

ระบบปฏิบัติการมีบทบาทหน้าที่หลักอะไรบ้าง

1. Referee ตัดสิน, ซี่จัด, ห้ามฝ่าฝืนไม่เข็นระบบล่ม

- Resource allocation among users, applications ต้องการ Resource ในการทำงาน โดย OS จะตัดสินว่า Resource ไหน ให้ Program ได้
- Isolation of different users, applications from each other
- Communication between users, application

2. Illusionist สร้าง illusion ขึ้นมาว่า program ที่กำลัง run อยู่สามารถใช้งานได้เต็มกำลัง ให้ได้ทั้งเครื่อง

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

ระบบปฏิบัติการที่օอจะไร

ระบบปฏิบัติการ เป็นโปรแกรมที่ทำงานเป็นตัวกลางระหว่างผู้ใช้ กับเครื่องคอมพิวเตอร์ และฮาร์ดแวร์ โดยมีวัตถุประสงค์เพื่อจัด สภาพแวดล้อมให้ผู้ใช้ระบบสามารถปฏิบัติงานบนเครื่อง คอมพิวเตอร์ได้ โดยจะเข้ามายกการพัฒนาและการใช้ โปรแกรมต่างๆ รวมถึงการจัดสรรทรัพยากรต่างๆ ให้ได้อย่างมีประสิทธิภาพ

Operating System Evaluation

- Reliability: ต้องมีความเสถียร

- Availability: ต้องพร้อมใช้งานตลอดเวลา

- Security: ป้องกันภัยไวรัส

- Privacy: แบ่งแยกไฟล์ของแต่ละ ex.ไฟล์ของ application ได้ มีแต่ app นั้นๆที่สามารถเข้าถึงได้

- Portability: สามารถเคลื่อนย้ายโปรแกรมได้

- For programs ต้องมี interface API, Abstraction virtual machine
- For OS "hardware abstraction layer" เป็นการสร้าง Abstraction ที่ใช้ในการอ้างอิงระหว่างผู้ใช้ hardware กับผู้ใช้ที่เป็นคนเขียน 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 ผู้ใช้คนนั้นจะสามารถทำงานได้กับโปรแกรมเดียว

2. Multitasking Operating System เป็น OS ประเภทที่อนุญาติให้ผู้ใช้ ใช้งาน มากกว่า 1 โปรแกรมพร้อมกันได้ในเวลาเดียวกันได้ ซึ่งในความเป็นจริงแล้ว ใน CPU ตัวหนึ่ง จะมีความสามารถในการทำงานให้กับโปรแกรมเดียว 1 โปรแกรม ในช่วงเวลาหนึ่ง 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 เมื่อสิ่ง CPU หลายตัว เราก็สามารถทำงานได้ หลายคำสั่งมากขึ้นในเวลาเดียว

4. Virtual Machine (VM) เป็น OS ที่อนุญาติให้ คอมพิวเตอร์ ตัวเดียวสามารถ run OS ที่ลงทะเบียนมากกว่า 1 OS. โดย VM จะจัดสรรอุปกรณ์ต่าง ๆ ในตัวคอมพิวเตอร์ให้กับ OS แต่ละตัว อย่างเป็นสัดส่วน ทำให้ผู้ใช้มองเห็นเครื่องคอมพิวเตอร์นี้เสมือนกับเป็นคอมพิวเตอร์ 2 ตัว ที่แยกกัน ข้อดีคือ องค์กรสามารถให้ 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 จำนวนมากเข้ามาใช้งาน ตลอดเวลาตามความต้องการของแต่ละ application ซึ่ง Kernel มีหน้าที่ตัดสินใจว่า memory ส่วนไหนที่ process แต่ละอัน สามารถใช้งานได้ 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 จะถูกรวบอยู่ใน module เดียวกันภายใน kernel ซึ่งเป็นผลทำให้ Kernel มีขนาดใหญ่ และ ยากต่อการดูแล ภายนลังจึงได้มีการแยก module ออกมาและทำการเลือก load ใช้งานตามความเหมาะสม เป็นเมโมรี่ extension ใน OS เลือกใช้ ทำให้ไม่ต้องทำการปิดและ compile ในทั้งหมด เมื่อมีการแก้ bug

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

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

การป้องกันระดับฮาร์ดแวร์ Hardware Protection

1. การดำเนินการโหมดคู่กัน (Dual-Mode Operation)

เพื่อประกันความถูกต้องของการปฏิบัติการและทุกโปรแกรมตลอดทั้งช่วงของโปรแกรมเหล่านี้ จากการรุกรานของโปรแกรมผิดปกติ การปักป่องนี้มีความจำเป็นต้องใช้ โหมด (modes) ใน การปฏิบัติการ ได้แก่

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

ทั้งสองโหมดจะใช้ฮาร์ดแวร์เข้ามาช่วย โดยกำหนด mode bit ใน monitor (0) และ user (1) ทำให้ปฏิบัติการกับคำสั่งบางอย่างจะสามารถทำได้ด้วยเฉพาะในฐานะของระบบ ปฏิบัติการเท่านั้นและบางคำสั่งจะทำได้ในฐานะของผู้ใช้ การออกแบบระบบปฏิบัติการโดยป้องกันคำสั่งระดับเครื่องที่อาจเป็นอันตรายอย่างเช่นคำสั่ง จำพวก คำสั่งอภิสิทธิ์ (privileged instructions) โดยกำหนดให้ ฮาร์ดแวร์จะยอมรับคำสั่งประเภทอภิสิทธิ์จากการปฏิบัติการใน monitor mode เท่านั้น ถ้ามีความพยายามที่จะเรียกใช้คำสั่งเหล่านี้จาก user mode, ฮาร์ดแวร์จะสืบว่าเป็นการกระทำที่ผิดปกติและจะ trap ไปยังระบบปฏิบัติการทันที

