ระบบปฏิบัติการมีบทบาทหน้าที่หลักอะไรบ้าง

1. **Referee** ตัดสิน, ชี้ขาด, ห้ามฝ่าฝืนไม่งั้นระบบล่ม
   - Resource allocation among users, applications ต้องการ Resource ในการทำงาน โดย OS จะตัดสินว่า Resource ไหน ให้ Program ใด
   - Isolation of different users, applications from each other
   - Communication between users, application

2. **Illusionist** สร้าง illusion ขึ้นมาว่า program ที่กำลัง run อยู่สามารถใช้งานได้เต็มกำลัง ใช้ได้ทั้งเครื่อง

3. **Glue** เชื่อม library เข้าด้วยกัน (ระหว่าง interface, Libraries)

ระบบปฏิบัติการคืออะไร

ระบบปฏิบัติการ เป็นโปรแกรมที่ทำงานเป็นตัวกลางระหว่างผู้ใช้กับเครื่องคอมพิวเตอร์ และฮาร์ดแวร์ โดยมีวัตถุประสงค์เพื่อจัดสภาพแวดล้อมให้ผู้ใช้ระบบสามารถปฏิบัติงานบนเครื่องคอมพิวเตอร์ได้ โดยจะเอื้ออำนวยการพัฒนาและการใช้โปรแกรมต่างๆ รวมถึงการจัดสรรทรัพยากรต่างๆ ให้ได้อย่างมีประสิทธิภาพ

Operating System Evaluation

- **Reliability**: ต้องมีความเสถียร
- **Availability**: ต้องพร้อมใช้งานตลอดเวลา
- **Security**: ป้องกันพวกไวรัส
- **Privacy**: แบ่งแยกไฟล์ของแต่ละ ex.ไฟล์ของ application ใดก็มีแต่ app นั้นๆที่สามารถเข้าถึงได้
- **Portability**: สามารถเคลื่อนย้ายโปรแกรมได้
  - For programs ต้องพิจารณา API, Abstraction virtual machine
  - For OS "hardware abstraction layer" เป็นการสร้าง Abstraction ที่ใช้ในการอ้างอิงระหว่างฝั่ง hardware กับฝั่งที่เป็นคนเขียน operating system
- **Performance**:
  - Latency: ความเร็วในส่วนของ response
  - Throughput: จำนวนงานที่ทำได้ per time unit
  - Overhead: ให้ overhead น้อย
  - Fairness: ความ fair ของการใช้งานของแต่ละ program
  - Predictability

ประเภทของ Operating System

1. **Single Program Operating System** เป็น OS ที่อนุญาติให้ผู้ใช้เพียงคนเดียวทำงานกับคอมพิวเตอร์ โดยที่ ในเวลาหนึ่ง ๆ ผู้ใช้คนนั้นจะสามารถทำงานได้กับโปรแกรมเพียงโปรแกรมเดียว

2. **Multitasking Operating System** เป็น OS ประเภทที่อนุญาติให้ผู้ใช้ ใช้งาน มากกว่า 1 โปรแกรมพร้อมกันได้ในเวลาเดียวกันได้ ซึ่งในความเป็นจริงแล้ว ใน CPU ตัวหนึ่ง จะมีความสามารถในการทำงานให้กับโปรแกรมเพียง 1 โปรแกรมในช่วงเวลาหนึ่ง ๆ แต่ Multitasking OS นั้นมีความสามารถในการสลับการทำงานไปมาระหว่าง โปรแกรม ต่าง ๆ ได้

3. **Multiprocessing Operating System** วิธี multiprocessing เป็นการนำ CPU มากกว่า 1 ตัว เข้ามาทำงานร่วมกัน และหน้าที่หลักของ Multiprocessing Operating System คือ ประสานการทำงาน ของ CPU เหล่านี้ เพราะว่า CPU แต่ละตัว ใน multiprocessor computer นั้นสามารถทำงานได้ 1 คำสั่งในเวลาหนึ่ง ๆ เมื่อมี CPU หลายตัว เราก็สามารถทำงาน ได้ หลายคำสั่งมากขึ้นในเวลาเดียว

4. **Virtual Machine (VM)** เป็น OS ที่อนุญาติให้ คอมพิวเตอร์ตัวเดียวสามารถ run OS ทีละมากกว่า 1 OS. โดย VM จะจัดสรรอุปกรณ์ต่าง ๆ ในตัวคอมพิวเตอร์ให้กับ OS แต่ละตัวอย่างเป็นสัดส่วน ทำให้ผู้ใช้มองเห็นเครื่องคอมพิวเตอร์นี้เสมือนกับเป็นคอมพิวเตอร์ 2 ตัว ที่แยกกัน ข้อดีก็คือ องค์กรสามารถใช้ OS ที่ต่างกันและเหมาะสมกับงานต่าง ๆ กันได้ในเวลาเดียวกัน

หน้าที่ของ Kernel

หลักๆคือทำงานเป็นสื่อกลางในการเข้าถึงทรัพยากรของระบบ

1. **Central processing unit** ทำหน้าที่ควบคุมจัดการ program ที่กำลังทำงาน โดย Kernel จะรับผิดชอบในการตัดสินใจว่า program แต่ละตัวจะจองหน่วยประมวลผล core ไหน และ กี่ core ในการทำงาน

2. **Random-access memory** ใช้ในการเก็บข้อมูลของ program ที่ใช้งาน ซึ่งโดยปกติจะมี program จำนวนมากเข้ามาใช้งานตลอดเวลาตามความต้องการของแต่ละ application ซึ่ง Kernel มีหน้าที่ตัดสินใจว่า memory ส่วนไหนที่ process แต่ละอันสามารถใช้งานได้ และ ควรทำอย่างไรเมื่อ memory ไม่เพียงพอ

3. **Input/Output(I/O)** devices I/O ของแต่ละอุปกรณ์ เช่น keyboard, mouse, disk, printer, network adapter หรือ จอ monitor ทั้งหมดนี้ Kernel จะควบคุมการสื่อสารระหว่าง application และ hardware ให้

ประเภทของ Kernel

1. **Monolithic Kernels process** และการจัดการ memory จะถูกรวมอยู่ใน module เดียวกันภายใน kernel ซึ่งเป็นผลทำให้ Kernel มีขนาดใหญ่ และ ยากต่อการดูแล ภายหลังจึงได้มีการแยก module ออกมาและทำการเลือก load ใช้งานตามความเหมาะสม เป็นเสมือน extension ให้ OS เลือกใช้ ทำให้ไม่ต้องทำการปิดและ compile ใหม่ทั้งหมด เมื่อมีการแก้ bug

2. **Microkernels** จากปัญหาในเรื่องขนาดของ Kernel ที่โตขึ้นเรื่อย ๆ ของ monolithic ทำให้มีการแยกส่วนของระบบพื้นฐาน เช่น driver, protocol stack, file system ออกมารันข้างนอก ทำให้ลดขนาดของ Kernel ลง และยังเพิ่ม security และ stability ให้กับ OS อีกด้วย โดยทั้งหมดจะทำงานในส่วนของ user space และทำงานบนระบบตามการเรียกใช้ของ program

3. **Hybrid kernels** ถูกนำมาใช้งานกับ OS ระดับ commercial มีลักษณะคล้าย microkernel ยกเว้นแต่ว่ามันได้รวมเอา code เสริมใน kernel space มาเพิ่มความสามารถโดยใช้เป็น extension ให้กับ microkernel ด้วยคุณสมบัติของ monolithic kernel ซึ่งต่างจาก monolithic แท้ๆเพราะอันนั้นไม่สามารถ load module ในขณะทำงานได้ เพราะฉะนั้นจึงสรุปได้ว่า Hybrid kernel เป็น microkernel ที่มี code เสริมบางอย่างบน kernel space ที่ช่วยทำให้ทำงานได้ไวขึ้น

