

Operating System

Final

**Edited 1
06/05/2018**

Memory Hierarchy Design

Terminology

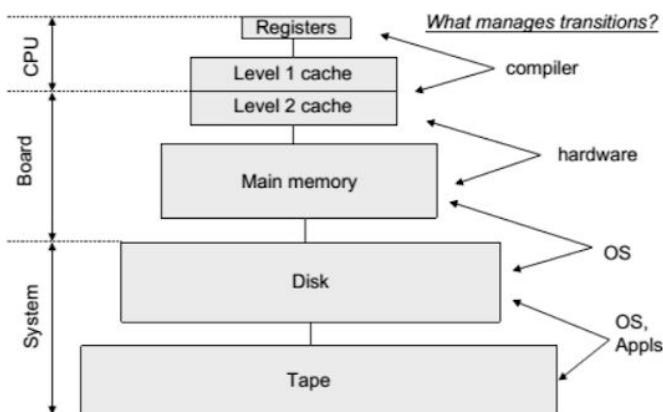
- Memory Latency
ระยะเวลาตั้งแต่ cpu ส่ง request ไปจนกระทั่งได้ data กลับมา
- Memory Bandwidth
อัตราการส่งข้อมูลในหนึ่งหน่วยเวลา
- Memory Hierarchy
การจัดวาง memory เป็นลำดับชั้นเรียงตามความเร็วในการทำงาน
- Upper Level memory
Memory ที่อยู่ชั้นบน มีความเร็วสูงกว่า อญี่โภค์ cpu มากกว่า
- Lower Level memory
Memory ที่อยู่ชั้นล่าง มีความเร็วต่ำกว่า อญี่โภค์ cpu
- Cache
คือ หน่วยความจำอย่างนึง มีความเร็วในการเข้าถึงและการถ่ายโอนข้อมูลที่สูง ซึ่งมีหน้าที่ในการเก็บข้อมูลที่เราต้องการจะใช้งานบ่อยๆ

Remind

สิ่งที่บันทึกนี้ชอบพูดถึงเสมอคือ data ที่ใช้บ่อยๆมาอยู่ใกล้ๆ cpu ความใกล้นี้ไม่ได้พูดถึงในเชิงระยะทาง ไม่อย่างนั้น เอา hard disk มาต่อ กับ cpu ก็จะแล้ว ความใกล้นี้พูดถึงในเชิงเวลา ใกล้หมายถึงเข้าถึงได้เร็ว

Aim of Memory Hierarchy

- Goal
อย่างได้ memory ที่เร็วๆ และเร็วๆ ถูกๆ ด้วย
- Solution
สร้างระบบลำดับของ memory
 - ใช้ memory ราคาถูกๆ จัดเก็บข้อมูล เพื่อให้ดูเหมือนว่าเก็บข้อมูลได้มาก
 - ใช้ memory เร็วๆ มาทำงาน เพื่อให้ดูเหมือนว่าทำงานได้เร็ว



Almost Everything is a cache

- Register เป็น cache ของตัวแปร
- L1 cache เป็น cache ของ L2 cache
- Memory เป็น cache ของ Disk

Inclusive

- Data อะไรมีก็ตาม ถ้าเจอกับ upper level ก็จะเจอกับ lower level ด้วยเสมอ

Locality

- Temporal Locality ประโยชน์ด้านเวลา
 - การ copy ข้อมูลมาจาก memory ไว้ใน cache ซึ่งจะทำให้ memory speed แค่ครึ่งเดียว
 - การเรียกใช้ข้อมูลที่อยู่ใน cache หลายครั้ง
- Spatial Locality ประโยชน์ด้านพื้นที่
 - การ copy ข้อมูลมาใส่ใน cache ทั้ง block ทำให้ไม่ต้อง copy บ่อยๆ

Terminology 2

- Hit ข้อมูลที่อยู่ได้ อยู่ใน cache
 - Hit rate = จำนวนครั้งที่ hit / จำนวนครั้งที่หา
 - Hit time = เวลาเข้าถึงข้อมูล + เวลาที่เหลือว่า hit เป็น
- Miss ข้อมูลที่อยู่ได้ ไม่อยู่ใน cache
 - Miss rate = 1 - hit rate = จำนวนครั้งที่ miss / จำนวนครั้งที่หา
 - Miss penalty = time(replace memory to cache) + time(replace cache to cpu)
= initial access time + transfer time
- Cache block Memory ชุดหนึ่ง ที่มีข้อมูลที่ร้องขอไปอยู่
- Block address ตำแหน่งของ block ใน memory

Four Memory Hierarchy Problem

- Where

จะเอา block จาก lower ไปไว้ตรงไหนของ upper => Block Placement

- Direct Mapped มีตำแหน่งเดียวที่จะวางได้
 - Block no. = (Block address) mod (Number of block in cache)
- Fully Associative ตรงไหนก็ได้
- Set Associative กำหนด set ของ block เช่น 1 set = 2 block
 - เรียก 1 set = n block ว่า n-way set associative
 - Set no. = (Block address) mod (Number of set in cache)
- Example ถ้า cache ขนาด 8 block จะเอา block address = 12 มาใส่
 - Direct Mapped
Block no. = 12 mod 8 = 4 จะวางที่ตำแหน่ง 4 ใน cache
 - Fully Associative วางตรงไหนก็ได้
 - Set Associative (n = 2)
Set no. = 12 % 4 = 0 จะวางใน set ที่ 0 (แต่จะวาง block ไหนต้องเลือกอีกที)

■ Associativity

สมมติมี cache **m** block

- Direct Mapped = 1-way set associative
- Fully Associative = m-way set associative

- How

จะอยู่ได้ใน block ที่ห้าอยู่ใน upper level => Block Identification

- นั่นคือ กูก็ไม่รู้ ตัวอย่างก็ไม่มีใน

- Which

จะเอา block ใหม่มาแทน block เก่าอันไหน => Block Replacement

- Random สุ่มแมง ไม่เห็นต้องคิดมาก
- Least Recently Used เอา block ที่ไม่ได้ใช้มานานที่สุดออก
- FIFO ใหม่มาเก่า ออกก่อน