2. การป้องกัน I/O (I/O Protection)

กำหนดให้ทุกคำสั่งเกี่ยวกับ I/O เป็นคำสั่งอภิสิทธิ์

ต้องประกันว่า โปรแกรมผู้ใช้จะไม่ได้รับอนุญาตให้ควบคุมคอมพิวเตอร์ในฐานะโหมดมอนิเตอร์ได้

3. การป้องกันหน่วยความจำ (Memory Protection)

ต้องปักป่อง interrupt vector (ตารางที่เก็บตัวชี้ไปยัง interrupt service) ไม่ให้ถูกแก้ไขค่าได้โดยโปรแกรมผู้ใช้

และปักป่องรูทีนบริการขัดจังหวะ (interrupt service routine) ในระบบปฏิบัติการไม่ให้ถูกแก้ไขได้

จุดมุ่งหมายก็เพื่อป้องกันการรุกล้ำระบบปฏิบัติการจากโปรแกรมผู้ใช้ และป้องกันโปรแกรมผู้ใช้จากการรุกล้ำของผู้ใช้คนอื่นทั้งโดยเจตนาและไม่เจตนา ทำได้โดยจัดสรรหน่วยความจำออกเป็นส่วนๆ โดยยินยอมให้โปรแกรมผู้ใช้เข้าสิ่งได้เฉพาะพื้นที่ของตนเองที่ได้รับอนุญาต เท่านั้น ด้วยการใช้ register 2 ตัว

- base register เก็บค่าเริ่มต้นของหมายเลขอ้างหนึ่ง หน่วยความจำที่ยอมให้เข้าสิ่งได้
- limit register เก็บขนาดของพื้นที่หน่วยความจำที่จะยอมให้ได้

4. การป้องกันชิปยุ (CPU Protection)

โดยที่เราจะต้องปักป่องโปรแกรมผู้ใช้ไม่ให้ติดบล็อกอยู่ในจังหวะไม่รู้จบ (infinite loop) และไม่ยอมส่งคืนการควบคุมให้แก่ระบบปฏิบัติการ วิธีการนี้เราจะใช้ timer ซึ่งเป็นฮาร์ดแวร์ เข้ามาช่วย timer สามารถตั้งค่าการขัดจังหวะคอมพิวเตอร์ตามเวลาที่กำหนด ซึ่งอาจเป็นเวลาครึ่งหรือเปลี่ยนก็ได้ โดยจะถูกใช้วัดร้าที่คงที่ของนาฬิกา และตัวนับ (counter) ซึ่งระบบปฏิบัติการจะเป็นผู้ตั้งค่าตัวนับ

User Mode ศือ

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

Kernel Mode ศือ

Kernel Mode เป็นการรันโค้ดคำสั่งที่ผู้ใช้สามารถควบคุมการเข้าสิ่งฮาร์ดแวร์, หน่วยความจำ, I/O และระบบต่างๆ ได้อย่างเต็มที่ โดยการทำงานของ Kernel Mode อาจจะเป็นลำดับชั้น (Layer) จากสูงไปต่ำที่มีการใช้งาน Virtual Address Space ร่วมกัน

Process State

ใน Process State จะมีชั้นตอนที่เป็นส่วนประกอบอยู่ด้วยกัน 5 ส่วน ซึ่งได้แก่

1. new: บ่งบอกถึง ได้ทำการสร้าง หรือของพื้นที่ใน memory แล้ว
2. ready: เป็นส่วนของการรอ processor ที่จะทำการติดคำสั่งไปให้ใน การ execute
3. running: คำสั่งนั้นกำลังจะถูก execute
4. waiting: เป็นการรอคำสั่งนั้นให้เกิดขึ้นก่อน
5. terminate: process ได้ถูก execute เรียบร้อยแล้ว

Process Concept

แนวคิดของ process ที่โดยทั่วไป แนวคิดนี้คือ โปรแกรมที่กำลัง execution อยู่ จะเรียกว่า ลำดับกันไปเรื่อยๆ เพราต้องดำเนินไปตามลำดับ ซึ่งจะประกอบไปด้วย 3 ส่วน นั้นคือ

1. program counter: นับจำนวนโปรแกรม
2. stack: เก็บคำสั่งในรูปแบบ stack
3. data section: ข้อมูลที่ใช้ในการทำงาน

Process Control Block

เรียกว่า 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: เก็บค่าต่างๆ ของ 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 ให้ทำงานเป็นคิวโดยสามารถแยกย่อยได้ คือ

1. Job queue: ชุดการทำงานทั้งหมด ในระบบ
2. Ready queue: ชุดการทำงาน ที่ถูกพักเอาไว้ในหน่วยความจำ ซึ่งจะพร้อมรับการ execute
3. Device queues: ชุดการทำงาน ที่รอคำสั่งจาก I/O device