การป้องกันระดับฮาร์ดแวร์ Hardware Protection

1. การดำเนินการโหมดคู่กัน (Dual-Mode Operation)

เพื่อประกันความถูกต้องของการปฏิบัติการและทุกโปรแกรมตลอดทั้งข้อมูลของโปรแกรมเหล่านั้นจากการรุกรานของโปรแกรมผิดปกติ การปกป้องนี้มีความจำเป็นต้องใช้ โหมด (modes) ในการปฏิบัติการ ได้แก่

- โหมดผู้ใช้ (user mode)
- โหมดมอนิเตอร์ (monitor mode)

ทั้งสองโหมดจะใช้ฮาร์ดแวร์เข้ามาช่วย โดยกำหนด mode bit ให้ monitor (0) และ user (1) ทำให้ปฏิบัติการกับคำสั่งบางอย่างจะสามารถทำได้ด้วยเฉพาะในฐานะของระบบ ปฏิบัติการเท่านั้นและบางคำสั่งจะทำได้ในฐานะของผู้ใช้ การออกแบบระบบปฏิบัติการโดยป้องกันคำสั่งระดับเครื่องที่อาจเป็นอันตรายอย่างเช่นคำสั่งจำพวก คำสั่งอภิสิทธิ์ (privileged instructions)โดยกำหนดให้ฮาร์ดแวร์จะยอมรับคำสั่งประเภทอภิสิทธิ์จากการปฏิบัติการใน monitor mode เท่านั้น ถ้ามีความพยายามที่จะเรียกใช้คำสั่งเหล่านี้จาก user mode, ฮาร์ดแวร์จะถือว่าเป็นการกระทำที่ผิดปกติและจะ trap ไปยังระบบปฏิบัติการทันที

2. การป้องกัน I/O (I/O Protection)

กำหนดให้ทุกคำสั่งเกี่ยวกับ I/O เป็นคำสั่งอภิสิทธิ์
ต้องประกันว่าโปรแกรมผู้ใช้จะไม่ได้รับอนุญาตให้ควบคุมคอมพิวเตอร์ในฐานะโหมดมอนิเตอร์ได้

3. การป้องกันหน่วยความจำ (Memory Protection)

ต้องปกป้อง interrupt vector (ตารางที่เก็บตัวชี้ไปยัง interrupt service) ไม่ให้ถูกแก้ไขค่าได้โดยโปรแกรมผู้ใช้
และปกป้องรูทีนบริการขัดจังหวะ (interrupt service routine) ในระบบปฏิบัติการไม่ให้ถูกแก้ไขได้
จุดมุ่งหมายก็เพื่อป้องกันการรุกล้ำระบบปฏิบัติการจากโปรแกรมผู้ใช้ และป้องกันโปรแกรมผู้ใช้จากการรุกล้ำของผู้ใช้คนอื่นทั้งโดยเจตนาและไม่ เจตนา ทำได้โดยจัดสรรหน่วยความจำออกเป็นส่วนๆ โดยยินยอมให้โปรแกรมผู้ใช้เข้าถึงได้เฉพาะพื้นที่ของตนเองที่ได้รับอนุญาต เท่านั้น ด้วยการใช้ register 2 ตัว

- **base register** เก็บค่าเริ่มต้นของหมายเลขตำแหน่งหน่วยความจำที่ยอมให้ใช้งานได้
- **limit register** เก็บค่าขนาดของพื้นที่หน่วยความที่จะยอมให้ใช้ได้

4. การป้องกันซีพียู (CPU Protection)

โดยที่เราจะต้องปกป้องโปรแกรมผู้ใช้ไม่ให้ติดบ่วงอยู่ในวังวนไม่รู้จบ (infinite loop) และไม่ยอมส่งคืนการควบคุมให้แก่ระบบปฏิบัติการ วิธีการนี้เราจะใช้ timer ซึ่งเป็นฮาร์ดแวร์ เข้ามาช่วย timer สามารถตั้งค่าการขัดจังหวะคอมพิวเตอร์ตามเวลาที่กำหนด ซึ่งอาจเป็นเวลาคงที่หรือแปรเปลี่ยนก็ได้โดยจะถูกใช้อัตราที่คงที่ของนาฬิกา และตัวนับ (counter) ซึ่งระบบปฏิบัติการจะเป็นผู้ตั้งค่าตัวนับ

User Mode คือ

User Mode หรือ "โหมดผู้ใช้งาน" เป็นการเปิดใช้งานแอปพลิเคชันที่มีรันโค้ดคำสั่งผ่านระบบ API ที่สามารถเข้าถึงได้เฉพาะ Private Virtual Address Space และ Private Handle Table เท่านั้น โดยแต่ละแอปพลิเคชันก็จะมีการทำงานแยกส่วนกัน

Kernel Mode คือ

Kernel Mode เป็นการรันโค้ดคำสั่งที่ผู้ใช้สามารถควบคุมการเข้าถึงฮาร์ดแวร์, หน่วยความจำ, I/O และระบบต่าง ๆ ได้อย่างเต็มที่ โดยการทำงานของ Kernel Mode มักจะเป็นลำดับขั้น (Layer) จากสูงไปต่ำที่มีการใช้งาน Virtual Address Space ร่วมกัน

Process State

ใน Process State จะมีขั้นตอนที่เป็นส่วนประกอบอยู่ด้วยทั้งหมด 5 ส่วน ซึ่งได้แก่

1. new: บ่งบอกถึง ได้ทำการสร้าง หรือจองพื้นที่ใน memory แล้ว

2. ready: เป็นส่วนของการรอ processor ที่จะทำการดึงคำสั่งไปใช้ในการ execute

3. running: คำสั่งนั้น กำลังจะถูก execute

4. waiting: เป็นการรอคำสั่งอื่นให้เกิดขึ้นก่อน

5. terminate: process ได้ถูก execute เรียบร้อยแล้ว

Process Concept

แนวคิดของ process ที่โดยทั่วไป แนวคิดนั้นคือ โปรแกรมที่กำลัง execution อยู่ จะเรียงลำดับกันไปเรื่อย ๆ เพราะต้องดำเนินไปตามลำดับ ซึ่งจะประกอบไปด้วย 3 ส่วน นั่นคือ

1. program counter : นับจำนวนโปรแกรม

2. stack : เก็บคำสั่งในรูปการ stack

3. data section : ข้อมูลที่ใช้ในการทำงาน

Process Control Block

เรียกกันว่า PCB ซึ่งก็คือ ข้อมูลที่เกี่ยวข้องกันในแต่ละ process โดย OS จะเก็บไว้เพื่อติดตามการทำงาน ประกอบไปด้วย

1. Process state : บอกสถานะปัจจุบันของ Process

2. Process number : เพื่อใช้แยกแยะ Process

3. Program counter: เก็บ Address ของคำสั่งที่ทำการ execute

4. CPU registers : เก็บค่าต่างๆของ CPU ที่กำลัง Process

5. Memory-management information : จัดการหน่วยความจำ

6. Accounting information : บัญชีตัวข้อมูล

7. I/O status information : สถานะอุปกรณ์ I/O

Process Scheduling Queues

เป็นวิธีการจัดการ การทำงานของ process ให้ทำงานเป็นคิว โดยสามารถแยกย่อยได้ คือ

1. Job queue: ชุดการทำงานทั้งหมด ในระบบ

2. Ready queue: ชุดการทำงาน ที่ถูกพักเอาไว้ในหน่วยความจำ ซึ่งจะพร้อมรอสำหรับการ execute

3. Device queues: ชุดการทำงาน ที่รอคำสั่งจาก I/O device

Context Switch

ในการที่ CPU เปลี่ยนจากการทำงานงานหนึ่ง ไปอีกงานหนึ่ง หรือการสลับการทำงาน เราจะเรียกวิธีนี้ว่า context Switch ในการทำงาน เมื่อมีการเปลี่ยนงาน ระบบจะทำการบันทึกสถานะเดิมไว้แล้วจะทำการเปลี่ยนไปทำอีกงาน และเมื่อจะกลับมางานก่อนหน้าระบบก็จะทำการโหลดสถานะที่ได้บันทึก เพื่อนำมาทำงานต่อ ให้เกิดความต่อเนื่อง และจะสลับวิธีการนี้ไปเรื่อย ๆ เมื่อมีการเปลี่ยนตัวงาน

From user mode to kernel mode

Interrupts

- Triggered by timer and I/O devices

Exception

- Triggered by unexpected program behavior
- Or malicious behavior!