- What happen

จะเขียนข้อมูลกลับไปใน lower level ตอนไหน => Write Strategies

- Write on hit

การเขียนใน block ที่ hit มีได้ 2 แบบ

- Write Through

เวลาจะเขียนลง upper level ก็จะเขียนลง lower level ไปด้วยพร้อมๆกัน

- Write Stall cpu ต้องรอ lower เขียนเสร็จด้วย
 - Write Buffer cpu เขียนลง buffer ก่อนแล้วไปทำงานต่อ
 - Cache always clean ข้อมูลใน upper และ lower ตรงกันตลอด

- Write Back

เวลาจะเขียน เขียนใน upper อ่อนๆเดียว จะเขียนลง lower ก็ต่อเมื่อ จะโดน replace

- มี dirty bit บอกว่า block นี้ถูกแก้ไขหรือไม่
 - 0 = clean ตอน replace ไม่ต้องเขียนใน lower
 - 1 = dirty ตอน replace ต้องเขียนใน lower

- Write on miss

การเขียนใน block ที่ miss มีได้ 2 แบบ

- Write Allocate

- ก่อนจะเขียน จะต้องเอา block มาไว้ใน upper ก่อนแล้วค่อยเขียน
 - หลังจากนั้นก็ทำงานเหมือน write back

- No-Write Allocate

- ไม่เขียนใน lower เลย ไม่ต้องเอาขึ้นมา

Review Performance Equations

- CPU execution time

$$\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory Stall cycle}) * \text{Clock cycle time}$$

- Memory Stall cycle

$$\text{Memory Stall cycle} = \text{Number of misses} * \text{Miss penalty}$$

$$= \text{Instruction count} * (\text{Misses / Instruction}) * \text{Miss Penalty}$$

$$= \text{Instruction count} * (\text{Read / Instruction}) * \text{Miss rate} * \text{Miss Penalty}$$

+

$$\text{Instruction count} * (\text{Write / Instruction}) * \text{Miss rate} * \text{Miss Penalty}$$

- Average Memory Access Time (AMAT)

$$\text{AMAT} = \text{Hit time} + \text{Miss rate} * \text{Miss Penalty}$$

Two type of cache

- Unified cache

เป็น cache ที่เก็บทั้ง data และ instruction

- Advantage

- Hit rate สูง เพราะสมดุลของขนาด data cache และ instruction cache จะเกิดขึ้นเอง
 - ต้องการ cache แค่อันเดียว ก็สามารถใช้งานได้แล้ว

- Split cache

แบ่ง cache ออกเป็น data cache และ instruction cache

- Advantage

- Data และ instruction ไม่ต้องแบ่ง memory กัน

- Disadvantage

- จำกัดพื้นที่ของแต่ละ cache

เช่น ถ้า data cache เต็ม แต่ instruction cache ว่าง data ใหม่ก็เข้าไม่ได้

Memory Management

Memory Allocation

1. Contiguous Allocation

1.1. Fixed Partitioning

- ให้ memory ทั้ง process ตามขนาดที่กำหนดไว้
- ทำให้เกิด **internal fragmentation**
- fix: ขนาด, จำนวน, ตำแหน่ง ของ Partition

1.2. Dynamic Partitioning

- ให้ memory ทั้ง process ตามขนาดที่ process ต้องการ
- ทำให้เกิด **external fragmentation**
- แก้ปัญหาโดยการ Compaction (memory) / defragmentation (disk) / garbage collection คือ การย้ายพื้นที่ว่างมาวางเรียงต่อกัน

2. Non-Contiguous Allocation (Virtual Memory)

2.1. Paging

■ Basic Structure

- แบ่ง memory ออกเป็นส่วนๆเท่ากัน เรียกว่า frame
- แบ่ง process ออกเป็นส่วนๆเท่ากัน เรียกว่า page
- Frame และ Page มีขนาดเท่ากัน
- OS มี page table ไว้สำหรับแต่ละ process
 - เก็บตำแหน่ง frame สำหรับแต่ละ page
 - Memory address = (page number, offset)
- Number of Virtual Page = Virtual address Space / Page Size
- Page Address Length = \log_2 (Number of Page)

■ Page Table

- ใช้เพื่อเปลี่ยน virtual address ให้เป็น physical address
 - Virtual address = [virtual page number][offset]
 - Page number = Virtual address // Page size
 - Offset = Virtual address % Page size
 - Physical page number = **Page Table**[page number]
 - Physical address = [physical page number][offset]

- Page Table Entry

- Two level page table

- Translation Lookaside Buffers (TLB)

- Hardware พิเศษ ช่วยให้หา อะไรวะ? ได้เร็วขึ้น
 - Effective Access Time 🍪😊🍪

$$\blacksquare \quad EAT = (m + e)a + (2m + e)(1 - a)$$

- m = memory cycle time ⏱
 - e = TLB Lookup
 - a = hit rate

- Hashed Page Table

- มี Hash Function เปลี่ยนค่า virtual page number => key
 - มี Hash Table เก็บ chain ของ element
 - Virtual page number => physical page number

$$\blacksquare \quad \text{Hash(virtual page number)} \Rightarrow \text{key}$$

$$\blacksquare \quad \text{elements} = \text{Hash Table}[key]$$

■ find element in element with same Virtual Page Number

■

- Inverted Page Table

- Table เก็บเฉพาะ Frame Page ที่มีการ map ซึ่งทำให้ Page Table เล็กลง แต่ค้นหาอย่างยากกว่าเดิม
- Page Table จะมีขนาดเท่ากับจำนวน Frame เรียงตาม Frame number
- Frame ทั้งหมดเป็นของหลาย Process ปะปนกัน จึงต้องมี attribute ที่บอกว่า Frame นี้ เป็นของ Process ID อะไร และมี Virtual Page ไหน map มา

- Page Replacement Algorithm

วิธีการเลือก page ที่จะถูกแทนที่

- Optimal Page Replacement Algorithm

- มองออกไปยังอนาคต ลบอันที่อยู่ไกลสุด
- ข้อเสีย : ชีวิตจริงมองเห็นอนาคตได้ที่ไหนล่ะ

- Not Recently Used

- แต่ละ page มี R bit และ M bit ประจำตัว
- ทุกๆรอบการทำงาน $R, M = 0$
- ถ้ามีการอ่าน $R=1$, ถ้ามีการเขียน $M=1$
- จัดความสำคัญของ page

- Class 1 - $R=0, M=0$ ต่ำสุด
- Class 2 - $R=0, M=1$ ↓ ↓
- Class 3 - $R=1, M=0$ ↓ ↓
- Class 4 - $R=1, M=1$ สูงสุด