Context Switch

ในการที่ CPU เปลี่ยนจากการทำงานหนึ่ง ไปอีกงานหนึ่ง หรือ การสลับการทำงาน เราจะเรียกว่า context switch ใน การทำงาน เมื่อมีการเปลี่ยนงาน ระบบจะทำการบันทึกสถานะเดิมไว้ แล้วจะทำการเปลี่ยนไปทำงานใหม่ และเมื่อจะกลับมางานก่อนหน้า ระบบก็จะทำการโหลดสถานะที่ได้บันทึก เพื่อนำมาทำงานต่อ ให้เกิดความต่อเนื่อง และจะลับวิธีการนี้ไปเรื่อยๆ 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. การสร้างโปรเซส (Process Creation) โปรเซสใด ๆ สามารถสร้างโปรเซสใหม่ได้ด้วยการเรียกใช้คำสั่งระบบ (System command) ของ OS หรือผ่านทาง System call ที่ชื่อ Fork โปรเซสที่สร้าง โปรเซสใหม่เรียกว่า โปรเซสแม่ เมื่อสร้าง โปรเซสลูกแล้วสามารถทำงานต่อไปพร้อมกับโปรเซสลูก หรือ หยุดร่วมกันกับ โปรเซสลูกจะทำงานเสร็จ โปรเซสที่ถูกสร้าง เรียกว่า โปรเซสลูก สามารถร้องขอทรัพยากรจาก OS หรือถูก จำกัดให้ได้เฉพาะทรัพยากรของ โปรเซสแม่เท่านั้น ทั้งนี้ โปรเซสแม่จะเป็นต้องทราบหมายเลขอรุ่นของ โปรเซสลูกทั้งหมด

2. การสิ้นสุดของโปรเซส (Process Termination) ในการ กำจัด โปรเซส โปรเซสจะสิ้นสุดลงเมื่อสิ้นสุดการทำงานในคำสั่ง สุดท้าย และแจ้งให้ระบบปฏิบัติการลบมันออกไปโดยใช้ System call ที่ชื่อ Exit หรือยกเลิก โปรเซสเมื่อทำงานเสร็จสิ้น เมื่อ โปรเซสแม่ทำงานเสร็จสิ้น โปรเซสลูกและ โปรเซสที่ถูกสร้างโดย โปรเซสลูกจะต้องสิ้นสุดตามไปด้วย เมื่อ โปรเซสแม่สิ้นสุดไปแล้ว โปรเซสลูกทั้งหมดก็จะสิ้นสุดไปด้วย สิ่งที่เกิดขึ้นในลักษณะนี้ เรียกว่า การสิ้นสุดเป็นขั้น ๆ (Cascading termination)

Performing I/O

- open, read, write, close

Communicating between processes

- pipe, dup, select, connect

Shell คืออะไร

Shell คือ โปรแกรมที่ทำหน้าที่รับคำสั่งจากผู้ใช้ส่งให้ kernel ของ ระบบปฏิบัติการ เป็น command interpreter ในแปลงคำสั่งที่ ได้รับ ให้เป็นคำสั่งที่ระบบปฏิบัติการเข้าใจ ทำงานอยู่ระหว่างผู้ใช้ กับ kernel ผ่านทาง interface 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 (ໂໂລດໂຄດໃໝ່)

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

UNIX exec (ໂໂລດໂຄດໃໝ່ມາແກ່ນທີ່ໂຄດເກົ່າ)

- 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 เนื่องที่ไม่พอให้สร้าง 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 ที่ทำการประมวลผลอยู่ในระบบนี้ อาจ เป็นได้ทั้ง independent process และ dependent process
- Independent process คือ process ที่ไม่เกี่ยวข้องกับ process อื่น เช่น ไม่มีผลกระทบ ไม่มีการใช้ข้อมูลร่วม
- Dependent process คือ process ที่มีความเกี่ยวข้อง กับ 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 ไม่ได้เกิดขึ้นเฉพาะภายใน คอมพิวเตอร์เดียว กันเท่านั้น แต่ยังมีการสื่อสารของ processes ระหว่างคอมพิวเตอร์ที่เชื่อมต่อกันในเครือข่าย ซึ่งการสื่อสารเป็น รูปแบบ client-server communication

Client-Server communication มี 3 รูปแบบ คือ 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” หมายถึงการที่ทราบแซนชันหลาย ๆ ท่าน แซนชันต้องการเรียกใช้ข้อมูลเดียวกัน และในเวลาเดียวกันจาก ฐานข้อมูล อาจจะกล่าวให้เข้าใจได้ง่าย ๆ ก็คือ “ภาวะความพร้อม กัน” ก็คือกระบวนการสำหรับการควบคุมข้อมูลที่ถูกเรียกใช้โดย แต่ละงานในมีความถูกต้องอยู่เสมอในหนึ่งเดียว