System calls (aka protected procedure call)

- Request by program for kernel to do some operation on its behalf
- Only limited # of very carefully coded entry points

From kernel mode to user mode

New process/new thread start

- Jump to first instruction in program/thread

Return from interrupt, exception, system call

- Resume suspended execution

Process/thread context switch

- Resume some other process

User-level upcall (UNIX signal)

- Asynchronous notification to user program

Device Interrupts

OS kernel needs to communicate with physical devices

Devices operate asynchronously from the CPU

- Polling: Kernel waits until I/O is done
- Interrupts: Kernel can do other work in the meantime

Device access to memory

- Programmed I/O: CPU reads and writes to device
- Direct memory access (DMA) by device
- Buffer descriptor: sequence of DMA's
    - E.g., packet header and packet body
- Queue of buffer descriptors
    - Buffer descriptor itself is DMA'ed

How do we take interrupts safely?

Interrupt vector

- Limited number of entry points into kernel

Kernel interrupt stack

- Handler works regardless of state of user code

Interrupt masking

- Handler is non-blocking

Atomic transfer of control

- "Single instruction"-like to change:
    - Program counter
    - Stack pointer
    - Memory protection
    - Kernel/user mode

Transparent restorable execution

- User program does not know interrupt occurred

The Kernel Stack

Solution: two-stack model

- Each OS thread has kernel stack (located in kernel

memory) plus user stack (located in user memory)

Place to save user registers during interrupt

Interrupt Stack

Per-processor, located in kernel (not user)

Memory

- Usually a process/thread has both: kernel and user stack

At end of handler

Handler restores saved registers

Atomically return to interrupted

process/thread

- Restore program counter
- Restore program stack
- Restore processor status word/condition codes
- Switch to user mode

Interrupt Masking

Interrupt handler runs with interrupts off

- Re-enabled when interrupt completes

OS kernel can also turn interrupts off

- E.g., when determining the next process/thread to run
- On x86
    - CLI: disable interrupts
    - STI: enable interrupt
    - Only applies to the current CPU (on a multicore)

Hardware support: Interrupt Control

Interrupt processing not visible to the user process:

- Occurs between instructions, restarted transparently
- No change to process state

Interrupt Handler invoked with interrupts 'disabled'

- Re-enabled upon completion
- Non-blocking (run to completion, no waits)
- Pack up in a queue and pass off to an OS thread for hard work
    - wake up an existing OS thread

OS kernel may enable/disable interrupts

- On x86: CLI (disable interrupts), STI (enable)
- Atomic section when select next process/thread to run
- Atomic return from interrupt or system call

HW may have multiple levels of interrupts

- Mask off (disable) certain interrupts
- Certain Non-Maskable-Interrupts (NMI)

Creating and managing processes

1. การสร้างโพรเซส (Process Creation) โพรเซสใด ๆ สามารถสร้างโพรเซสใหม่ได้ด้วยการเรียกใช้คำสั่งระบบ (System command) ของ OS หรือผ่านทาง System call ที่ชื่อ Fork โพรเซสที่สร้าง โพรเซสอื่นเรียกว่า โพรเซสแม่ เมื่อสร้างโพรเซสลูกแล้วสามารถทำงานต่อไปพร้อมกับโพรเซสลูก หรือหยุดรอจนกว่าโพรเซสลูกจะทำงานเสร็จ โพรเซสที่ถูกสร้างเรียกว่า โพรเซสลูก สามารถร้องขอทรัพยากรจากOS หรือถูกจำกัดให้ใช้ได้เฉพาะทรัพยากรของโพรเซสแม่เท่านั้น ทั้งนี้โพรเซสแม่จำเป็นต้องทราบหมายเลขโพรเซสลูกทั้งหมด

2. การสิ้นสุดของโพรเซส (Process Termination) ในการทำลายโพรเซส โพรเซสจะสิ้นสุดลงเมื่อสิ้นสุดการทำงานในคำสั่งสุดท้าย และแจ้งให้ระบบปฏิบัติการลบมันออกไปโดยใช้ System call ที่ชื่อ Exit หรือยกเลิกโพรเซสเมื่อทำงานเสร็จสิ้น เมื่อโพรเซสแม่ทำงานเสร็จสิ้น โพรเซสลูกและโพรเซสที่ถูกสร้างโดยโพรเซสลูกจะต้องสิ้นสุดตามไปด้วย เมื่อโพรเซสแม่สิ้นสุดไปแล้วโพรเซสลูกทั้งหมดก็จะสิ้นสุดไปด้วย สิ่งที่เกิดขึ้นในลักษณะนี้เรียกว่า การสิ้นสุดเป็นขั้น ๆ **(Cascading termination)**

Performing I/O

- open, read, write, close

Communicating between processes

- pipe, dup, select, connect

Shell คืออะไร

Shell คือ โปรแกรมที่ทำหน้ารับคำสั่งจากผู้ใช้ส่งให้ kernel ของระบบปฏิบัติการ เป็น command interpreter ในแปลงคำสั่งที่ได้รับ ให้เป็นคำสั่งที่ระบบปฏิบัติการเข้าใจ ทำงานอยู่ระหว่างผู้ใช้กับ kernel มีด้วยกัน 2 แบบคือ

- **CLI – Command line interface** รับคำสั่งโดยข้อมูล text และแสดงผลในรูปแบบ text เช่นกัน
- **GUI – Graphical user interface** รับคำสั่งโดยอาศัย mouse และ รูปบนจอ monitor

UNIX Process Management

UNIX fork (โหลดโค้ดใหม่)

- system call to create a copy of the current process, and start it running

UNIX exec (โหลดโค้ดใหม่มาแทนที่โค้ดเก่า)

- system call to change the program being run by the current process

UNIX wait (Option กรณีที่ใช้ shell เป็น text)

- system call to wait for a process to finish

UNIX signal (Option กรณีที่ใช้ shell เป็น text)

- system call to send a notification to another process

Can UNIX fork() return an error?

- Ram เต็ม เนื้อที่ไม่พอให้สร้าง Address ใหม่

Can UNIX exec() return an error? Why?

- ไฟล์หายไป ไฟล์เรียกใช้งานไม่ถูกต้อง

Can UNIX wait() ever return immediately? Why?