- เมื่อจะเอา page ออก จะสุ่มเอา page ที่มี class ต่ำสุดออก

- First In First Out

- มาก่อน ออกก่อน
- ทำงานง่ายๆโดยใช้ queue

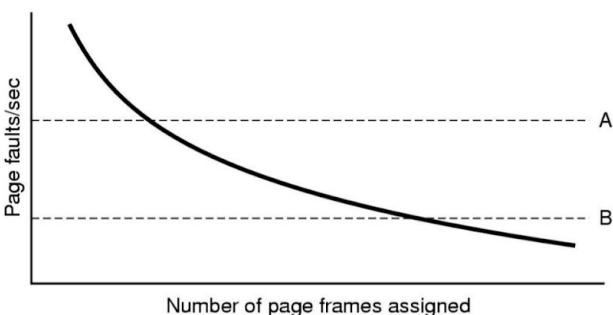
- Second Chance

- เป็นร่างสองของ FIFO
- แต่ละ page จะมี R bit และ load time
- $R=1$ เมื่อถูกอ่าน
- เมื่อจะเอา page ออก จะเลือกจากหัวคิว
 - ถ้ามี $R=1$
 - รีเซ็ต $R=0$
 - แก้ load time = now
 - ย้ายไปต่อท้าย queue
 - ถูด้วนต่อไป
 - ถ้ามี $R=0$
 - เอาออก

- Clock

- เรียง page เป็นวงกลม
- แต่ละ page มี R bit
- มีเข็มซึ่ง page เมื่อมีนาฬิกา เมื่อต้องการจะลบ page ดูที่เข็ม
 - $R=1$
 - รีเซ็ต $R=0$
 - เลื่อนเข็ม หมุนตามเข็มนาฬิกา
 - $R=0$
 - เอาออก

- Least Recently Used (LRU)

- เอ้า page ที่ไม่ได้ใช้งานที่สุดออก
 - $n \times n$ Matrix
 - สร้าง matrix ขนาด $page * page$
 - ใส่ 0 ทุกช่อง
 - เมื่อเข้าถึง page k
 - $row(k) = 1$
 - $col(k) = 0$
 - เมื่อลง จะลบ page ที่ $sum(row)$ น้อยสุด
 - 64-bit Counter
 - Not Frequently Used
 - Aging
 - แต่ละ page มี counter ของตัวเอง
 - มี R bit ด้วย
 - แต่ละรอบถ้าถูกอ่าน $R=1$
 - หลังจากนั้นแต่ละรอบจะนำ R bit ไปบวกที่หน้า counter
 - ลบ page ที่มี counter น้อยสุด
 - Working Set
 - Working Set Clock
 - Belady's Anomaly
 - เหตุการณ์ที่เพิ่ม frame แต่ทำให้ Page Faults มากขึ้น
 - อาจเกิดจาก $M(m, r) \not\subseteq M(m+1, r)$
 - Stack Algorithm
 - Thrash
 - คือเหตุการณ์ที่ OS ไปจัดการ page faults บ่อยจนไม่ว่างทำอย่างอื่น
 - Cycle of Death
 - Thrashing \rightarrow I/O เยอะ \rightarrow CPU utilization ต่ำ \rightarrow load process เพิ่ม \rightarrow Thrashing $\rightarrow \dots$
 - Thrashing จะเกิดเมื่อ
 - $sum(working\ set) > total\ memory\ size$
 - ดูจากกราฟ ถ้า degree of multiprogramming เยอะ แต่ cpu utilization ต่ำ มาก แสดงว่าจะเกิดแล้วล่ะ
 - วิธีหลีกเลี่ยง thrashing
 

The graph plots 'Page faults/sec' on the vertical axis against 'Number of page frames assigned' on the horizontal axis. Two curves are shown: Curve A, which starts at a higher point and levels off at a higher horizontal value; and Curve B, which starts lower and levels off at a lower horizontal value. Dashed lines connect the curves to their respective asymptotes.

 - ถ้าเกินขีดจำกัดบน (เส้น A) \Rightarrow เพิ่ม frame ที่ให้
 - ถ้าเกินขีดจำกัดล่าง (เส้น B) \Rightarrow ลด frame ที่ให้
 - Page Size
 - Small Page Size
 - เกิด internal fragment น้อย
 - Page table ใหญ่
 - Large Page Size
 - Page fault ต่ำ

- ชื่อ module ที่ไปเรียนมาอาจไม่ได้ใช้
- Overhead = page table space + internal fragment
 - = $(s^e)/p$ + $p/2$
 - s = average page size
 - p = page size
 - e = page entry
- Optimized when $p = \text{root}(2se)$
- Shared Pages
- Cleaning Policy
- Fetch Policy
 - Demand paging
 - ไปดึง page มาเมื่อต้องการจะใช้
 - Pre-paging
 - ไม่ได้ใช้ก็เอามาเก็บ
- Page Fault Handling

2.2. Segmentation

- Basic Structure
 - แบ่ง process ออกเป็นส่วนๆเรียกว่า segment
 - แต่ละ segment ไม่จำเป็นต้องมีขนาดเท่ากัน
 - แต่ละ segment สามารถได้

