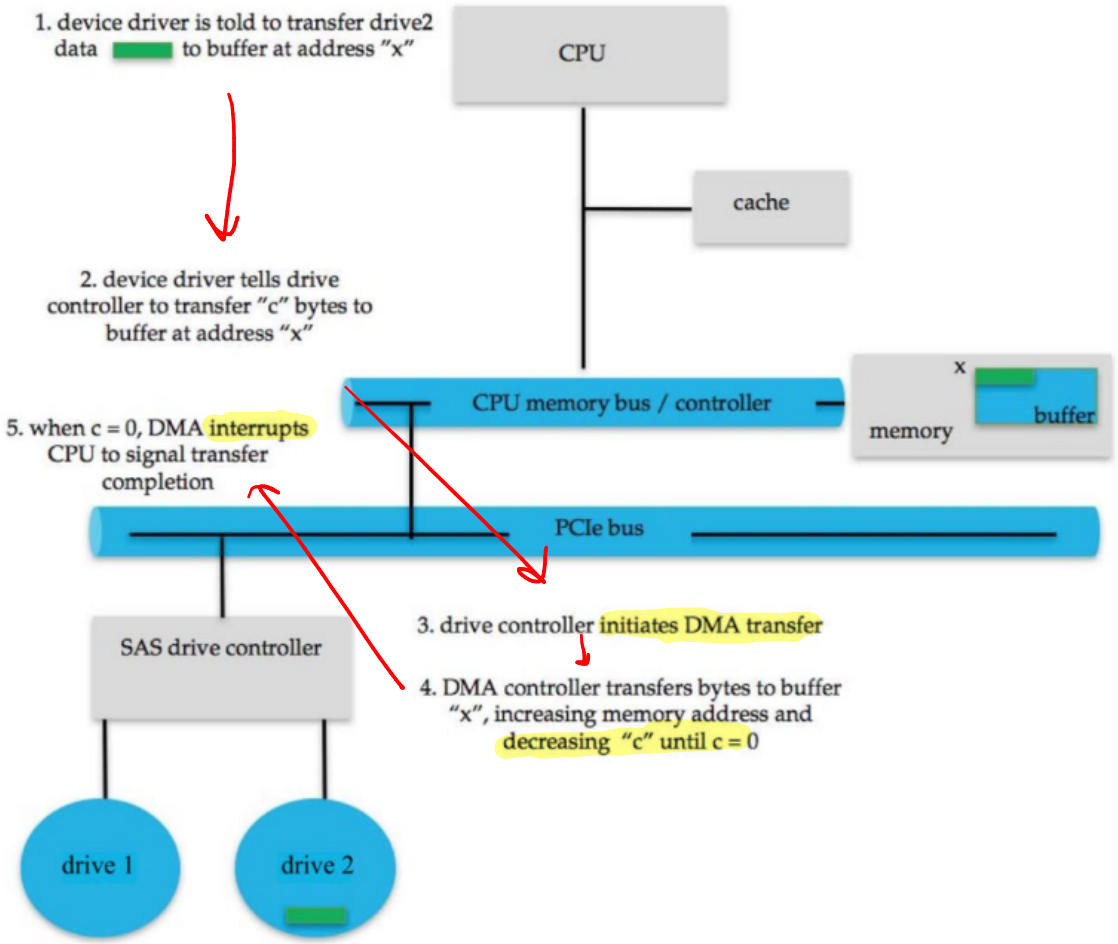
โดยปกติ OS ใน desktop จะเกิด interrupt หลักร้อย / วินาที
Server จะเกิด interrupt หลักพัน / วินาที