- Wait ทำงานสัมพันธ์กับ process ที่ทำงานอยู่

Interprocess communication (IPC)

คือการสื่อสารระหว่าง processes

- **Processes** ที่ทำการประมวลผลอยู่ในระบบนั้นอาจเป็นได้ทั้ง independent process และ dependent process
- **Independent process** คือ process ที่ไม่เกี่ยวข้องกับ process อื่น เช่น ไม่มีผลกระทบ ไม่มีการใช้ข้อมูลร่วม
- **Dependent process** คือ process ที่มีความเกี่ยวข้องกับ process อื่นๆ เช่น ผลลัพธ์มีผลกับ process อื่น หรือข้อมูลต้องใช้ร่วมกับ process อื่นเป็นต้น

Producer-consumer

Output of one program is accepted as input of another program

- One-way communication
- Pipe

Client-Server

การสื่อสารกันระหว่าง processes ไม่ได้เกิดขึ้นเฉพาะภายในคอมพิวเตอร์เดียวกันเท่านั้น แต่ยังมีการสื่อสารของ processes ระหว่างคอมพิวเตอร์ที่เชื่อมต่อกันในเครือข่าย ซึ่งการสื่อสารเป็นรูปแบบ client-server communication

Client-Server communication มี 3 รูปแบบคือ Sockets, Remote Procedure Calls (RPCs) และ Pipes

File system

- Write data to a file then read file as an input
- Reader and writer are not need to running at the same time

Concurrency

"ภาวะความพร้อมกัน" หมายถึงการที่ทรานแซคชันหลาย ๆ ทรานแซคชันต้องการเรียกใช้ข้อมูลเดียวกัน และในเวลาเดียวกันจากฐานข้อมูล อาจจะกล่าวให้เข้าใจได้ง่าย ๆ ก็คือ "ภาวะความพร้อมกัน" ก็คือกระบวนการสำหรับการควบคุมข้อมูลที่ถูกเรียกใช้โดยแต่ละงานให้มีความถูกต้องอยู่เสมอนั่นเอง

Parallel

เป็นการทำงานหลาย ๆ งานในเวลาเดียวกัน นั่นคือเรื่องของทำงาน หรือ ประมวลผล (Execution) เป้าหมายหลักเพิ่มประสิทธิภาพของการทำงาน **Parallel hardware** ประกอบไปด้วย multi-core processor, GPU และ computer cluster เป็นต้น

Threads in the Kernel and at User-Level

Multi-threaded kernel

- multiple threads, sharing kernel data structures, capable of using privileged instructions

Multiprocess kernel

- Multiple single-threaded processes
- System calls access shared kernel data structures

Multiple multi-threaded user processes

- Each with multiple threads, sharing same data structures, isolated from other user processes

Thread

Thread คือ ส่วนประกอบย่อยของโปรเซส ถ้า thread ที่เป็นส่วนประกอบย่อยจะเรียกว่า **Lightweight process (LWP)** แต่ถ้าโปรเซสดั้งเดิมที่มีการควบคุมเพียง 1 thread แสดงว่าทำงานได้เพียง 1 งานจะเรียกว่า **Heavyweight process** โดยปกติ Process ที่มี 1 thread จะเรียกว่า Single thread แต่ถ้า 1 process มีหลาย thread จะเรียกว่า Multithread เพราะ ใน Process หนึ่งอาจมีได้หลาย Thread เช่น Web browser 1 หน้าอาจมีทั้งการ download ข้อมูลพร้อมกับการแสดง text แสดงรูปภาพ หรือ java มาแสดงในหน้าเดียวกัน Thread มี 3 รูปแบบ

1. User-level threads
2. Kernel-level threads
3. Combining user and Kernel-level threads

แต่ละ Thread ประกอบไปด้วย

- **Thread ID** หมายเลข Thread ใน process
- **Program counter** ใช้นับคำสั่งที่ประมวลผลเป็นลำดับ
- **Register set** ใช้เก็บค่าที่ทำงานอยู่
- **Stack** ใช้เก็บประวัติการประมวลผล

Synchronization

คือการทำงานของprocessที่มีความเกี่ยวข้องกันไม่ว่าจะมีปัจจัยเริ่มต้นเป็นการใช้ทรัพยากรระบบร่วมกันหรืออาจจะเป็นการรอให้processหนึ่งเริ่มทำงานหลังจากอีก process หนึ่งทำงานเสร็จสิ้นแล้วก็ตาม ทั้งนี้ในการทำงานยังต้องอาศัยการ synchronize ที่เหมาะสมเพื่อให้มีการทำงานที่ถูกต้องซึ่งแน่นอนว่าไม่ใช่เรื่องง่ายเลยแม้ว่าจะมีแก้ปัญหาด้วยวิธีต่างๆมาแล้วเช่น Test and Set Instruction, Swap Instruction เป็นต้น

Semaphore

จากวิธีการแก้ปัญหาทั้งหมดก็ยังไม่สะดวกที่จะช่วยแก้ไขปัญหาที่มีความซับซ้อนมากได้ ดังนั้นจึงได้มีวิธีใหม่ถูกคิดค้นขึ้นมานั่นก็คือเซมาฟอร์ซึ่งเซมาฟอร์เป็นอัลกอริทึมหนึ่งที่ง่ายต่อการจัดการทรัพยากรให้process หากดูตามนิยามแล้วเซมาฟอร์แทนได้ด้วย 'S' ซึ่งจะมีค่าเป็นเลขจำนวนเต็มและมีการเรียกใช้งานได้จาก 2 คำสั่งคือ Signal แทนด้วย V ย่อมาจาก Verhogen และ Wait แทนด้วยP ย่อมาจาก Proberen ในการใช้งาน semaphore เราสามารถใช้เซมาฟอร์ในการแก้ไขปัญหาได้เช่น เมื่อพิจารณา process2 ตัวที่ทำงานพร้อมกันโดยให้ p1 มีสภาวะการทำงานที่ statement 1 และ p2 มีสภาวะการทำงานที่ statement 2

Locks

การประมวลผลพร้อมกัน และมีคุณสมบัติการไม่เกิดร่วม โดยวิธีการทางฮาร์ดแวร์สามารถทำได้

หลายวิธีดังนี้

1. การปิดกั้น (lock)
2. การปิดทางขัดจังหวะ (disabling interrupt)
3. คำสั่งทดสอบและเซ็ต (test and set instruction)

การปิดกั้น (Lock)

เมื่อโพรเซสต้องการเข้าไปทำงานในส่วนวิกฤติโพรเซสจะต้องตรวจสอบก่อนว่า ส่วนวิกฤติถูกล็อค หรือถูกปิดกั้นหรือไม่ หากพบว่าส่วนวิกฤติไม่ถูกล็อค แสดงว่าขณะนั้นไม่มีโพรเซสใดกำลังทำงานในส่วนวิกฤติโพรเซสสามารถเข้าไปทำงานในส่วนวิกฤติได้โดยก่อนที่จะเข้าไปทำงานในส่วนวิกฤติโพรเซสต้องใส่ล็อคเพื่อเป็นการปิดกั้นไม่ให้โพรเซสอื่นเข้าไปทำงานในส่วนวิกฤติได้จนกระทั่งโพรเซสทำงานในส่วนวิกฤติเสร็จเรียบร้อยจึงปลดล็อคเพื่อเปิดโอกาสให้โพรเซสอื่นสามารถเข้าทำงานในส่วนวิกฤติได้

การปิดทางขัดจังหวะ (Disable Interrupt)

สาเหตุหนึ่งที่ทำให้เกิดปัญหาในการทำงานพร้อมกันหลายโพรเซสคือ ขณะที่โพรเซสหนึ่งกำลังทำงานในส่วนวิกฤติอาจมีโพรเซสอื่นขอขัดจังหวะ เพื่อให้ได้โอกาสเข้าไปทำงานในส่วนวิกฤติเป็นผลให้โพรเซสที่กำลังทำงานในส่วนวิกฤติต้องหยุดทำงาน โดยที่การทำงานยังไม่เสร็จสมบูรณ์และสลับให้ โพรเซสอื่นเข้าไปทำงานในส่วนวิกฤติแทน โดยที่โพรเซสที่เข้าทำงานภายหลัง อาจแทรกแซงการทำงานหรือเปลี่ยนแปลงค่า หรือมีการประมวลผลใด ๆ ที่ส่งผลกระทบต่อผลลัพธ์ในการทำงานของโพรเซสแรก