Parallel

เป็นการทำงานหลาย ๆ งานในเวลาเดียวกัน นั่นคือเรื่องของ ทำงาน หรือ ประมวลผล (Execution) เป้าหมายหลักเพื่อ ประสิทธิภาพของการทำงาน Parallel hardware ประกอบไป ด้วย 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 ส่วนประกอบย่อยของโปรแกรม ถ้า thread ที่เป็นส่วนประกอบย่อยจะเรียกว่า **Lightweight process (LWP)** และถ้าโปรแกรมตั้งเดิมที่มีการควบคุมเพียง 1 thread แสดงว่าทำงานได้เพียง 1 งานจะเรียกว่า **Heavyweight process** โดยปกติ Process ที่มี 1 thread จะเรียกว่า Single thread แต่ถ้า 1 process มีหลาย thread จะเรียกว่า Multithread เพราะใน Process หนึ่งอาจมีได้หลาย Thread เช่น Web browser 1 หน้า อาจมีห้อง download ข้อมูลร่วมกับการแสดง text และรูปภาพ 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 หมายเลขอารบิกที่กำหนดให้กับ thread ใน process
- Program counter ใช้บันทึกตำแหน่งใน memory ที่ต้องการให้คำสั่ง
- Register set ให้เก็บค่าที่ทำงานอยู่
- Stack ให้เก็บประวัติการประมวลผล

Synchronization

คือการทำงานของprocessที่มีความเกี่ยวข้องกันไม่ว่าจะมีปัจจัยใดๆ ก็ตามที่มีผลต่อการทำงานของprocess ที่ต้องการร่วมกัน เช่น การเขียนข้อมูลลงในไฟล์เดียวกัน หรือการอ่านข้อมูลจากไฟล์เดียวกัน หรือการเขียนข้อมูลลงในฐานข้อมูลเดียวกัน หรือการอ่านข้อมูลจากฐานข้อมูลเดียวกัน เป็นต้น

Test and Set Instruction, Swap Instruction เป็นต้น

Semaphore

จากรากศัพท์ภาษาอังกฤษ Semaphores แปลว่า semaphore คือ เครื่องมือที่ใช้ในการจัดการทรัพยากร่วมกัน หรือการร่วมกันของ process ที่มีความซ้อนซ้อนกัน เช่น การเขียนข้อมูลลงในไฟล์เดียวกัน หรือการอ่านข้อมูลจากไฟล์เดียวกัน หรือการเขียนข้อมูลลงในฐานข้อมูลเดียวกัน หรือการอ่านข้อมูลจากฐานข้อมูลเดียวกัน เป็นต้น

Locks

การประมวลผลร่วมกัน และมีคุณสมบัติการไม่เกิดร่วม โดยวิธีการทาง hardware สามารถทำได้

หลายวิธีดังนี้

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

การปิดกั้น (Lock)

เมื่อโปรแกรมต้องการเข้าไปทำงานในส่วนวิกฤติ โปรแกรมจะต้องตรวจสอบก่อนว่า ส่วนวิกฤติถูกล็อก หรือถูกปิดกั้นหรือไม่ หากพบว่า ส่วนวิกฤติไม่ถูกล็อก และดังว่าขณะนี้ไม่มีโปรแกรมใดกำลังทำงานในส่วนวิกฤติ โปรแกรมจะดำเนินการต่อไป

ได้โดยก่อนที่จะเข้าไปทำงานในส่วนวิกฤติ โปรแกรมต้องไล่ล็อก เพื่อเป็นการปิดกั้นไม่ให้โปรแกรมอื่นเข้าไปทำงานในส่วนวิกฤติได้ จนกระทั่งโปรแกรมทำงานในส่วนวิกฤติเสร็จเรียบร้อยแล้วล็อกจะเปิดโอกาสให้โปรแกรมอื่นสามารถเข้าทำงานในส่วนวิกฤติได้

การปิดทางขัดจังหวะ (Disable Interrupt)

สาเหตุหนึ่งที่ทำให้เกิดปัญหาในการทำงานพร้อมกันหลายโปรแกรม คือ ขณะที่โปรแกรมนึงกำลังทำงานในส่วนวิกฤติอาจมีโปรแกรมอื่น ขอขัดจังหวะ เพื่อให้ได้โอกาสเข้าไปทำงานในส่วนวิกฤติเป็นผลให้ โปรแกรมที่กำลังทำงานในส่วนวิกฤติต้องหยุดทำงาน โดยที่การทำงานยังไม่เสร็จสมบูรณ์และลับให้ โปรแกรมอื่นเข้าไปทำงานในส่วนวิกฤติแทน โดยที่โปรแกรมที่เข้าทำงานภายหลัง อาจแทรกแซงการทำงานหรือเปลี่ยนแปลงค่า หรือมีการประมวลผลได้ ที่ส่งผลกระทบต่อผลลัพธ์ในการทำงานของโปรแกรมแรก

คำสั่งทดสอบและเซต (Test and Set Instruction)

วิธีนี้พิจารณาแก้ปัญหาของวิธีการปิดกั้น และการปิดทางขัดจังหวะ ด้วยการใช้คำสั่งทดสอบและเซต

และเซต ซึ่งเป็นคำสั่งระดับhardware โดยการทำงานของคำสั่งไม่สามารถถูกขัดจังหวะได้ การทำงานของวิธีนี้คือ โปรแกรมที่ต้องการเข้าไปทำงานในส่วนวิกฤติจะต้องเรียกใช้คำสั่งทดสอบและเซต เพื่อตรวจสอบว่ามีโปรแกรมใดทำงานอยู่ในส่วนวิกฤติหรือไม่ หากพบว่าขณะนี้ไม่มีโปรแกรมใดทำงานในส่วนวิกฤติ โปรแกรมจะเซตค่าล็อก เพื่อเป็นการป้องกันไม่ให้โปรแกรมอื่นเข้าไปทำงานในส่วนวิกฤติพร้อมกันได้ ทำให้ระบบสามารถมีโปรแกรมจำนวนมากทำงานร่วมกัน โดยมีคุณสมบัติการไม่เกิดร่วม