---

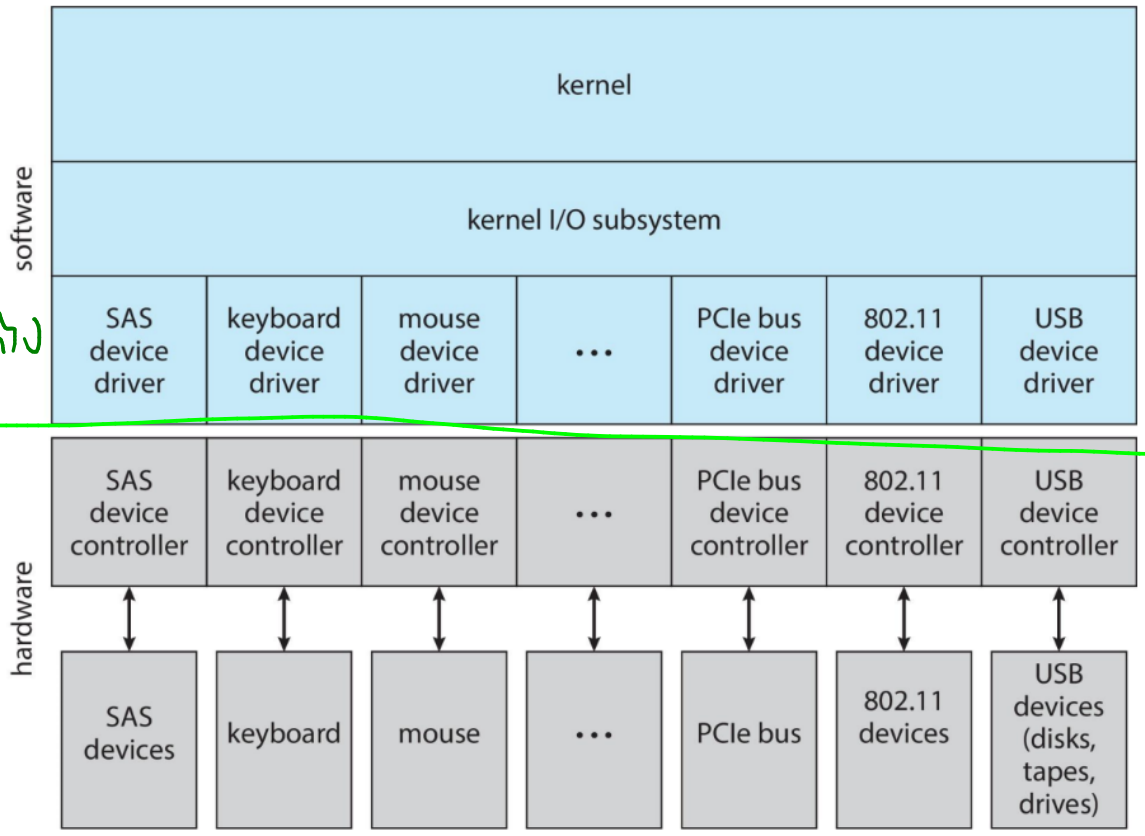
Direct Memory Access
- ต้องมี hardware controller
- ใช้สำหรับ I/O <-คุยกับ-> memory (RAM) โดยตรง
- โดย OS จะเขียน Command ต่างๆ ลงไปยัง memory
  - src → dest address
  - R / W mode
  - นับจำนวน bytes
  - ตำแหน่งที่จะเขียนลงไปยัง DMA controller
  - Bus Mastering (steal CPU cycle)
  - เมื่อคำสั่งใน DMA ทำงานเสร็จสิ้น → ส่ง interrupt

* ถ้าเป็น virtual address จะเรียกว่า DVMA

ขั้นตอนการทำงานของ DMA




Application I/O Interface




OS

- แตกต่างที่ของกัน

HW

ลักษณะของ I/O devices

| aspect | variation | ex. |
| --- | --- | --- |
| - data - transfer mode | char, block | terminal, disk |
| - access method | seq, rand | modem, CD |
| - transfer schedule | sync, async | tape, kbd |
| - sharing | dedicated, sharable | tape, kbd |
| - device speed | latency, seek time | HDD, SSD |
| - I/O direction | R, W, R/W | CD, GPU, disk |

OS จะ Group I/O เป็น 4 ประเภท

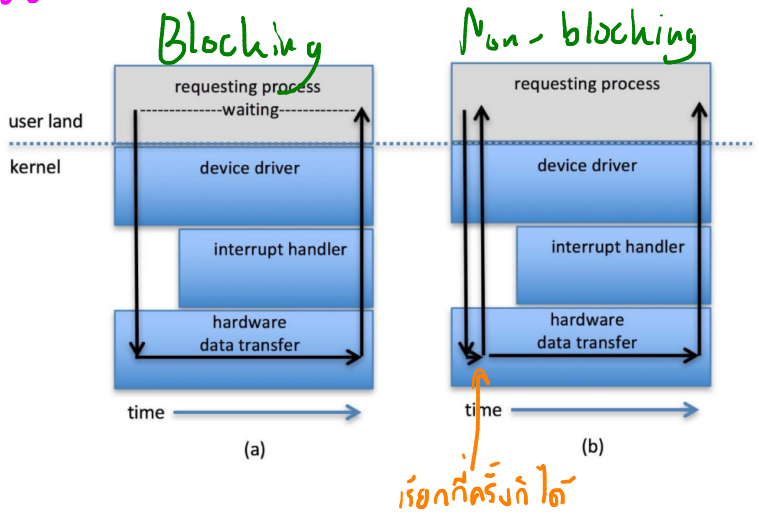
- Block I/O
  - disk drive
  - read, write, seek
  - DMA - อาจจะมี
- Character I/O
  - keyboard, mouse
  - getc(), putc()
- Memory-Mapped File access
- Network Socket
  - ใช้ socket interface
  - select()
  - สามารถสร้างได้หลายแบบ < pipes, FIFOs, streams, queues, mailboxes