คำสั่งทดสอบและเซต (Test and Set Instruction)

วิธีนี้พยายามแก้ปัญหาของวิธีการปิดกั้น และการปิดทางขัดจังหวะด้วยการใช้คำสั่งทดสอบ

และเซต ซึ่งเป็นคำสั่งระดับฮาร์ดแวร์โดยการทำงานของคำสั่งไม่สามารถถูกขัดจังหวะได้การทำงานของวิธีนี้คือ โพรเซสที่ต้องการเข้าไปทำงานในส่วนวิกฤติจะต้องเรียกใช้คำสั่งทดสอบและเซตเพื่อตรวจสอบว่ามีโพรเซสใดทำงานอยู่ในส่วนวิกฤติหรือไม่ หากพบว่าขณะนั้นไม่มีโพรเซสใดทำงานในส่วนวิกฤติโพรเซสจะเซตค่าล็อค เพื่อเป็นการป้องกันไม่ให้โพรเซสอื่นเข้าไปทำงานในส่วนวิกฤติพร้อมกันได้ทำให้ระบบสามารถมีโพรเซสจำนวนมากทำงานร่วมกัน โดยมีคุณสมบัติการไม่เกิดร่วม

Synchronization Problem

Synchronization Problem แบบต่าง ๆ ทั้งนี้สามารถที่จะนำเซมาฟอร์นี้นำไปใช้แก้ปัญหาพื้นฐานที่ใช้ทดสอบวิธีการแก้ปัญหาการประสานเวลาให้ตรงกันได้ เช่น ปัญหาการ Bounded-Buffer, ปัญหาการ Readers – Writers และปัญหาการทำ Dining-Philosophers ที่มีความสำคัญหลัก เนื่องจากปัญหาเหล่านี้คือตัวอย่างปัญหาหลักพื้นฐานที่ถูกใช้ในการทดสอบเกือบจะทุกโครงการเลยทีเดียว

Scheduling

รู้จักกับ **Processes Scheduling** ก่อนอื่นขออธิบายถึงเนื้อหาก่อน เรื่องนี้อยู่ในวิชา OS ซึ่งคนทั่วไปก็จะไม่ค่อยได้เรียนนัก น่าจะต้องเป็นสาย IT จึงจะได้เรียนวิชานี้ ผมจะอธิบายคร่าวๆ ว่ามันคืออะไร โดยปกติแล้ว ในคอมพิวเตอร์จะมี Process หรือเรียกว่างาน เกิดขึ้น จะเกิดจากผู้ใช้หรือการทำงานของคอมพิวเตอร์นี่แหละ และแน่นอนว่ามันมีจำนวนมากพอตัวเลย เป้าหมายที่มันเกิดมา คือมันต้องการประมวลผล โดย CPU ดังนั้นจึงทำให้เกิดการแย่งชิงเพื่อเข้าใช้งาน CPU แต่ทว่าเจ้า CPU เนี่ย มันก็ทำได้ทีละอย่าง บางงานก็ต้องทำนาน บางงานก็ทำแปบเดียว บางงานก็สำคัญมาก บางงานไม่จำเป็นต้องทำตอนนี้ ดังนั้นจึงต้องมีวิธีจัดการอย่างมีประสิทธิภาพ ซึ่ง OS จะต้องเลือกวิธีการจัดการงาน เรียกว่า CPU Scheduler (จัดการงานของ CPU) หลักๆเลย OS มันจะต้องจัดการงาน (Process) ให้กับ CPU ซึ่งมันก็มี วิธีการจัดการ (Scheduling Algorithms) หลายอย่างซึ่งจะเหมาะกับสถานการต่างๆดังที่กล่าวไป

ประเภทของ CPU scheduler

สามารถแบ่ง **CPU scheduler** ได้ 2 แบบคือ **Preemptive scheduling** (โดนแทรกแซงได้) กับ **Nonpreemptive scheduling** (ไม่โดนแทรกแซง) อันนี้เข้าใจได้ง่ายเลยคือ ถ้าเป็นแบบ Preemptive จะถูก process อื่นเข้ามาแย่ง CPU ได้ เช่น CPU เห็นว่ามี process ที่สำคัญกว่ายิ่งยวดมาใหม่ในคิว ก็จะลัดคิวให้เลย process ที่กำลังทำอยู่จะถูกหยุดชั่วคราว คล้ายๆกับนักเรียน กำลังรอสั่งอาหารแล้วครูมา นักเรียนก็สละให้ครู (ก็ครูปกครอง ไม่สละได้ไง) ส่วนประเภท Nonpreemptive ก็จะไม่มีการลัดคิวเลย ครูก็ต้องไปต่อแถวนั่นเอง

Switching context

การที่ CPU เปลี่ยนไปเป็นอีกงานหนึ่ง คือ การสลับ เรียกว่า **switching context** ซึ่งทั้ง preemptive หรือ nonpreemptive ก็เกิดการสลับได้ทั้งนั้น (ไม่สลับก็ทำได้งานเดียวสิ) แต่ Preemptive scheduling (โดนแทรกแซงได้) อาจจะทำให้เกิดการสลับไปสลับมาของ process ได้ เข้าๆออกๆ จำนวนมากได้ ค่า switch นี้ยิ่งมีค่ามากก็จะแสดงว่า CPU ต้องสลับงานบ่อยนั่นเอง

Scheduling Criteria

OS จะต้องจัดการงานต่างๆให้กับ CPU ดังนั้นจึงต้องมีการประเมินความสามารถของกระบวนการพวกนี้เพื่อจะสามารถบอกได้ว่า ที่ทำอยู่ว่าดีแค่ไหน ดีที่สุดหรือยัง โดยจะมีการคำนวณค่าดังนี้

1. **CPU utilization** คือ ความสามารถใช้งาน CPU ได้อย่างเต็มประสิทธิภาพ มีค่าเป็น % ถ้าทำงานเต็ม 100 คือ ทำงานเต็มประสิทธิภาพที่มันทำได้ แต่ถ้าทำ 100 ตลอด มันก็จะเหนื่อยทำให้อายุการใช้งานสั้นลง ดังนั้น OS จึงให้ประมวลผลเต็มประสิทธิภาพที่สุดโดยที่ไม่กระทบกับ CPU จนเกินไปด้วย
2. **Throughput** คือ จำนวนงานที่ทำได้ในช่วงเวลาหนึ่ง ยิ่งมากก็ยิ่งดี เพราะจะหมายความว่าทำงานได้เยอะ
3. **Turnaround time** คือ เวลาทั้งหมดที่ process หนึ่งต้องใช้ นับตั้งแต่เริ่มต้นทำงานจนเสร็จเลย นับเวลารอด้วย ดังนั้นบางงานได้ทำแรกสุด แต่ก็อาจต้องรอ จนทำเสร็จอันสุดท้ายก็ได้ เพราะโดนงานอื่นมาแทรก
4. **Waiting time** คือ เวลาที่ process ต้องรอ เช่น พอมาถึง ก็ไม่สามารถประมวลผลได้ เพราะต้องรองานที่มาก่อนทำเสร็จก่อน
5. **Response time** คือ เวลามีผลลัพธ์ออกมาครั้งแรก หลังจาก process แรกเข้าไป