2.3. Multics : Paging + Segmentation

File Systems

- หน่วยเก็บ Long-term storage สามารถเก็บได้
 - หน่วยในการเก็บข้อมูลทุกอย่างใน storage เราเก็บเป็น electronic file
 - สมมุติถ้าเก็บข้อมูลที่ disk(storage) จะหาได้อย่างไรว่าเก็บที่ตรงไหน และดูว่าต้องมีการติดตามการใช้พื้นที่ ทำยังไงเพื่อ protection การเข้าถึง และถ้าอยากรอพื้นที่เพิ่ม จะรู้ได้ว่าว่าพื้นที่ตรงไหนว่างอยู่ และว่างพอไหม
1. สร้าง File Systems เพราะจะช่วยให้เข้าถึงแฟ้มได้ตามความต้องการ
 - มองว่า Storage คือ Disk ประกอบด้วย Sequence ของ block เรียงต่อกันเป็นเชิงเส้น
 - โดยแต่ละ block มีขนาดคงที่ สามารถเข้าถึงเพื่ออ่านหรือเขียนได้
 2. การตั้งชื่อ file
 - ประกอบด้วย Extension = นามสกุลไฟล์ จะผูกติดกับ Application ที่จะทำงานกับไฟล์นั้นๆได้
 3. File Structure
 - Byte Sequence ต้องรู้ว่าแต่ละ byte เป็นอะไร
 - Record Sequence แต่ละ record จะมี attribute โครงสร้าง, อ่าน-เขียน ยก record
 - Tree เข้าถึงโดยการระบุ index - เป็นแบบ direct access
 4. File Type
 - Executable
 - มีMetadata (data of data) = data ที่เกี่ยวกับไฟล์ ใช้ในการจัดการ/ทำงานกับfile นี้
 - Text size
 - Data size
 - Symbol table size
 - Archive
 - Header
 - Metadata: Date, Owner, Protection, Size
 - Object Module
 5. File Attributes อยู่ใน Metadata เป็นรายละเอียดของ File
 - Owner, Password, Protection
 - Creator, Creation time
 - Time of last access, Time of last change
 6. นิยาม File Operations - เราทำอะไรกับไฟล์ได้บ้าง
 - Create, Delete, Open, Close ระดับของ file
 - Read, Write รายละเอียด(Content) ใน file
 - Append, Seek, Get Attributes, Set Attributes, Rename
 7. Directory Operation - System call สำหรับจัดการ Directory
 - Create, Delete, Opendir, Closedir, Readdir, Link, Uplink
 8. File System Layout - โครงสร้างใน Storage ประกอบด้วย Liner-Sequence ของ Block
 - Disk มีการแบ่งส่วนของพื้นที่ออกเป็นหลาย Partition
 - จะมี Partition table(ก็เป็น Metadata) ที่ดูบอกว่า แบ่งพื้นที่ออกเป็นกี่ Partition แต่ละ Partition มีSizeเท่าไร อยู่ตำแหน่งไหน
 - 1 Partition มี
 - MBR (Master Boot Record) = ตำแหน่งเริ่มต้น หรือ sector แรกของฮาร์ดิสก์ / ใช้เก็บค่าเริ่มต้นของระบบปฏิบัติ
 - Boot block จะทำหน้าที่ในการโหลดระบบปฏิบัติการใน Partition ออกมาทำงาน ทุก ๆ Partition จะมี Boot block เป็นของตัวเอง
 - Superblock เก็บรักษาข้อมูล เกี่ยวกับระบบไฟล์ทั้งระบบ และรวมไฟล์ต่อไปนี้:
 - ขนาดของระบบไฟล์
 - จำนวนบล็อกข้อมูลในระบบไฟล์
 - Flag ที่ระบุสถานะของระบบไฟล์
 - ขนาดกลุ่มการจัดสรร

9. Contiguous Allocation จัดสรรพื้นที่ต่อเนื่องกัน

- แต่ละไฟล์ต้องมี size ที่แน่นอน
- 1 file มี หลาย Block แต่ทุก Block เรียงต่อกัน
- เกิด external fragmentation พื้นที่ว่างที่อยู่ภายนอก block ที่ทำให้พื้นที่ว่างอยู่กระจำจัดกระจายกัน
- ข้อดี
 - เช้าถึงไฟล์ได้ง่ายขึ้น
- ข้อเสีย
 - ถ้า Size ไฟล์ใหญ่ขึ้นจะทำให้ไปหักกับไฟล์อื่น ทำให้จัดสรรพื้นที่ยาก ต้องย้ายไปหาพื้นที่ใหม่ที่ใหญ่กว่า

10. Linked List Allocation แก้ปัญหาการเปลี่ยนแปลงของ Size ไฟล์

- 1 file มี หลาย Block แต่ละ Block ไม่เรียงต่อกัน (แต่จะใช้การ link ทุก Block เช้าด้วยกัน)
- แต่ต้องเสียพื้นที่ภายใน Block เพื่อบอกว่า Block ถัดไปอยู่ที่ไหน
- หรืออาจใช้ File Allocation Table
 - คือตารางที่เก็บหมายเลขของ Block บอกว่า Block นี้ link ไปที่ Block ไหน
 - Table นี้ต้องอยู่ที่ Memory

11. I-nodes แนวคิดของ UNIX

- เป็นเสมือน Table ของ 1 file ใน Table จะบอกว่าไฟล์นั้นมี Block หมายเลขอะไรบ้าง

12. Implementing Long File Name

- ชื่อไฟล์ 5 Char นามสกุล 3 Char -> ----- . ---
- ปัจจุบัน File System รองรับ Long File Name - ชื่อยาวแค่ไหนก็ได้
- โดยมี 2 วิธี คือ
 - In-line
 - เก็บชื่อไปพร้อมกับรายละเอียด(Attribute)
 - ชื่อที่ยาวไม่เท่ากันทำให้แต่ละรายการ (ใน Directory) มีขนาดไม่เท่ากัน
 - การที่มันยาวไม่เท่ากันทำให้เราต้องเก็บ length ด้วย จะต้องรู้ว่าต้องอ่านมากแค่ไหน
 - In a heap
 - เก็บชื่อแยกออกจากรายละเอียด
 - โดยชื่อของแต่ละรายการมาเก็บที่ Heap โดยที่แต่ละรายการจะมี Pointer โยงมาที่ชื่อของตัวเอง
 - ทำให้ชื่อไฟล์ยาวแค่ไหนก็ได้ เพราะใช้พื้นที่แยกออกจากรายการ

13. Shared Files แนวคิดของ UNIX

- สามารถเข้าถึงไฟล์เดียวกัน โดยที่มองเห็นการอพเดต/การเปลี่ยนแปลงไปเหมือนๆ กัน
- เมื่อไฟล์ผูกกับ I-node ที่叫作 I-node แทนการแชร์ไฟล์

14. Disk Quotas - โควัตตาของแต่ละ User

- Open file table - บอกว่าตอนนี้มีไฟล์อะไร Open อยู่บ้าง โดยใคร
- ในแต่ละรายการก็จะบอกว่า ใครเป็น Owner ของไฟล์นี้ และก็จะมี Pointer ชี้ไปที่ Quota table ที่จะบอกว่า User แต่ละคนมีโควัตตาในการใช้ disk space อย่างไรบ้าง ในการสร้างไฟล์กี่ไฟล์
- Soft Limit การเตือนว่า disk ใกล้เต็ม หรือจำนวนไฟล์ที่สร้างได้ใกล้จะครบทั้งหมด
- Hard Limit อาจทำการ Lock ระบบไม่สามารถสร้างไฟล์ได้ หรือเพาะ Disk เต็ม