Clock & Timers

- มี current time
  elapsed time
  timer

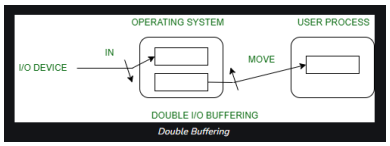
ioctl()

- Blocking I/O → process หยุดทำงานจนกว่า I/O เสร็จ
- Non-Blocking I/O → I/O เรียกเท่าไหร่ก็ได้
  - ใช้ใน multi-thread
- Asynchronous → process ทำงานระหว่าง I/O ทำงานไปด้วย
  - ใช้ยาก

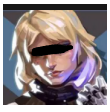



Vectored I/O
- ช่วยทำให้ 1 system call เรียกหลายๆ I/O ขึ้นมาได้
- ช่วยลด context switching และจำนวน System call ที่มากเกินไป

Kernel I/O Subsystem
- Scheduling
- Buffering - ใช้เก็บ data ใน memory ในขณะที่ส่งข้อมูลระหว่างอุปกรณ์
  - cope speed / transfer size ที่แตกต่างกัน
  - Double Buffering




- Caching - ใช้ในการ copy [เร็ว]
- Spooling - ใช้ hold output ex. printer
- Device Reservation - ให้ vip ในการเข้าถึง device
  ↓
  ทำให้เกิด deadlock ได้



## Error Handling

- OS สามารถ Recover จากความผิดพลาดของ I/O ได้

  เช่น - Retry

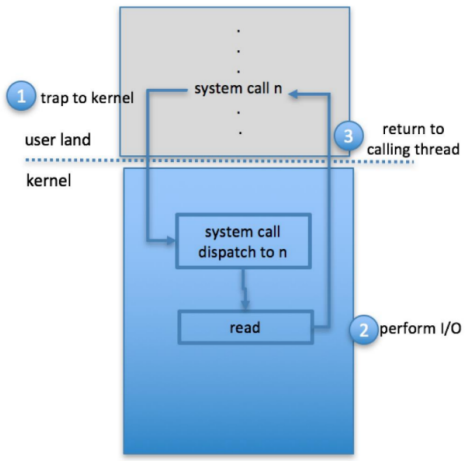
  - Return Error Code

  - บันทึกลง Log

## I/O protection

- I/O ต้องเรียกผ่าน system calls เท่านั้น

  (user ไม่สามารถทำได้)




## Kernel Data Structures

- ใช้เก็บ state ของ I/O
- มีหลายรูปแบบในการเก็บ
- ex. Windows ใช้ message passing

## Power Management

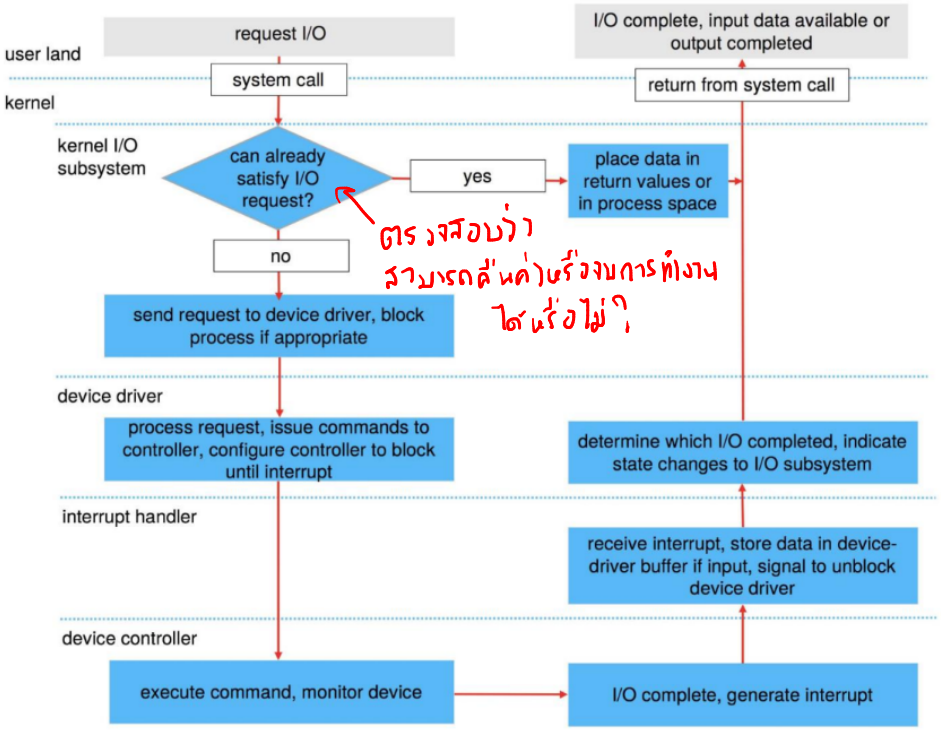
- ใช้ cooling ในการลดความร้อน
- ใน mobile device เป็น OS First-class aspect

ex. Android

- สร้าง tree
- อันไหนไม่ได้ใช้ → ปิด
- ถ้าใน tree ไม่ได้ใช้หมดแล้ว → ปิดหมด

- ปัจจุบันมี ACPI (Advance Configuration and power interface)

# Life Cycle of An I/O Request




ตรวจสอบว่า สามารถตอบสนองการทำงานได้เร็วไม่?