Scheduling Algorithms

มาถึงพระเอกของเรื่องแล้วละ จากที่ OS เลือกว่าจะอนุญาตให้งานอื่นมากแทรกได้มั้ย (preemptive VS nonpreemtive) OS ก็จะต้องเลือกกระบวนการ (Algorithm) ด้วย ซึ่งจะมีกระบวนการหลายแบบ ให้เลือกสรรเลยละ แต่ละแบบก็มีข้อดี ข้อเสียต่างกัน เหมาะกับสถานการณ์ต่างๆกันไป แต่บาง Algorithm ก็ต้องใช้กับ nonpreemtive หรือ preemptive เท่านั้นนะ โดยทั่วไปจะมีที่นิยมยกตัวอย่างอยู่ 4 แบบ

First Come First Served Scheduling (FCFS)

สำหรับตัวแรกง่ายมาก คือมาก่อนก็ได้ก่อน อันนี้จะไม่มีการแทรกแซงของงานอื่นเลย (nonpreemtive) จัดเป็นวิธีที่ง่ายที่สุดแล้วละ แค่มี Queue สักตัวก็ทำได้แล้ว ซึ่งเป็นข้อดีของมัน แต่ว่าถ้ามี process นิงมาถึงก่อน แต่ทำงานนานมาก จะทำให้ process อื่นๆที่มารอ ต้องคอยนานมากด้วย ส่งผลให้ค่า Waiting time สูงไปด้วย แบบว่า ครูขึ้นลิฟท์ไปชั้น 40 นักเรียนจะขึ้นลิฟท์ไปชั้น 4 นี่เอง ต้องรอซะนานเลย ส่งผลให้ นักเรียนที่มาต่อแถวก็ต้องรอเพิ่มขึ้นไปอีก

Shortest Job First Scheduling (SJF)

JSF เป็นวิธีที่น่าสนใจ หลักการคือ จะให้งานที่จะใช้เวลาน้อยที่สุดทำงาน ถ้ามีงานมาถึงพร้อมกัน งานที่ใช้เวลาน้อยกว่าก็จะได้ทำก่อน เหมือนคนจะแย่งกันเข้าลิฟท์ คนตัวเล็กก็จะได้เข้าไปก่อนนั่นเอง การทำแบบนี้ทำให้ waiting time มีค่าน้อยกว่าแบบ FCFS มาก ค่าของ turn around time ก็น้อยกว่าอีกด้วย เพราะไม่ต้องรอมากนัก แต่ข้อเสียของมันคือ CPU ไม่มีทางรู้อนาคตว่างานที่มาจะใช้เวลาทำเท่าไหร่ ทำให้ SJF ใช้งานจริงไม่ได้ เป็นเหมือนแนวคิดในอุดมคติ หลักการของ SJF สามารถใช้ได้ทั้งแบบ preemtive และ nonpreemtive นะ

Priority Scheduling

วิธีการนี้จะนำความสำคัญมาจัดการ หากมี process มาพร้อมกัน OS ก็จะให้ process ที่สำคัญกว่าเข้าทำงานก่อน ตัวเลขความสำคัญ (Priority) ที่นิยมใช้คือ 1-5 โดยยิ่งน้อยยิ่งสำคัญ ดังนั้น process ที่มี priority เป็น 1 จะสำคัญที่สุด เรียกว่ายิ่งใหญ่สุดเลยละ ถ้ามาทุกคนต้องหลบ หลักการนี้สามารถใช้ได้ทั้งแบบ preemtive และ nonpreemtive วิธีการนี้ถือว่า ดีพอสมควรเพราะจะได้ค่าเฉลี่ยที่น่าพึงพอใจ งานที่ไม่สำคัญก็ทำทีหลังไปเลย แต่ทว่าอาจเกิดปัญหาที่เรียกว่า ถูกแช่แข็ง ได้ (Starvation) ปัญหานี้ก็คือ process ที่ไม่สำคัญรอคิวนานมากนานจนไม่มีวันได้ทำเลยละ เพราะถูกงานสำคัญๆ มาแย่งตลอด อาจารย์เล่าว่า มีคอมพิวเตอร์สมัยก่อนที่ทำงานมาหลายสิบปี แต่โปรแกรมเมอร์พึ่งพบว่าก็มีงานที่รอคิวอยู่ นานเป็นสิบๆปีเลย (น่าสงสารมากอะ) วิธีการแก้ก็คือ ถ้า process รอนาน ก็เพิ่ม priority ให้มันเรื่อยๆ (เหมือนเพิ่มยศให้มัน) แล้วเดี๋ยวมันจะได้ทำงานเอง

Round Robin Scheduling (RR)

round robin ตอนแรกผมนึกว่าคือชื่อคนคิดค้นซะอีก แต่ความจริงมันแปลว่า รอบวง ใช่แล้วครับ วิธีการนี้คือกำหนดให้ process ทำงานเป็นรอบๆ โดยจะกำหนดเวลาที่เท่าเทียมขึ้นมาเรียกว่า Quantum time โดย process ที่จะเข้าไปทำงาน จะทำงานแค่เวลาที่กำนด ใครทำไม่เสร็จก็ออกมาต่อแถวแล้วทำต่อ ส่วน process ไหนใช้เวลาน้อยกว่า Quantum time ก็จะสบายตัวไปไม่ต้องไปต่อแถวอีก ซึ่งเหมาะกับ CPU มีงานเข้ามาจำนวนมาก สิ่งสำคัญของ round robin คือ Quantum time ซึ่งหากกำหนดมากเกินไปก็จะไม่ต่างกับ FCFS โดยทั่วไปจะมีค่าประมาณ 10 – 100 ms

Address Translation

1. การย้ายตำแหน่ง (Relocation)
ระบบปฏิบัติการในปัจจุบัน ยอมให้โปรแกรมทำงานพร้อมกันได้หลายงานแบบ multiprogramming ซึ่งโปรเซสต่าง ๆ เข้าใช้งานหน่วยความจำร่วมกัน จึงต้องมีการสลับโปรแกรมให้เข้าออกหน่วยความจำได้ รวมถึงการเปลี่ยนแปลงค่าตำแหน่งในหน่วยความจำที่อ้างถึงในโปรแกรม ให้ถูกต้องตามตำแหน่งจริงในหน่วยความจำ เช่นโปรแกรม a อ้างถึงตำแหน่งที่ 1000 และโปรแกรม b ก็อ้างถึงตำแหน่งที่ 1000 เช่นกัน
ค่า address แบ่งได้ 2 ค่า

- **Absolute address** หมายถึง ตำแหน่งจริงของโปรเซสที่อยู่ในหน่วยความจำ
- **Relative address** หมายถึง ตำแหน่งของคำสั่ง หรือโปรแกรมของโปรเซสหลังจากการ compile

2. การป้องกันพื้นที่ (Protection)
ระบบปฏิบัติการควรสามารถป้องกันโปรเซส จากการถูกรบกวนทั้งทางตรง และทางอ้อม ดังนั้นก่อนให้โปรเซสใดเข้าครอบครองหน่วยความจำ จะต้องมีการตรวจสอบก่อน และใช้เวลาค้นหาเพื่อตรวจสอบตลอดเวลา

3. การใช้พื้นที่ร่วมกัน (Sharing)
การป้องกันเพียงอย่างเดียว อาจทำให้การใช้ทรัพยากรไม่คุ้ม จึงต้องมีการจัดสรรให้ใช้พื้นที่ของหน่วยความจำร่วมกันอย่างยืดหยุ่น

4. การจัดการแบ่งโปรแกรมย่อย (Logical organization)
ระบบปฏิบัติการจะแบ่งโปรแกรมเป็นโปรแกรมหลัก และโปรแกรมย่อย โดยนำเฉพาะโปรแกรมหลักลงในหน่วยความจำ แต่นำโปรแกรมย่อยลงหน่วยความจำเฉพาะเมื่อมีการเรียกใช้เท่านั้น