Synchronization Problem

Synchronization Problem แบบต่างๆ ที่มีสามารถที่จะนำเข้ามาฟอร์มได้ นำไปใช้แก้ปัญหาฟีนฐานที่ใช้ทดสอบวิธีการแก้ปัญหา การประสานเวลาให้ตรงกันได้ เช่น ปัญหาการ Bounded-Buffer, ปัญหาการ Readers – Writers และปัญหาการทำ Dining-Philosophers ที่มีความสำคัญมาก เนื่องจากปัญหาเหล่านี้ คือตัวอย่างปัญหาลักษณะที่มีฐานที่ถูกใช้ในการทดสอบ เกือบทุกโครงการเลยที่เดียว

Scheduling

รู้จักกับ Processes Scheduling ก่อนอื่นขออธิบายถึงเนื้อหา ก่อน เรื่องนี้อยู่ในวิชา OS ซึ่งคนที่จะเข้าใจก็จะไม่ค่อยได้เรียนนัก น่าจะต้องเป็นสาย IT จึงจะได้เรียนวิชานี้ ผู้จะอธิบายคร่าวๆ ว่ามันคืออะไร โดยปกติแล้ว ในคอมพิวเตอร์จะมี Process หรือ เรียกว่างาน เกิดขึ้น จะเกิดจากผู้ใช้หรือการทำงานของ

คอมพิวเตอร์นี้แหล่ง และแน่นอนว่ามันมีจำนวนมากพอตัวเลย เป็นหมายที่มันเกิดมา คือมันต้องการประมวลผล โดย CPU ตั้งนั้นจะทำการแบ่งเวลาให้กับการแย่งชิงเพื่อเข้าใช้งาน CPU แต่ทว่าเจ้า CPU เนี่ย มันก็ทำได้ทีละอย่าง บางงานก็ต้องทำงาน บางงานก็ทำแบบเดียว บางงานก็สำคัญมาก บางงานไม่จำเป็นต้องทำงานนี้ ตั้งนั้นจะต้องมีวิธีจัดการอย่างมีประสิทธิภาพ ซึ่ง OS จะต้องเลือกวิธีการจัดการงาน เรียกว่า CPU Scheduler (จัดการงานของ CPU) หลักๆ เลย OS มันจะต้องจัดการงาน (Process) ให้กับ CPU ซึ่งมันก็มีวิธีการจัดการ (Scheduling Algorithms) หลายอย่างซึ่งจะแนะนำกับสถานการณ์ต่างๆ ดังที่กล่าวไป

ประเภทของ CPU scheduler

สามารถแบ่ง CPU scheduler ได้ 2 แบบคือ Preemptive scheduling (โドนแทรกแข่งได้) กับ Nonpreemptive scheduling (ไม่โドนแทรกแข่ง) ขึ้นนี้เข้าใจได้ง่ายเลยคือ ถ้าเป็นแบบ Preemptive จะถูก process ที่เข้ามาแย่ง CPU ได้ เช่น CPU เห็นว่ามี process ที่สำคัญกว่าอยู่ด้วยมาใหม่ในคิว ก็จะลัดคิวให้เลี้ยง process ที่กำลังทำอยู่จะถูกหยุดชั่วคราว คล้ายๆ กับนักเรียน กำลังรอสิ่งอาหารแล้วครุ่น นักเรียนก็จะลัดหน้าครู (ครูปากรอง ไม่ลลล์ได้ใจ) ส่วนประเภท Nonpreemptive ก็จะไม่มีการลัดคิวเลย ครูต้องไปต่อແຄวนน์เอง

Switching context

การที่ CPU เปลี่ยนไปเป็นอีกงานหนึ่ง คือ การสลับ เรียกว่า switching context ซึ่งทั้ง preemptive หรือ nonpreemptive ก็เกิดการสลับได้ทั้งนั้น (ไม่ลับก็ทำได้งานเดียวสิ) แต่ Preemptive scheduling (โドนแทรกแข่งได้) อาจจะทำให้เกิดการสลับไปสลับมาของ process ได้ เข้าๆ ออกๆ จำนวนมากๆ ค่า switch นี้ยิ่งมีมากก็จะแสดงว่า CPU ต้องสลับงานบ่อยนั่นเอง

Scheduling Criteria

OS จะต้องจัดการงานต่างๆ ให้กับ CPU ดังนี้นึงต้องมีการประเมินความสามารถของระบบว่ามีอะไรสามารถสนับสนุนได้ ที่ทำอยู่ว่าต้องไหน ต้องสุดหรือยัง โดยจะมีการคำนวณค่าดังนี้