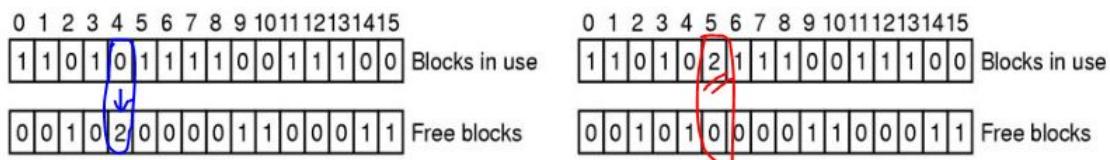
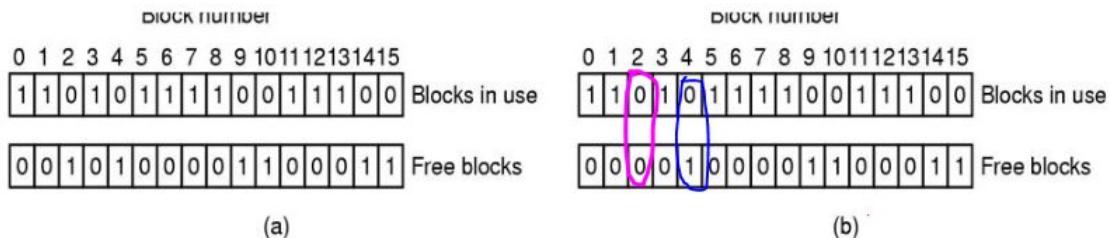
15. File System Backups

- การ Backup มี 2 แบบ ดังนี้
 - Full Backup - Backup ทั้งหมด, นานๆ ทำที, ใช้เวลานาน
 - Incremental Backup - Backup เฉพาะส่วนที่ปรับปรุงหรือเพิ่มขึ้นมาใหม่, ใช้เวลาอ้อย

16. File System Consistency

- เก็บข้อมูลจาก table ต่างๆ ใน File system ว่าเกิดปัญหาอะไรบ้าง
- ประกอบด้วย 2 Table คือ

- Blocks in use - Block ที่ถูกใช้จะมี Flag เป็น 1 ช่องเดียวจาก I-node
 - Free blocks - เก็บมาจาก Free Space
 - ถ้าไฟล์อยู่ในสถานะ **Consistency** แสดงว่าจะมี flag ที่เป็น 1 อยู่ที่ table เดียวกันนั้น ไม่ Blocks in use ก็ต้อง Free blocks
- ปัญหา เช่น
- สีชมพู(2) - Blocks in use และ Free Blocks เป็น 0 -> เพิ่ม Block 2 ไว้ที่ Free Blocks
 - สีน้ำเงิน(4) - Free Blocks เป็น 2 มีโอกาสเกิด Allocate 2 ครั้ง -> ลบ Block 4 ออกจาก Free Blocks ไป 1 อัน
 - สีแดง(5)- Blocks in use เป็น 2 แสดงว่าอยู่ใน 2 I-node(หรือ 2 file) -> สร้างอีก Block และ ให้ไป และ copy content ของ Block 5 ไป



I/O management and disk scheduling

Categories of I/O devices

- Human readable
 - ติดต่อกับคน
 - Ex. printer, monitor, keyboard
- Machine readable
 - ใช้ในการทำงาน, ติดต่อกันเอง
 - Ex. disk, sensor, controller
- Communication
 - ใช้ติดต่อกันเองระยะไกล
 - Ex. modem

Difference in I/O devices

- Data rate
 - มีอัตราการขนย้ายข้อมูลที่แตกต่างกัน
- Application
 - มีกลไกการเข้าถึงข้อมูลที่แตกต่างกัน
- Complexity of control
 - มีความซับซ้อนของการทำงานต่างกัน
- Unit of transfer
 - หน่วยของข้อมูลในการขนย้ายต่างกัน
 - บางอย่างถูกจัดเป็น block บางทีก็เป็น segment, byte, bit
- Data representation
 - มีการ encode ข้อมูลต่างกัน
- Error condition
 - มีสาเหตุที่ทำให้เกิด error ต่างกัน
 - CD-drive อาจเกิดปัญหาผู้นุ่นการเลยอ่านไม่ได้ แต่ปัญหานี้ไม่เคยเกิดกับแม่ส์

Performing I/O

- Programmed I/O
 - ใช้ busy waiting รอให้ I/O ทำงานเสร็จ
- Interrupt-driven I/O
 - Processor ก็ทำอย่างอื่นไป I/O เสร็จชิ้นนึงแล้วจะมาบอกรอ
- Direct Memory Access (DMA)
 - มี DMA module ช่วยในการส่งงาน I/O เมื่อเสร็จงานชุดๆ นึงแล้วค่อยไปบอกรอ processor

Evolution of I/O Function

- Processor ติดต่อกับ device โดยตรง
- มี controller, I/O module มาช่วยติดต่อกับ device
 - Processor ไม่ต้องรู้รายละเอียดของ device
- Controller, I/O module with interrupt
 - Processor สามารถทำงานกับ I/O ได้โดยไม่ต้อง interrupt แล้ว
- กำเนิด DMA
 - ควบคุมการขนย้ายข้อมูลได้เอง
 - Processor จะควบคุมแค่ตอนเริ่มและตอนจบการขนย้าย
- I/O module กลายเป็น processor
 - I/O module มี local memory ของตัวเอง

- มีความสามารถที่ยืนได้กับคอมพิวเตอร์เครื่องหนึ่ง

Direct Memory Access (DMA)

- Processor จะมอนหมายงานเกี่ยวกับ I/O ให้กับ DMA
- DMA ถ่ายข้อมูลระหว่าง device และ memory
- พอดีร์จหมดแล้วจะส่ง interrupt signal ไปหา processor

Operating System Design Issues

- Efficiency
 - Device ส่วนมากทำงานช้ากว่า memory อายุยิ่ง
 - Device มี speed ตาม processor ไม่ทันแน่นอน
 - เพื่อให้รอง multiprogramming ต้องสามารถทำให้ process อื่นทำงานได้ ในขณะที่ process หนึ่งๆ รอ device
- Generality
 - ต้องสามารถรับรอง device ทุกชนิดในรูปแบบเดียวกัน
 - ชั้นรายละเอียดการทำงานกับ device ไว้ในระดับล่าง (lower-level routine)
 - ในระดับบน (upper-level routine) รับรู้เพียงคำสั่งพื้นฐาน read, write, open, ...