Performance

- I/O เป็นส่วนหลักของความเร็วในระบบ
  - CPU execute driver, kernel I/O code
  - จำนวน context switching
  - Data Copying
  - โดยเฉพาะ: Network traffic

Performance Improving

- ลดจำนวน context switches
- ลดการ copy data
- ลด interrupts ด้วยการ transfer ไฟล์ขนาดใหญ่, ควบคุม controller
- ใช้ DMA
- ใช้ HW ใหม่ๆ
- Balance Spec ให้ได้ throughput ที่เหมาะที่สุด
- เอา user-mode process ไปยัง kernel threads /daemons

# File - System (Interface)

File F ประกอบด้วย

- Name
- Identifier
- Type
- Location
- Size
- Protection
- Datetime และ user identification
  - ข้อมูลในไฟล์

เป็นส่วนหนึ่งของ Directory Structure

เป็น node เก็บข้อมูลในทุกไฟล์

File Operation

- create
- Write
- Read
- Reposition
- Delete
- Truncate
- Open ($F_i$) เอา content โยกจาก disk ไป memory
- Close ($F_i$) โยกกลับเข้า disk

ในการเปิดไฟล์ จะมีการจัดการดังนี้ เพื่อเปิด File

- open file table
- file pointer
- file open count

File Locking

- Shared Lock → ใช้อ่าน
- Exclusive Lock → ใช้เขียน

ประเภทการ lock

- mandatory
  - บังคับ
- Advisory
  - แนะนำ

# File Structure

- none ← word, bytes
- simple record structure ← csv, \n \r
- complex structure ← xml, json

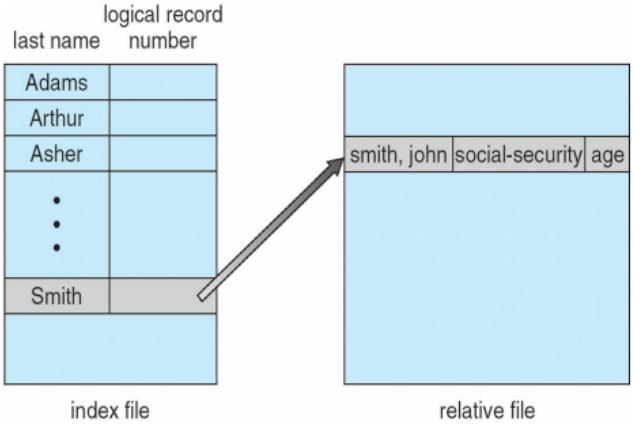
# Access Method

- Sequential Access (คล้ายอ่านเทป)
  - read next
  - write next
  - reset ← เริ่มใหม่
- Direct Access
  - read n
  - write n
  - position to n ← ไปที่ไหนก็ได้
    - read next
    - write next
    - rewrite n

* n = relative block number.

วิธีอ่านไฟล์แบบอื่นๆ

- Index and Relative file




(Database?)

# Disk Structure

- Disk สี่แบบ Partitions ได้
- RAID
- Disk อาจมี FS หรือ ไม่มีเลยก็ได้ (File System)

## ประเภทของ File System

Solaris

- tmpfs – memory-based volatile FS for fast, temporary I/O
- objfs – interface into kernel memory to get kernel symbols for debugging
- ctfs – contract file system for managing daemons
- lofs – loopback file system allows one FS to be accessed in place of another
- procfs – kernel interface to process structures
- ufs, zfs – general purpose file systems

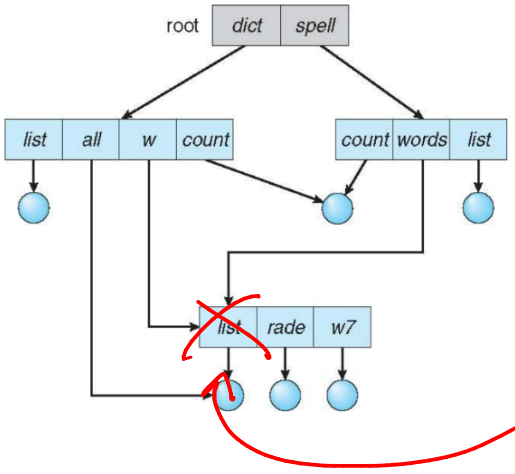
## คำสั่งที่สามารถใช้ใน directory

- หา File (find)
- สร้าง (create)
- ลบ (delete)
- list
- rename
- traverse

## การจัดการ Directory

- Single-level directory → ทุก user ใช้ directory เดียว
  → มีปัญหา Naming, Grouping
- Two-level directory → 1 user 1 directory
  → search เร็วขึ้น
  - มี path name
  แต่ ยังไม่มี grouping
- Tree-Structure directory
- Acyclic-Graph Directories – มี file และ subdirectory ที่ shared ร่วมกัน