1. CPU utilization คือ ความสามารถใช้งาน CPU ได้อย่างเต็มประสิทธิภาพ มีค่าเป็น % ถ้าทำงานเต็ม 100 คือ ทำงานเต็มประสิทธิภาพที่มันทำได้ แต่ถ้าทำ 100 ตลอด มันก็จะเหนื่อยทำให้อายุการใช้งานล้นลง ดังนั้น OS จึงให้ประเมินผลเต็มประสิทธิภาพที่สุดโดยที่ไม่กระทบกับ CPU จนเกินไปด้วย
2. Throughput คือ จำนวนงานที่ทำได้ในช่วงเวลาหนึ่ง ยิ่งมากก็ยิ่งดี เพราะจะหมายความว่าทำงานได้เยอะ
3. Turnaround time คือ เวลาทั้งหมดที่ process นั้นต้องใช้นับตั้งแต่เริ่มต้นทำงานจนเสร็จเลย นับเวลารอด้วย ดังนั้นบางงานได้ทำเร็วสุด แต่ถ้าอย่างต้องรอ จนทำเสร็จขั้นสุดท้ายก็ได้ เพราะโดยงานอื่นมาแทรก
4. Waiting time คือ เวลาที่ process ต้องรอ เช่น พอมาก็ไม่สามารถประเมินผลได้ เพราะต้องร่องงานที่มาก่อนทำเสร็จก่อน
5. Response time คือ เวลาที่ผู้ใช้ผู้ใช้สั่งเรียก หลังจาก process แรกเข้าไป

Scheduling Algorithms

มาถึง部分ของการเรื่องแล้วจะจากที่ OS เลือกว่าจะอนุญาตให้งานใดๆ ก็ได้มีสิ่ง (preemptive VS nonpreemptive) OS ก็จะต้องเลือกรอบวนการ (Algorithm) ด้วย ซึ่งจะมีกระบวนการหลายแบบ ให้เลือกสรรเลยละ แต่ละแบบก็มีข้อดี ข้อเสียต่างกัน หมายเหตุกับสถานการณ์ต่างๆ กันไป แต่บาง Algorithm ก็ต้องให้กับ nonpreemptive หรือ preemptive เท่านั้นนะ โดยทั่วไปจะมีที่นิยมยกตัวอย่างอยู่ 4 แบบ

First Come First Served Scheduling (FCFS)

ลำดับตัวแรกจ่ายมาก ต้องมาก่อนก็ได้ก่อน ขั้นนี้จะไม่มีการแทรกแข่งของงานชั่นเลย (nonpreemptive) จัดเป็นวิธีที่ง่ายที่สุด แล้วละ แค่มี Queue ลักษณะที่ทำได้แล้ว ซึ่งเป็นข้อดีของมัน แต่ว่า ถ้ามี process นึงมาถึงก่อน แต่ทำงานนานนาน ก็จะทำให้ process ชั่นๆ ที่มารอ ต้องค่อยนานมากๆ ด้วย ส่งผลให้ค่า Waiting time สูงไปด้วย แบบว่า ครุ๊กส์ลิฟท์ไปชั้น 40 นักเรียนจะขึ้นลิฟท์ไปชั้น 4 นี่เอง ต้องรอชั่นนานเลย ส่งผลให้ นักเรียนที่มาต่อແຄวต์ต้องรอเพิ่มขึ้นไปอีก

Shortest Job First Scheduling (SJF)

SJF เป็นวิธีที่น่าสนใจ หลักการคือ จะให้งานที่จะใช้เวลาห้องที่สุดทำงาน ถ้ามีงานมาถึงพร้อมกัน งานที่ใช้เวลาห้องกว่าก็จะได้ทำก่อน เนื่องจากคนจะยังคงเข้าลิฟท์ คนตัวเล็กก็จะได้เข้าไปก่อนนั้นเอง การทำงานแบบนี้ทำให้ waiting time มีค่าน้อยกว่าแบบ FCFS มาก ค่าของ turn around time ก็น้อยกว่าอีกด้วย เพราะไม่ต้องรอนานนัก แต่ข้อเสียของมันคือ CPU ไม่มีทางรู้อนาคตว่า งานที่มาจะใช้เวลาทำเท่าไหร่ ทำให้ SJF ใช้งานจริงไม่ได้ เป็นเนื้อหาแนวคิดในอุดมคติ หลักการทำงาน SJF สามารถใช้ได้ทั้งแบบ preemptive และ nonpreemptive นะ

Priority Scheduling

วิธีการนี้จะนำความสำคัญมาจัดการ หากมี process มากพร้อมกัน OS ก็จะให้ process ที่สำคัญกว่าเข้าทำงานก่อน ตัวเลข ความสำคัญ (Priority) ที่นิยมใช้คือ 1-5 โดยยิ่งน้อยยิ่งสำคัญ ดังนั้น process ที่มี priority เป็น 1 จะสำคัญที่สุด เรียกว่า ยิ่งใหญ่สุดเลยละ ถ้ามาทุกคนต้องหลบ หลักการนี้สามารถใช้ได้ทั้งแบบ preemptive และ nonpreemptive วิธีการนี้สืบต่อ ดี พอกล่าวเพราจะได้ค่าเฉลี่ยที่น่าพึงพอใจ งานที่ไม่สำคัญก็ทำทีหลังไปเลย แต่กว่าอาจเกิดปัญหาที่เรียกว่า ถูกแซงซึ่งได้ (Starvation) ปัญหานี้ก็คือ process ที่ไม่สำคัญรอคิวนานมากจนไม่มีวันได้ทำเลยละ เพราะถูกงานสำคัญมาเบี่ยงตลอด อาจารย์เล่าไว้ว่า มีคอมพิวเตอร์สมัยก่อนที่ทำงาน慢ๆ อย่างสิบปี แต่โปรแกรมเมอร์พึงพอใจว่ามีงานที่รอคิวอยู่นานเป็นสิบๆ ปีเลย (น่าสงสารมากอะ) วิธีการแก้ก็คือ ถ้า process รอนาน ก็เพิ่ม priority ให้มันเรื่อยๆ (เนื่องจากมันมีค่า priority ในมันเรื่อยๆ) แล้วเดี๋ยวมันจะได้ทำงานเอง