I/O Buffering

- Reasons for buffering
 - Process ต้องรอ I/O ทำงานเสร็จลึกลงจะทำงานต่อได้
 - Page ยังคงอยู่ใน memory ขณะที่รอ I/O
- Buffer-oriented
 - Block-oriented
 - ข้อมูลจะเก็บในหน่วยของ block ที่มีขนาดแน่นอน
 - เวลาหยุดย้ายจะย้ายเป็น block
 - ใช้สำหรับ disk และ tape
 - Stream-oriented
 - โอนย้ายข้อมูลเป็น sequence ของ byte (👉 เป็นเส้นๆ เมื่อเน้นสปีก์ตี้ผัดขี้เมากุ้ง)
 - ใช้สำหรับ device ที่ไม่ใช่ disk กับ tape อีก
- Single Buffer
 - OS มี buffer ไว้สำหรับพักข้อมูล 1 อัน
 - ข้อมูลจะอ่านมาจาก I/O แล้วพักไว้ที่ buffer
 - เมื่อเสร็จแล้ว process จึงจะมาอ่านข้อมูลจาก buffer
 - Process ต้องรอ device ส่งเข้า buffer ให้เรียบร้อยก่อนลึกลงจะอ่านได้
 - Device ก็ต้องรอ process อ่านเสร็จลึกลงจะเปลี่ยนอันใหม่ได้
- Double Buffer
 - OS มี buffer ไว้สำหรับพักข้อมูล 2 อัน
 - ในขณะที่อ่านข้อมูลมาใช้ buffer แรก process สามารถอ่าน buffer ที่สองไปพร้อมกันได้
 - device และ process สลับกันใช้ 2 buffer นี้
 - Device ไม่ต้องรอ process อ่านเสร็จแล้ว สามารถไปเขียนอีกอันรอได้เลย
 - แต่อาจเกิดเหตุการณ์ที่ device เขียนอีกอันเสร็จแล้วแต่ process ก็ยังไม่เสร็จ ต้องรอ
- Circular Buffer
 - OS มี buffer ไว้สำหรับพักข้อมูลมากกว่า 2 อัน
 - Process และ device หลักกันใช้ buffer เมื่อ double แต่จะเวียน buffer เป็นวงกลม
 - บรรเทาปัญหาจาก double buffer ได้

Disk Performance Parameters

- Total transfer time = (wait device time) + (wait channel time) + (seek time) + (rotational delay) + (data transfer)
- Seek time
เวลาที่ head เลื่อนไปยัง track ที่ต้องการ
- Rotational delay (rotational latency)
เวลาที่รอ disk หมุนจน sector ที่ต้องการมาเจอ head
- Access time = seek time + rotational delay
- Data transfer เริ่มฟันเมื่อ head เจอ sector ที่ต้องการแล้ว

Disk Scheduling Policies

- Introduction
 - ประสิทธิภาพของ disk ขึ้นอยู่กับ seek time ยิ่งน้อยยิ่งดี
 - แต่ seek time ก็ขึ้นอยู่กับการจัดเรียงข้อมูลที่จะอ่านไปด้วย
 - $\text{avg(seek time)} = \text{sum(seek time)} / \text{count(request)}$
- Disk Scheduling Algorithm
 - First in, First out
 - ถ้า fair ดี แต่ performance ไม่ค่อยดี
 - Priority
 - งานที่ทำหนอยู่จะมี priority สูงกว่า
 - ให้ response time ที่ดี
 - Last-in, first-out
 -
 - อาจเกิด starvation
 - Shortest Service Time First
 - เลือก request ที่ต้อง move disk arm น้อยที่สุดจากตำแหน่งปัจจุบัน
 - SCAN (Elevator)
 - Disk arm จะเคลื่อนที่ไปทิศทางเดียวตลอด ไปจนสุดแล้วค่อยย้อนกลับ
 - Ex. อายากได้ track 1 5 8 6 3 7 2
 - เริ่มแรก disk arm กำลังขึ้น จะเก็บ 1 5 8 ไปจนสุด และเปลี่ยนทิศ
 - Disk arm กำลังลง จะเก็บ 6 3 2 ไปจนสุด และเปลี่ยนทิศ
 - Disk arm กำลังขึ้น จะเก็บ 7 ไปจนสุดทาง
 - C-SCAN
 - Disk arm จะเคลื่อนที่ไปทิศทางเดียวตลอด เมื่อ request ในทิศนั้นหมดแล้วจะย้อนกลับ
 - Ex. อายากได้ track 1 5 8 6 3 7 2
 - เริ่มแรก disk arm กำลังขึ้น จะเก็บ 1 5 8 และย้อนกลับทันที
 - Disk arm กำลังลง จะเก็บ 6 3 2 และย้อนกลับทันที
 - Disk arm กำลังขึ้น จะเก็บ 7

RAID

■ RAID 0 (non-redundant)

- ไม่มีการสำรองข้อมูล
- Disk1 disk2 disk3
 - [1] [2] [3] ← data number สมมติ
 - [4] [5] [6]
 - [7] [8] [9]

■ RAID 1 (mirrored)

- สำรองข้อมูลเหมือนกันเป็นๆ
- Disk 1 disk 2 disk 3 disk 4
 - [1] [2] [1] [2]
 - [3] [4] [3] [4]
 - [5] [6] [5] [6]

Disk 3,4 เป็น mirror ของ disk 1, 2

■ RAID 2 (redundancy through hamming code)

- แบ่งชื่อ默且แต่ละ word ออกไปใส่ disk แต่ละอัน และมี disk ที่เก็บ hamming code สำหรับตรวจสอบความถูกต้องของ word
- ถ้ามีข้อมูลใน disk ใดเสียหาย สามารถถูกกลับคืนได้ด้วย hamming code
- Disk 1 disk 2 disk 3 disk 4 disk 5 disk 6 disk 7
 - [A1] [A2] [A3] [A4] [f₁(A)] [f₂(A)][f₃(A)]
 - [B1] [B2] [B3] [B4] [f₁(B)] [f₂(B)][f₃(B)]