5. การจัดการแบ่งทางกายภาพ (Physical organization)
หน่วยความจำแบ่งเป็น 2 ส่วนคือ หน่วยความจำหลัก และหน่วยความจำสำรอง ลักษณะของหน่วยความจำหลักจะมีราคาแพง ทำงานได้เร็ว แต่เสื่อมหายได้ ในการทำงานจริง จึงต้องมีการเคลื่อนย้ายทางกายภาพระหว่างหน่วยความจำทั้ง 2 ตลอดเวลา ซึ่งเป็นหน้าที่ของระบบที่ต้องจัดสรรให้ให้สอดคล้องกับการทำงานแบบ multiprogramming

หน่วยความจำหลัก (Main memory)

การจัดการหน่วยความจำมีหลายระบบ เริ่มจากแบบไม่ซับซ้อน ไปถึงซับซ้อน ในบทนี้จะเรียนรู้แบบไม่ซับซ้อน ซึ่งไม่ถูกนำมาใช้งานในระบบปฏิบัติการปัจจุบัน แต่อาจใช้ในคอมพิวเตอร์ขนาดเล็กอยู่ การเรียนรู้เรื่องนี้อาจ นำไปประยุกต์ในการพัฒนางานด้าน software อื่น ๆ ได้

1. ระบบโปรแกรมเดียว (Monoprogramming) เป็นวิธีการจัดการที่ง่ายที่สุด โดยกำหนดเพียง 1 โปรแกรม ให้ทำงานในหน่วยความจำเพียงโปรแกรมเดียว

2. ระบบหลายโปรแกรมที่กำหนดขนาดพาร์ติชั่นคงที่ (Multiprogramming with fixed partition) ปัจจุบันระบบปฏิบัติการยอมให้มีหลายโปรเซสทำงานพร้อมกันได้ หมายความว่าขณะที่โปรเซสหนึ่งทำเสร็จ อีกโปรเซสที่รอ ก็เข้าใช้หน่วยความจำทันที โดยแบ่งหน่วยความจำออกเป็น partition การแบ่งเห็นแต่ละ partition แบบตายตัว มีจุดบกพร่อง จึงมีการจัดการหน่วยความจำแบบ FCFS : first come first serve การจัดการแบบนี้ย่อมมีปัญหา จึงเป็นหน้าที่ของระบบปฏิบัติการ ที่ต้องจัดการ

3. ระบบที่กำหนดขนาดของพาร์ติชั่นให้เปลี่ยนแปลงได้ (Dynamic partition) เพื่อแก้ปัญหาของการกำหนดแบบคงที่ จึงออกแบบการกำหนด partition แบบเปลี่ยนแปลงได้ ซึ่งมีความซับซ้อน มากขึ้น ตามโปรเซสที่เข้าใช้หน่วยความจำ

4. การจัดการแบบระบบบัดดี้ (Buddy system) ระบบจัดการหน่วยความจำแบบคงที่ และเปลี่ยนแปลงได้ อาจมีข้อจำกัดเรื่องการรองรับจำนวนโปรเซส และใช้พื้นที่ไม่เต็มประสิทธิภาพได้ ส่วนการแบ่งแบบเปลี่ยนแปลงได้ก็มีความซับซ้อนในทางปฏิบัติ และใช้ทรัพยากร จึงมีการผสมระหว่าง 2 ระบบนี้เข้าด้วยกัน ระบบนี้จะแบ่งพื้นที่เป็น 2 ส่วน และตรวจสอบว่าพอจะแบ่งให้กับโปรเซสล่าสุดได้หรือไม่ ถ้าแบ่งแล้วเข้าไม่ได้ ก็จะแบ่งขนาดเท่าก่อนหน้าที่ถูกแบ่ง และจัดเรียงพื้นที่เป็นแบบ link list ไว้รอการเข้าใช้ของโปรเซส

การจองพื้นที่ในหน่วยความจำหลัก มี 4 รูปแบบ

1. Single-user contiguous memory allocation ยุคแรกมีผู้ใช้คนเดียว โหลดไปไว้ในหน่วยความจำหลักหมด

2. Fixed Partitions ยุคมัลติโปรแกรมมิ่ง จะแบ่งส่วนที่ต้องอยู่กับส่วนที่เคลื่อนย้ายได้ โดยจัดการผ่าน Partition Manager มีปัญหา อาทิ เกิดพื้นที่ว่างใน Internal fragmentation มีขนาดใหญ่ เป็นต้น

3. Dynamic Partitions จะเปิดให้งานเข้าจองหน่วยความจำในขนาดที่ต้องการได้ แต่ก็ยังมีปัญหาขนาด External fragmentation เกิดขึ้น โดยแบ่งการจองพื้นที่เป็น 3 แบบ คือ First-fit, Best fit (เลือกพื้นที่ว่างเล็กสุด) และ worst-fit (เลือกพื้นที่ว่างใหญ่สุด)

4. Relocationable Dynamic Partitons จะมี Memory manager คอยจัดการขยับตำแหน่งการอ้างอิง ทำให้เหลือ block ใหญ่ที่เดียว เรียกว่า Compaction หรือ Garbage collection หรือ Defragmentation

หลักการของ **Overlay** คือ การแบ่งส่วนงานออกเป็นหลายส่วนตามการทำงาน นำเข้าหน่วยความจำหลักในเวลาที่ต้องใช้ และทับหน่วยความจำหลักเดิมที่ไม่ใช้ ซึ่งเป็นหลักการที่ใช้ใน Single-user contiguous memory allocation

Bound register คือ การเก็บตำแหน่งหน่วยความจำที่เก็บตำแหน่งสูงสุด หรือต่ำสุดที่มีการใช้งานหน่วยความจำหลัก แล้วอ้างอิงไปใช้ใน memory manager ไม่ให้ใช้เกินขอบเขตที่กำหนดไว้

Relocation register คือ การเก็บตำแหน่งงานว่างงานถูกเปลี่ยนไปจากตำแหน่งเดิมเท่าใด หากต้องการรู้ตำแหน่งปัจจุบันต้องใช้ตำแหน่งเดิม และตำแหน่ง Relocation register เพื่อใช้อ้างอิงในการนำมาใช้ต่อไป

Internal Fragmentation คือ งานที่มีขนาดใหญ่ เมื่อใช้ไปสักระยะ จะกินพื้นที่เล็กลง พื้นที่ ๆ เหลือ เรียกว่า การแตกกระจายภายใน (Internal Fragmentation)

External Fragmentation คือ งานที่ครอบครองหน่วยความจำเมื่อใช้งานเสร็จ และเลิกใช้ พื้นที่นั่นจะว่าง เรียกว่า การแตกกระจายภายนอก (External Fragmentation)

การแบ่งเป็นหน้า (Paging)

ระบบที่ใช้หน่วยความจำเสมือน (Virtual memory) มักใช้เทคนิคที่เรียกว่า การแบ่งหน้า หรือเพจจิ่ง(Paging) เพื่อเชื่อมระหว่างตำแหน่งทางตรรก กับเลขของเฟรมในหน่วยความจำหลัก

1. ตารางหน้า (Page table) หมายถึงการรับค่าตำแหน่งทางตรรกเป็นค่านำเข้า แล้วหาเลขของเฟรมในหน่วยความจำหลักออกมา เทคนิคตารางหน้า(Page table) มักมีขนาดใหญ่ และต้องทำด้วยความเร็วสูง หากสรุปแล้วตารางหน้า ก็คือตารางที่เก็บอาร์เรย์ของรีจิสเตอร์นั่นเอง