# Acyclic - Graph Directories (เพิ่มเติม)




- มี aliasing ( dict/w/list
spell/words/list)

↓

ถ้าลบ list ออกไป ex. ลบ w/list
จะทำให้เกิด dangling pointer

แก้ได้โดยทำ Back pointer
แล้วสั่งลบ pointers ทีหมด

→ เกิด directory entry type ใหม่

→ Link ← เหมือน shortcut ไปยังไฟล์ที่มีอยู่

[ Resolve the Link ใช้ในการ follow pointer
ไปยังไฟล์

# General Graph Directory

→ จะรู้ได้ยังไงว่าไม่มี cycle ใน Graph แน่ๆ

Allow → Link File ได้อย่างเดียว

→ Garbage Collection

→ เมื่อมีการ link ให้ run cycle detection algo.

# Current Directory ทำอะไรได้บ้าง

→ pwd

→ create / delete a file
touch rm

# Protection

→ R W X
Read Write Excute

→ Append, delete, list

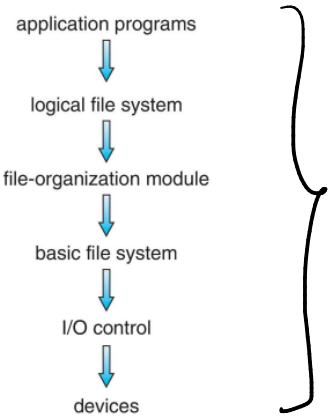
binary 3 bit ←

| | Owner | Group | Other |
|---|---|---|---|
| | rwx | r-x | r-x |
| | 4+2+1 | 4+0+1 | 4+0+1 |
| | 7 | 5 | 5 |

* อย่า 777

# File-System (Implementation)

- File System Structure
  - ทำใน secondary storage
    - I/O จะ perform เป็น blocks of sectors
  - ช่วย mapping ของจาก logical → physical
  - จัดการในเรื่องของการเข้าถึง data, การเก็บ และ การค้นหา
- Driver ใช้ควบคุม physical device.
- File System ช่วยจัดการไฟล์ให้เป็น layer




Layered File System.

## File System Layer

- Device Driver. - จัดการ I/O ที่ I/O control layer
- Basic File System - เป็นภาษาระดับสูงก่อนแปลงไปเป็นภาษา driver
  - นอกจากนี้ยังใช้จัดการ caches, buffers
- File Organize Module - ใช้ทำความเข้าใจ file, logical address และ physical block
  - ใช้แปลง logical block # → physical block #
  - จัดการ free space, disk allocation
- Logical File System - ใช้จัดการ metadata ของไฟล์
  - แปลงชื่อไฟล์ → file number, File handle, location ผ่านการ control block
  - จัดการ directory
  - การป้องกัน (protection)

การแบ่งระบบ File System ออกเป็น layer ช่วยลดความซับซ้อน แต่ทำให้ความเร็วลดลง

* Logical Layer สามารถ implement ด้วยภาษาอะไรก็ได้

ใน OS มี File System เช่น

- CD-ROM ใช้ ISO 9660
- UNIX ใช้ UFS
- Windows ใช้ NTFS
- Linux มีมากกว่า 130 ชนิด เช่น ext3

File System Operation

- Boot Control Block
  - ใช้กับ volume ที่มี OS เพื่อ boot ระบบขึ้นมา
- Volume Control Block
  - เก็บ volume detail → เช่น จำนวน blocks, free blocks, block size, free block pointers หรือ array.

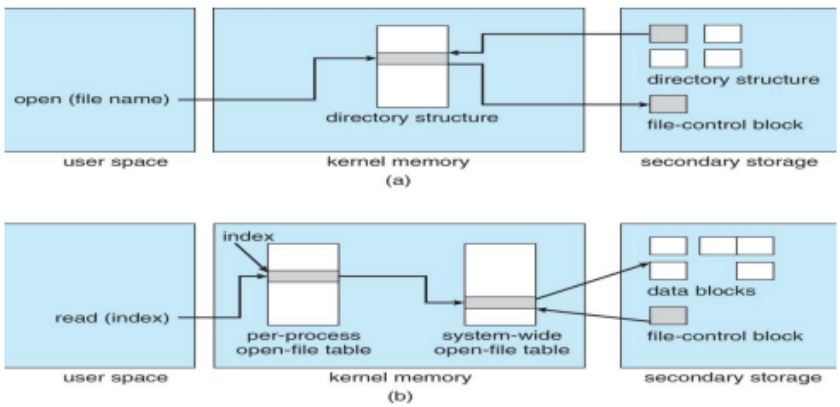
File Control Block (FCB)