■ RAID 3 (bit-interleaved parity)

- แบ่งชื่อ默且แต่ละ word ออกไปใส่ disk แต่ละอัน และมี disk ที่เก็บ parity
- parity=0 when word.count(1)%2 == 0
- Disk 1 disk 2 disk 3 disk 4 disk 5
 - [A1] [A2] [A3] [A4] [p(A)]
 - [B1] [B2] [B3] [B4] [p(B)]

■ RAID 4 (block-level parity)

- เก็บชื่อ默且แต่ละ block ใน disk
- มี parity bit รวมของ block ในแต่ละแฉว
- Disk1 disk2 disk3 disk4 disk5
 - [1] [2] [3] [4] [p(1-4)]
 - [5] [6] [7] [8] [p(5-8)]

■ RAID 5 (block-level distributed parity)

- ทำงานเหมือน RAID 4 แต่ไม่ระบุ disk ที่เก็บ parity เป็นพิเศษ
- Disk1 disk2 disk3 disk4 disk5
 - [1] [2] [3] [4] [p(1-4)]
 - [5] [6] [p(5-8)] [7] [8]

■ RAID 6 (dual redundancy)

- เหมือน RAID 5 แต่เพิ่ม parity อีก 1 block
- Disk1 disk2 disk3 disk4 disk5 disk6
 - [1] [2] [3] [4] [p(1-4)] [q(1-4)]
 - [5] [6] [p(5-8)] [7] [q(5-8)] [8]

Security

Type of Threats (ประเภทของภัยคุกคาม)

- Interruption
 - ทำให้ระบบไม่พร้อมใช้งาน
 - เป็นการทำลายฮาร์ดแวร์ โดยการตัดการสื่อสาร และปิดระบบการจัดการไฟล์
 - โจมตีด้วยการดักฟังข้อมูล หรือดักคัดลอกไฟล์ด้วยโปรแกรมที่ผิดกฎหมาย
- Interception
 - การลักลอบเอาข้อมูลของผู้อื่น ในขณะที่ผู้อื่นส่งข้อมูลถึงกัน
 - รวมถึงการลักลอบใช้โปรแกรมที่ละเมิดลิขสิทธิ์
 - Wiretapping = การดักจับข้อมูลบน Network
- Modification
 - ข้อมูลจะผ่านตัวโปรแกรมของมั่นก่อน และตัวโปรแกรมของมั่นจะเปลี่ยนข้อมูลที่จะส่งให้อีกคนนึง
- Fabrication
 - มั่นปลอมแปลงตัวเองเป็น sender โจมตีด้วยข้อมูลเท็จ โดยที่ทำเหมือนกับว่า มีsender ส่งข้อมูลให้ปกติ
 - ปลอมตัวเป็นผู้อื่น เพื่อส่งข้อมูล

Computer System Assets

* รูปแบบการโจมตีบน Network เมื่อ Mr.Bob  คุยกับ Ms.Alice  แต่มีมือที่สาม Mr.Darth  *

- Release of message contents
 - ดัก data message
 - ไม่เปลี่ยนแปลงข้อมูล
 - หน้าที่ของผู้ไม่ประสงค์ดี(Mr.Darth) : เผยแพร่ข้อมูลที่ตนเองไม่มีสิทธิ์
- Traffic analysis
 - ดัก รหัสผ่านหรือ รหัสเข้าสู่ ข้อมูลที่ถูกหลอก
 - หน้าที่ของผู้ไม่ประสงค์ดี(Mr.Darth) : ดักข้อมูล มีการนำข้อมูลไปวิเคราะห์ แล้วนำไปใช้ในทางที่ไม่ดี
- Masquerade
 - ปลอมเป็น sender เพื่อส่งข้อมูลให้ receiver
 - หน้าที่ของผู้ไม่ประสงค์ดี(Mr.Darth) : ปลอมตัวเป็น Sender ส่งข้อมูลในนามของผู้อื่น
- Replay
 - มีการดักจับ และ ปลอมแปลงข้อมูล ส่งข้อมูลกลับไปหา receiver อีกครั้ง
 - หน้าที่ของผู้ไม่ประสงค์ดี(Mr.Darth) : แปลงข้อมูล แล้วส่งไปให้ receiver
- Modification
 - มีการดักจับ และ ปลอมแปลงข้อมูล
 - หน้าที่ของผู้ไม่ประสงค์ดี(Mr.Darth) : ทำให้เกิด Delay เวลาส่ง เพื่อทำให้เกิดความผิดพลาดในการสื่อสาร
- Denial of service
 - โจมตี server จน overload ทำให้ sender เข้าใช้งานไม่ได้
 - หน้าที่ของผู้ไม่ประสงค์ดี(Mr.Darth) : ทำให้ sender ไม่สามารถเข้าถึง sever ได้

Protection

- No protection
 - ไม่ป้องกัน เพราะข้อมูลไม่ได้สำคัญ เผยแพร่ได้

- ถ้าข้อมูลสำคัญ ให้ไปทำงานในช่วงเวลาที่คนทั่วไปไม่สามารถเข้าถึงได้
- Isolation
 - แบ่งการทำงานแต่ละส่วน แยกออกจากกันอย่างชัดเจน ให้เป็นอิสระจากกัน
 - ถ้ามีการทำงานร่วมกันของแต่ละส่วน ให้ทำเท่าที่จำเป็น
- Share all or share nothing
 - คล้าย no protection
 - เผยแพร่ข้อมูลได้ แต่มีการกำหนดสิทธิ์ในการเข้าถึง
- Share via access limitation
 - มีการกำหนดสิทธิ์ในการเข้าถึง
 - OS จะเป็นคนให้ permission ในการให้ต่างๆ นั่นหมายความว่า OS จะเป็น Guard ให้นั่นเอง
- Share via dynamic capabilities
 - สามารถปรับแต่งสิทธิ์ในการเข้าถึง ในแต่ละ object ให้เหมาะสมมากขึ้น
- Limit use of an object
 - จำกัดการเข้าถึง และ การใช้ object นั้นๆ
 - สามารถกำหนดวิธีการเข้าใช้งานได้ เช่น อนุญาตให้ใช้ Function คำนวณข้อมูลได้ แต่ไม่สามารถเห็นข้อมูลได้ เป็นต้น