2. บัฟเฟอร์ค้นหาที่อยู่ (TLB : Translation lookaside buffer) ทุกครั้งที่เรียกใช้หน่วยความจำเสมือน ย่อมเรียกใช้หน่วยความจำหลัก 2 ครั้ง คือ การอ่านตารางหน้า และอ่านข้อมูลจริงจากหน่วยความจำหลัก ดังนั้นการทำงานแบบนี้จึงใช้เวลาถึง 2 เท่า จึงมีการใช้ cach memory ซึ่งทำหน้าที่เก็บตารางหน้าที่เคยถูกเรียกใช้ หรือเรียกว่า บัฟเฟอร์ค้นหาที่อยู่ (TLB : Translation Lookaside Buffer) ถ้าพบใน TLB จะเรียก TLB hit ถ้าไม่พบเรียก TLB miss และเข้าไปอ่านตารางหน้า เมื่ออ่านแล้วก็จะนำมาเพิ่มใน TLB สำหรับโอกาสที่จะถูกเรียกในอนาค

การแบ่งเป็นเซ็กเมนต์ (Segmentation)

การแบ่งหน้าจะแบ่งให้มีขนาดเท่ากัน แต่การแบ่งเป็นเซ็กเมนต์จะแบ่งโปรแกรมออกเป็นส่วน ๆ ไม่เท่ากัน และมีการใช้ตำแหน่งทางตรรก อ้างอิงตำแหน่งจริงเช่นกัน และมีการใช้ข้อมูล 2 ส่วนคือ เลขที่เซ็กเมนต์ และระยะเริ่มต้นของเซ้กเมนต์(Offset) สำหรับผลของการแบ่งเซ็กเมนต์ทำให้เกิดชิ้นส่วนไม่เท่ากัน (Dynamic partitioning) ซึ่งลดปัญหาการสูญเสียพื้นที่ **(Internal fragmentation)**

1. การนำวิธีการแบ่งเป็นเซ็กเมนต์มาใช้ในหน่วยความจำเสมือน
2. การรวมวิธีการแบ่งเป็นหน้ากับการแบ่งเป็นเซ็กเมนต์เข้าด้วยกัน

รูปแบบการจองพื้นที่หน่วยความจำเสมือน

1. Paged Memory Allocation (โหลดเข้าหน่วยความจำหลัก)

2. Demand Paging (โหลดเฉพาะที่ต้องใช้เข้าหน่วยความจำหลัก)

3. Segmented Memory Allocation (แต่ละเซกเมนต์ไม่จำเป็นต้องเท่ากัน)

4. Segmented/Demand Paged memory Allocation (แบ่งเป็นเซกเมนต์แล้วค่อยนำไปแบ่งเป็นเพจ)

Page frame คือ การแบ่งงานที่รองรับขนาดข้อมูลที่เท่า ๆ กันในแต่ละกรอบ โดย Page ที่ถูกวางข้อมูลไม่จำเป็นต้องเรียงชิดติดกัน ทำให้บริหารได้อย่างมีประสิทธิภาพ เพราะไม่มี Page ใดว่าง แต่จะเกิด Overhead ที่ต้องใช้ทรัพยากรในการจัดสรรให้เหมาะสม และเกิดปัญหา Internal Fragmentation และใช้พื้นที่เก็บ Job table และ Page map table

Segment frame คือ การแบ่งกรอบสำหรับรองรับข้อมูลตามขนาดของข้อมูล แต่ละกรอบงานจึงไม่เท่ากัน แต่จะแบ่งกรอบงานตามหน้าที่ เช่น โปรแกรมหลัก โปรแกรมย่อย1 โปรแกรมย่อย2 ไปในแต่ละเซกเมนต์ เมื่องานทำเสร็จแล้วก็จะมี External Fragmentation เกิดขึ้น ทำให้เกิด Overhead ในการเลือกพื้นที่สำหรับจอง Segment สำหรับงานใหม่

Virtual Memory

Virtual Memory คือ หน่วยความจำที่ถูกสร้างขึ้นมาในกรณีที่ RAM ไม่พอใช้ ซึ่งโดยส่วนมากมักเอาพื้นที่ในฮาร์ดดิสก์บางส่วนมาใช้แทน เนื่องจากหน่วยความจำของระบบมีจำกัดและมีราคาสูง เราจึงเรียกว่าความจำเสมือน การใช้หน่วยความจำเสมือนจะทำให้สามารถทำงานกับโปรแกรมขนาดใหญ่มาก ๆ ได้ โดยไม่มีปัญหาเรื่องหน่วยความจำไม่เพียงพอ ระบบการทำงานของหน่วยความจำเสมือนจะใช้วิธีแบ่งโปรแกรมออกเป็นส่วน ๆ และคอมพิวเตอร์จะทำการ สลับ (swap) ส่วนโปรแกรมที่ยังไม่ได้ใช้ลงไปยังหน่วยเก็บข้อมูลสำรอง และทำการสลับกลับมาในหน่วยความจำหลักเมื่อจำเป็นต้องใช้งาน ส่วนวิธีการใช้ประโยชน์จาก Virtual Memory ให้ได้ผลดีที่สุดจะต้องหาวิธีที่จะทำให้พื้นที่ที่กำหนดไว้ให้เป็น Swap File ซึ่งเป็นพื้นที่ที่อยู่บนฮาร์ดดิสก์ ทำให้การอ่านเขียนข้อมูลได้เร็วยิ่งขึ้น

Motivations for Virtual Memory

1. ใช้ DRAM ทางกายภาพเป็นแคชสำหรับดิสก์

- พื้นที่ที่อยู่ของกระบวนการสามารถเกินขนาดหน่วยความจำทางกายภาพ
- ผลรวมของพื้นที่ที่อยู่ของกระบวนการหลายเกินทางกายภาพหน่วยความจำ

2. จัดการหน่วยความจำลดความซับซ้อน

- มีถิ่นที่อยู่หลายกระบวนการในหน่วยความจำ แต่ละกระบวนการที่มีอยู่ของตัวเองเท่านั้น "งาน" รหัสและข้อมูลที่เป็นจริงในความทรงจำจัดสรรหน่วยความจำมากขึ้นในการดำเนินการตามที่จำเป็น

3. Provide Protection

- หนึ่งกระบวนการที่ไม่สามารถยุ่งเกี่ยวกับคนอื่นเพราะพวกเขาทำงานอยู่ในพื้นที่ที่แตกต่างกัน
- กระบวนการผู้ใช้ไม่สามารถเข้าถึงข้อมูลได้รับการยกเว้นส่วนต่าง ๆ ของช่องว่างอยู่มีสิทธิ์ที่แตกต่างกัน

Possibility of Thrashing

1. เพื่อรองรับกระบวนการมากที่สุดเท่าที่เป็นไปได้เพียงไม่กี่หน้าของแต่ละขั้นตอนจะถูกเก็บไว้ในหน่วยความจำ
2. แต่หน่วยความจำอาจจะเต็มเมื่อระบบปฏิบัติการนำหน้าหนึ่งในมันต้องสลับชิ้นหนึ่งออกมา
3. ระบบปฏิบัติการไม่ต้องสลับออกหน้าของกระบวนการก่อนที่หน้าเป็นสิ่งจำเป็น
4. ถ้ามันทำอย่างนี้บ่อยเกินไปนี้นำไปสู่thrashing: ประมวลผลใช้เวลาส่วนใหญ่สลับหน้าเว็บของตน ค่อนข้างกว่าการดำเนินการคำแนะนำของผู้ใช้

Process Execution(การดำเนินการกระบวนการ)

1. ระบบปฏิบัติการที่จะนำเข้ามาในหน่วยความจำเพียงไม่กี่หน้าของโปรแกรม (รวมถึงจุดเริ่มต้นของมัน)
2. รายการตารางแต่ละหน้าจะมีบิตปัจจุบัน (P บิต) ที่ถูกตั้งค่าเฉพาะในกรณีที่หน้าที่เกี่ยวข้องอยู่ในหน่วยความจำหลัก