Round Robin Scheduling (RR)

round robin ตอนแรกผิดนึกว่าคือชื่อคนคิดคันจะอึก แต่ความจริงมันแปลว่า รอบวง ใช้แล้วรับ วิธีการนี้ คือกำหนดให้ process ทำงานเป็นรอบๆ โดยจะกำหนดเวลาที่เท่าเทียมกันมาเรียกว่า Quantum time โดย process ที่จะเข้าไปทำงาน จะทำงานแค่เวลาที่กำหนด ใครทำไม่เสร็จก็ต้องมาต่อແຄวต์ต่อ ส่วน process ในนั้นให้เวลาห้องกว่า Quantum time ก็จะลบหายตัวไปเมื่อต้องไปต่อແຄวต์ ซึ่งหมายความว่า CPU มีงานเข้ามาจำนวนมาก ลิ่งสำคัญของ round robin คือ Quantum time ซึ่งหากกำหนดมากเกินไปก็จะไม่ต่างกับ FCFS โดยทั่วไปจะมีค่าประมาณ 10 – 100 ms

Address Translation

1. การย้ายตำแหน่ง (Relocation)

ระบบปฏิบัติการในปัจจุบัน ยอมให้โปรแกรมทำงานพร้อมกันได้หลายงานแบบ multiprogramming ซึ่งโปรแกรมต่าง ๆ เข้าใช้งานหน่วยความจำร่วมกัน จึงต้องมีการสลับโปรแกรมให้เข้าออกหน่วยความจำได้ รวมถึงการเปลี่ยนแปลงค่าตำแหน่งในหน่วยความจำที่อ้างถึงในโปรแกรม ให้ถูกต้องตามตำแหน่งจริงในหน่วยความจำ เช่น โปรแกรม a อ้างถึงตำแหน่งที่ 1000 และโปรแกรม b ก็อ้างถึงตำแหน่งที่ 1000 เช่นกัน

ค่า address แบ่งได้ 2 ค่า

- **Absolute address** หมายถึง ตำแหน่งจริงของโปรแกรมที่อยู่ในหน่วยความจำ
- **Relative address** หมายถึง ตำแหน่งของคำสั่ง หรือ โปรแกรมของโปรแกรมที่อ้างถึงจากการ compile

2. การป้องกันพื้นที่ (Protection)

ระบบปฏิบัติการควรสามารถป้องกันโปรแกรม จากการถูกรบกวน ทั้งทางตรง และทางอ้อม ดังนั้นก่อนให้โปรแกรมได้เข้าครอบครองหน่วยความจำ จะต้องมีการตรวจสอบก่อน และใช้วิธีค้นหาเพื่อตรวจสอบตลอดเวลา

3. การใช้พื้นที่ร่วมกัน (Sharing)

การป้องกันเพียงอย่างเดียว อาจทำให้การใช้ทรัพยากรไม่คุ้ม จึงต้องมีการจัดสรรให้ใช้พื้นที่ของหน่วยความจำร่วมกันอย่างยั่งยืน

4. การจัดการแบ่งโปรแกรมย่อย (Logical organization)

ระบบปฏิบัติการจะแบ่งโปรแกรมเป็นโปรแกรมหลัก และโปรแกรมย่อย โดยนำเฉพาะโปรแกรมหลักลงในหน่วยความจำแต่สำหรับโปรแกรมย่อยลงหน่วยความจำเฉพาะเมื่อมีการเรียกใช้เท่านั้น

5. การจัดการแบ่งทางกายภาพ (Physical organization)

หน่วยความจำแบ่งเป็น 2 ส่วนคือ หน่วยความจำลักษณะ และหน่วยความจำสำรอง ลักษณะของหน่วยความจำลักษณะมีราคาแพง ทำงานได้เร็ว แต่เลื่อนหายใจได้ ในการทำงานจริง จึงต้องมีการเคลื่อนย้ายทางกายภาพระหว่างหน่วยความจำทั้ง 2 ตลอดเวลา ซึ่งเป็นหน้าที่ของระบบที่ต้องจัดสรรให้ให้สอดคล้องกับการทำงานแบบ multiprogramming

หน่วยความจำหลัก (Main memory)

การจัดการหน่วยความจำหลักระบบเริ่มจากแบบไม่ซับช้อน ไปถึงซับช้อน ในบทนี้จะเรียนรู้แบบไม่ซับช้อน ซึ่งไม่ถูกนำมาใช้งานในระบบปฏิบัติการปัจจุบัน แต่อาจใช้ในคอมพิวเตอร์ขนาดเล็กอยู่ การเรียนรู้เรื่องนี้อาจ นำไปประยุกต์ในการพัฒนางานด้าน software ชีน ๆ ได้

1. ระบบโปรแกรมเดียว (Monoprogramming) เป็นวิธีการจัดการที่ง่ายที่สุด โดยกำหนดเพียง 1 โปรแกรม ให้ทำงานในหน่วยความจำเพียงโปรแกรมเดียว