User-Oriented Access Control

- พูดถึงการเข้าใช้งาน แบบผู้ใช้งาน (Log on)
- ประกอบด้วย ID และ Password

Data-Oriented Access Control

- พูดถึงการเข้าถึง data ซึ่งจะมี permission ของแต่ละ file ว่าใคร อ่าน เขียน ลบ ได้มั่ง
- OS เป็นตัวกำหนด rule นี้ ขึ้น

Access Matrix

กำหนด Data Structure ที่ใช้ในการควบคุมสิทธิ์ในการเข้าถึง Object มีองค์ประกอบ

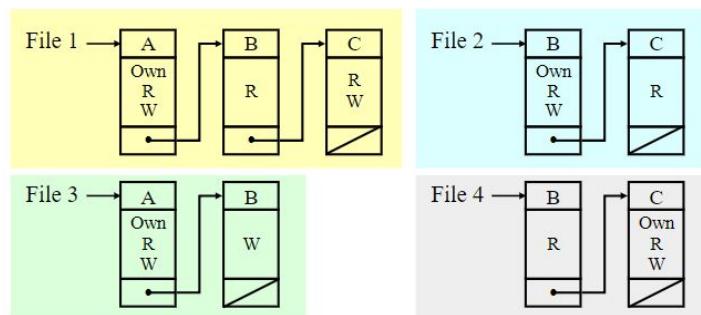
1. Subject -> ผู้ที่เข้าถึง เช่น User
2. Object -> สิ่งที่ถูกเข้าถึง เช่น File, Hardware
3. Access rights -> สิทธิ์ที่ Subject จะเข้าถึง Object นั้นๆ ได้

		Objects			
		File 1	File 2	File 3	File 4
Subjects	User A	Own Read Write		Own Read Write	
	User B	Read	Own Read Write	Write	Read
	User C	Read Write	Read		Own Read Write

- Sparse Matrix = Matrix ขนาดใหญ่ แต่มีข้อมูลน้อย -

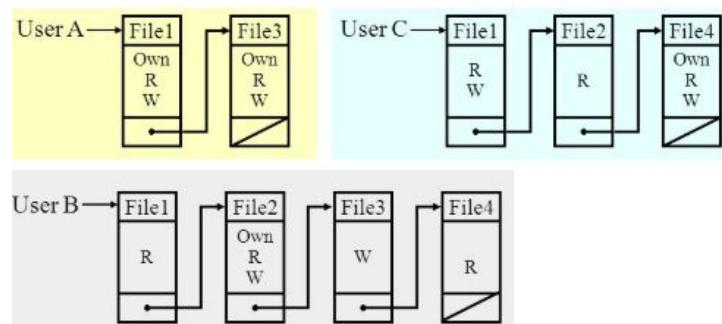
Access Control List

- ไฟล์ส่วนตัว (หรือ Object)
- แต่ละ Object มี User ใดเข้าถึงได้บ้าง



Capability Tickets

- ไฟล์ส่วนตัว (หรือ User)
- แต่ละ User มีความสามารถเข้าถึง Object ใดได้บ้าง



Intrusion Techniques (เทคนิคการบุกรุก)

- กลไกจะเข้า System และอาจไปเพิ่มสิทธิ์ให้ตัวเอง เช่น ทำให้ตัวเองเป็น Admin
- จะเข้ามาได้ก็ต้องรู้ Password หรือมีการคาดเดา Password ได้
- ต้องมีกลไกการป้องกัน

Intrusion Detection (หาพฤติกรรมของผู้บุกรุก)

- Rule-based detection
 - กำหนดติกาที่ชัดเจน เพื่อบอกพฤติกรรมของผู้บุกรุก
 - กติกาเมื่อการปรับเปลี่ยนเพื่อให้สอดคล้องกับสถานการณ์
- Audit record - การตรวจสอบ
 - Native audit records - ตรวจสอบเหตุการณ์ทั่วไป หรือ กติกาทั่วไป
 - Detection-specific audit records - ตรวจสอบเหตุการณ์ที่ไม่寻常สัก

Malicious Programs (โปรแกรมที่ไม่ประสงค์ดี)

1. กลุ่มที่แฟงตัวอยู่ในโปรแกรมอื่น
 - ต้องมีการ activate ให้โปรแกรมนั้นทำงาน Malicious ถึงจะทำงาน
 - เช่น
 - Trapdoor - เส้นทางลับที่จะช่วยผู้บุกรุกเข้าสู่ระบบได้โดยไม่ผ่านกระบวนการตรวจสอบ
 - Logic Bomb - เป็นโปรแกรมหรือส่วนของโปรแกรมที่ถูกสร้างขึ้นในซอฟต์แวร์ใดๆก็ตามโดยมีเป้าหมายในการทำลายข้อมูลตามเงื่อนไขที่ผู้เขียนโปรแกรมตั้งไว้เปิดเวลาไหน
 - Trojan Horse - โปรแกรมคอมพิวเตอร์ที่ถูกบรรจุเข้าไปในคอมพิวเตอร์ เพื่อ **ครอบครองข้อมูลของคอมพิวเตอร์** เครื่องนั้น
 - Virus - โปรแกรมที่ได้รับการออกแบบมาเพื่อก่อความเสียหายในคอมพิวเตอร์
2. เป็นอิสระจากโปรแกรมอื่น
 - Malicious เริ่มทำงานเมื่อตัวมันถูกรัน
 - เช่น
 - Worm - การแพร่กระจายผ่าน network connection คัดลอกตัวเองและสามารถส่งตัวเองไปยังคอมพิวเตอร์เครื่องอื่นๆได้อย่างอิสระ
 - Zombie - ใช้คอมพิวเตอร์เครื่องอื่นๆ ที่ติด worm, trojan ฯลฯ แพร่กระจาย Malicious

Trusted Systems (ทำให้ระบบ naïve ถือว่า)

- Multilevel security
 - การกำหนด security ไว้หลายระดับ
 - กำหนดว่า Object ไหน ควรอยู่ level ไหน
 - กำหนด Policy ในการจัดการ
 - No read up - อ่านได้เฉพาะ Object ที่อยู่ level เท่ากับหรือต่ำกว่า ของตนเอง
 - No write down - เขียนได้เฉพาะ Object ที่อยู่ level เท่ากับหรือมากกว่า ของตนเอง