- ใช้เก็บรายละเอียดของไฟล์ (FCB per File)

In-Memory File System Structures

- Mount Table - เก็บ file system mounts, mount points, file system types.
- System-wide open-file Table
- Per-process open-file Table

- Figure 12-3(a) refers to opening a file
- Figure 12-3(b) refers to reading a file

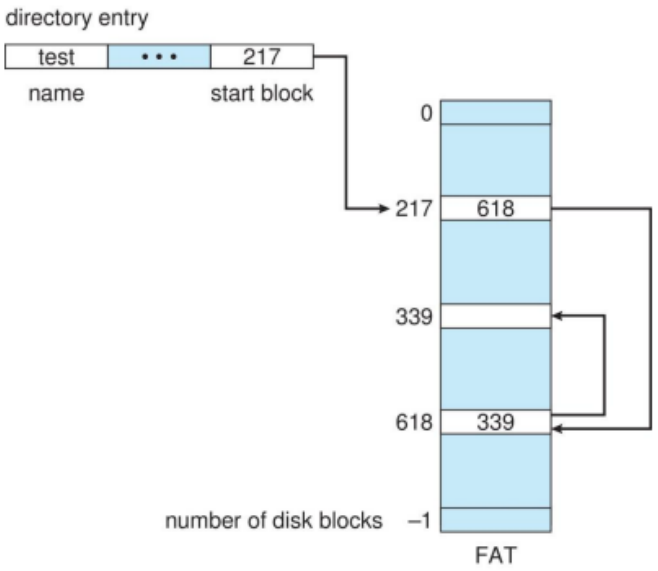


# Directory Implementation

- Linear List - ใช้เวลาหานาน (linear search)
- Hash Table - มีการ collision เมื่อชื่อไฟล์ซ้ำกัน

# Allocation Method

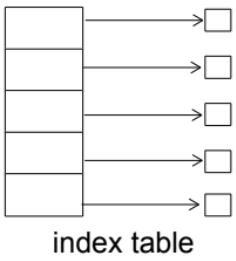
- Contiguous (ต่อเนื่องเป็นชิ้นเดียว) displacement : R
  - เร็วสุด
  - เรียบง่าย
  - ปัญหาที่พบ → - ต้องหาพื้นที่เพียงพอสำหรับ file นั้น
    - ต้องรู้ขนาด file
    - เกิด external fragmentation
  - * บาง OS เอาพาร์ติชันมีขอบเขต โดย 1 file อาจมีหลาย extents
- Linked displacement : R + 1
  - เก็บกระจายๆ แล้วเชื่อมเป็น linked-list 1/2 blocks
  - ไม่มี external fragmentation (มี internal fragmentation)
  - แต่ละ block จะมี pointer ไปยัง block ต่อไป
  - การค้นหาทำให้เสียเวลามากขึ้น (I/O, disk seek)
  - มีปัญหาเรื่อง Reliability
- File Allocation Table (FAT)

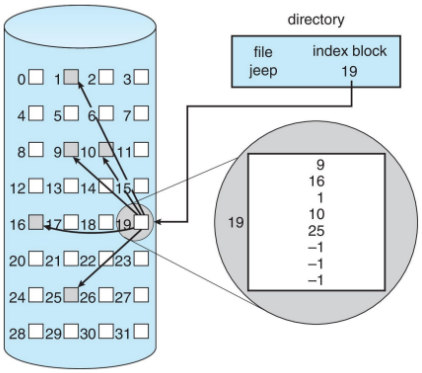


▷ Indexed Allocation Method

- แต่ละไฟล์จะมี index block of pointers ชี้ไปยัง data blocks

- Logical view




## Performance

- Contiguous เหมาะสำหรับ Sequential , random
- Linked เหมาะสำหรับ sequential
- Indexed จะซับซ้อนกว่า เพราะว่า
  - การเข้าถึง block จะต้องการ 2 index block เพราะ 1 block สำหรับเก็บ index ของข้อมูล
  - clustering ช่วยลด CPU overhead
- ใน NVM ต้องเลือกใช้ algorithm ดี ๆ
  - ถ้าใช้ algorithm เก่า อาจทำให้เปลือง CPU

## Free-Space Management

- ใน file system จะมีเก็บ free-space list ซึ่งสำหรับติดตามพื้นที่ที่เหลือ
  - ↳ implement ด้วย bit-vector หรือ bit-map

## Block Number Calculation

(number of bits per word) *(number of 0-value words) +offset of first 1 bit

example :

block size = 4KB = $2^{12}$ bytes
disk size = $2^{40}$ bytes (1 terabyte)
$n = 2^{40}/2^{12} = 2^{28}$ bits (or 32MB)
if clusters of 4 blocks -> 8MB of memory

เหมาะกับ contiguous file

## Linked free Space List on Disk

- หาพื้นที่เหลือต่อเนื่องยาก
- ไม่ต้อง traverse ทั้ง list (ถ้าบันทึกจำนวน blocks ที่ free อยู่)

## Free-Space Management

- Grouping - เก็บ จำนวนช่องที่ว่างต่อเนื่อง
- Counting
- Space Maps (ZFS)
  - แบ่งเป็น metaslab
    - record ลง log

## Effciency and Performance

ประสิทธิภาพ ขึ้นอยู่กับ

- Disk allocation and directory algorithm
- Types of data
- Pre-allocation
- Fixed-Size / Vary-Size data structure

Performance

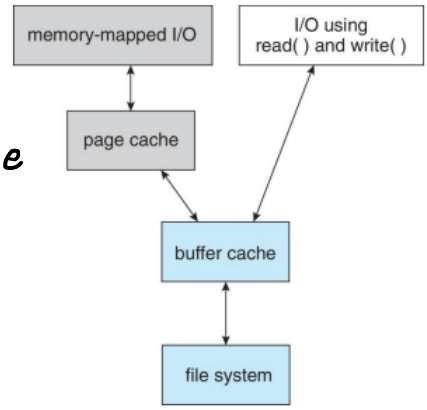
- เอา data และ metadata ไว้ใกล้กัน
- Buffer Cache แยก block ที่ใช้บ่อยออกมา
- Synchronous - เขียนเมื่อ app/OS ต้องการ, ไม่มี buffer (เข้า disk โดยตรง)
- Asynchronous - เขียนมากกว่า - เร็วกว่า
  - มี buffer
- Free-behind / Read-ahead

* Read ช้ากว่า Write

## Page Cache

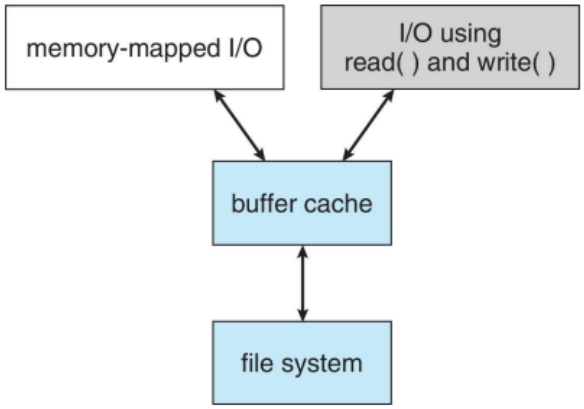
- เป็นการ cache page โดยใช้ virtual memory
- memory-mapped I/O ใช้สำหรับ page cache
- Routine I/O ผ่าน File system ใช้ buffer (disk) cache

# I/O Without a Unified Buffer Cache




# Unified Buffer Cache

- ใช้แทน page cache และ buffer cache เพื่อลดความซ้ำซ้อน (หลีกเลี่ยง double caching)




# Recovery

- Consistency checking - ตรวจสอบ data ใน directory structure กับ data blocks บน disk และ fix inconsistencies
  - อาจช้าและบางครั้งก็พัง
- ใช้โปรแกรม Backup และ restore เมื่อไฟล์หาย

# Log Structure File Systems (Journaling)

- ทุกการกระทำจะถูกลง log
  - บางครั้ง log อาจจะแยกไปอีกอุปกรณ์
  - แต่บางครั้ง file system อาจจะยังไม่ updated
- จด log แบบ asynchronous
- ถ้า file system ล่ม ก็จะสามารถนำ transaction ที่เหลือมาทำให้หมดได้
- recover เร็ว ex. WAFL, APFS, HFS+

# File System Internals

## File System

- Computer อาจมีหลาย storage
- อุปกรณ์จะถูกแบ่งเป็น partitions → hold → volume
- volume อาจมีหลาย partitions ก็ได้
- แต่ละ volume ก็มี file system ของมันเอง

## Partitions and Mounting

- partition
  - raw = ไม่มี file system
  - cooked = มี file system แล้ว
- boot block → boot volume → boot loader

หรือใช้ boot-manager ถ้ามีหลาย OS

- Root Partition
  - ใส่ OS เข้าไปแล้ว partition อื่นๆ ใส่ OS อื่นๆ เข้าไปได้
  - mounted ตอน boot
  - partition อื่นๆอาจ mount auto หรือ manual ก็ได้

- ในช่วงเวลาที่ mount จะมีการเช็ค file system
  - ตรวจสอบ metadata ว่าถูกต้องหรือไม่
    - ถ้าไม่ → ซ่อม → เอาใหม่
    - ถ้าถูกต้อง → เอาเข้า mount table และ allow access

## File Sharing

- เป็นการ allow ให้ user/system เข้าถึง file เดียวกัน
- permission ต้องถูกต้องและเหมาะสม
  - OS ส่วนมากจะมี group, owner member
  - มีวิธีที่จะ apply เข้าไปยังระบบได้

## Virtual File Systems

- เป็นของ Unix ช่วยในการสร้าง file-system ในรูปแบบ Object-Oriented
  - ใช้ API เดียวกันใน File system ที่แตกต่างกัน

ใน Linux จะมี 4 object types
- inode, file, superblock, dentry

VFS จะมี กลุ่มของ operation ไว้ใช้ในการ implement

## Remote File Systems

- เป็นการ share file ใน network
  - FTP
  - ใช้ Distributed File System
  - ใช้ www ( google drive)

## Client-Server Model

- เป็นการ share โดย server allow ผ่าน network protocol
  client ใช้ protocol ในการเข้าถึง remote file system
- พื้นฐานต้องตอบด้วย network ID เพื่อใช้การ encryption

## Distributed Information Systems

- เป็นวิธีการเข้าถึง remote computing
  aka. distributed naming services
- DNS → domain เป็น ip-address
- Network Information Service - provide
  - username
  - password
  - user ID
  - group information
- common internet file system ใช้ network info + auth เพื่อสร้าง network login
  - Active Directory distributed-naming service
  - Kerberos-derived network auth protocol
- Lightweight directory-access protocol

# Consistency Semantics

- เป็น criteria สำหรับ การ evaluating file ใน sharing-FS
- เจาะจง user หลายๆ คน ว่ามีการเข้าถึง shared-file ได้อย่างไร
  - เมื่อ modify ข้อมูล user อื่นจะมองเห็น
  - การทำให้ไฟล์ atomic
- จะเรียก การทำนี้ว่า file-session

  open/close

## UNIX

- คนอื่นจะเห็นว่ามีการ เขียนในไฟล์ที่เปิดอยู่
- share pointer ใน I/O location ใน file
- physical image อันเดียว access แบบ exclusive

  แต่ทำให้เกิด process delay

## Session Semantic (Open AFS)

- ขณะเขียนคนอื่นมองไม่เห็น (เห็นเมื่อ ปิดไฟล์)
- อาจมี หลาย copy ได้

## Sun Network File System (NFS)

- เป็น file system สำหรับเข้าถึง file ผ่าน LAN สำหรับเข้าถึงผ่าน

  TCP, UDP
- remote directory mount ทับ local directory
- เป็น non-transparent ต้องใช้ hostname ก่อน

  ถึงจะเข้าถึงแบบ transparent ได้

NFS - สามารถใช้ bridge ในระบบที่ต่างกันได้

NFS mount protocol

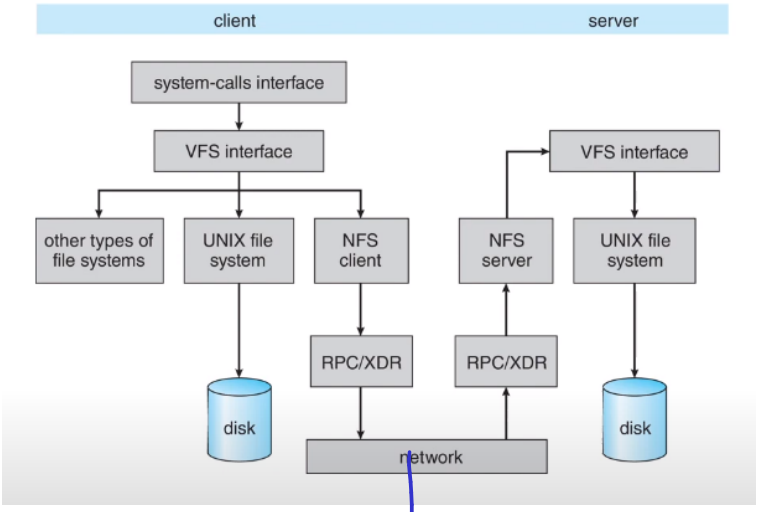
requirement : - ย้าย directory ระหว่าง server - client

- mount request เป็น RPC
- เมื่อ request แล้ว server จะให้ key กลับมาเข้าถึง
- server ไม่เปลี่ยนแปลง

ใน NFS protocol จะมีคำสั่งมาให้ตามนี้

- search
- read
- link
- เข้าถึง file attributes
- R/W

* เป็น stateless ต้องใส่ argument ในครั้งเดียว (ขอ request ในแต่ละครั้ง)




มีการทำ path translation