2. ระบบหลายโปรแกรมที่กำหนดขนาดพาร์ติชันคงที่ (Multiprogramming with fixed partition) ปัจจุบันระบบปฏิบัติการยอมให้มีหลายโปรแกรมทำงานพร้อมกันได้ หมายความว่า ขณะที่โปรแกรมหนึ่งกำลังรัน หักโปรแกรมที่รอ ก็เข้าใช้หน่วยความจำทันที โดยแบ่งหน่วยความจำออกเป็น partition การแบ่งเห็นแต่ละ partition แบบตายตัว มีจุดบกพร่อง ซึ่งมีการจัดการ

หน่วยความจำแบบ FCFS : first come first serve การจัดการแบบนี้ย่อมมีปัญหา จึงเป็นหน้าที่ของระบบปฏิบัติการ ที่ต้องจัดการ

3. ระบบที่กำหนดขนาดของพาร์ติชันให้เปลี่ยนแปลงได้ (Dynamic partition) เพื่อแก้ปัญหาของการกำหนดแบบคงที่ จึงออกแบบการกำหนด partition แบบเปลี่ยนแปลงได้ ซึ่งมีความซับซ้อนมากขึ้น ตามโปรแกรมที่เข้าใช้หน่วยความจำ

4. การจัดการแบบระบบบัดดี้ (Buddy system) ระบบจัดการหน่วยความจำแบบคงที่ และเปลี่ยนแปลงได้ อาจมีข้อจำกัดเรื่องการรองรับจำนวนโปรแกรม และใช้พื้นที่ไม่เต็มประสิทธิภาพได้ ส่วนการแบ่งแบบเปลี่ยนแปลงได้ก็มีความซับซ้อนในทางปฏิบัติ และใช้ทรัพยากร ซึ่งมีการยั่งยืนระหว่าง 2 ระบบนี้ เข้าด้วยกัน ระบบนี้จะแบ่งพื้นที่เป็น 2 ส่วน และตรวจสอบว่าพื้นที่แบ่งให้กับโปรแกรมล่าสุดได้หรือไม่ ถ้าแบ่งแล้วเข้าไม่ได้ ก็จะแบ่งขนาดเท่ากันหน้าที่ถูกแบ่ง และจัดเรียงพื้นที่เป็นแบบ link list ไว้รองการเข้าใช้งานโปรแกรม

การจองพื้นที่ในหน่วยความจำหลัก มี 4 รูปแบบ

1. **Single-user contiguous memory allocation** ยุคแรกมีผู้ใช้คนเดียว โหลดไปไว้ในหน่วยความจำหลักหมด

2. **Fixed Partitions** ยุคแม่ลติ โปรแกรมมิ่ง จะแบ่งล่วงที่ต้องอยู่กับล่วงที่เคลื่อนย้ายได้ โดยจัดการผ่าน Partition Manager มีปัญหา อาทิ เกิดพื้นที่ว่างใน Internal fragmentation มีขนาดใหญ่ เป็นต้น

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

4. **Relocationable Dynamic Partitions** จะมี Memory manager อยู่จัดการขับตำแหน่งการอ้างอิง ทำให้เหลือ block ในญี่ปุ่น เรียกว่า Compaction หรือ Garbage collection หรือ Defragmentation

หลักการของ Overlay คือ การแบ่งล่วงงานออกเป็นหลายล่วงตามการทำงาน นำเข้าหน่วยความจำหลักในเวลาที่ต้องใช้ และทับหน่วยความจำหลักเดิมที่ไม่ใช่ ซึ่งเป็นหลักการที่ใช้ใน Single-user contiguous memory allocation

Bound register คือ การเก็บตำแหน่งหน่วยความจำที่เก็บตำแหน่งสูงสุด หรือต่ำสุดที่มีการใช้งานหน่วยความจำหลัก และอ้างอิงไปใช้ใน memory manager ไม่ให้เกินขอบเขตที่กำหนดไว้

Relocation register คือ การเก็บตำแหน่งงานว่างงานถูกแบ่งเปลี่ยนไปจากตำแหน่งเดิมเท่าใด หากต้องการรู้ตำแหน่งปัจจุบัน ต้องใช้ตำแหน่งเดิม และตำแหน่ง Relocation register เพื่อใช้อ้างอิงในการนำมาใช้ต่อไป

Internal Fragmentation คือ งานที่มีขนาดใหญ่ เมื่อใช้ไปปลักระยะ จะเกินพื้นที่เล็กลง พื้นที่ ๆ เหลือ เรียกว่า การแตกกระจายภายใน (Internal Fragmentation)

External Fragmentation คือ งานที่ครอบครองหน่วยความจำ เมื่อใช้งานเสร็จ และเลิกใช้ พื้นที่นั้นจะว่าง เรียกว่า การแตกกระจายภายนอก (External Fragmentation)

การแบ่งเป็นหน้า (Paging)

ระบบที่ใช้น่วยความจำเสมือน (Virtual memory) มักใช้เทคนิคที่เรียกว่า การแบ่งหน้า หรือเพจจิ้ง(Paging) เพื่อเพิ่มระดับว่างๆ แบ่งหน้าทางตรรกะ กับเลขของเฟรมในหน่วยความจำหลัก

1. ตารางหน้า (Page table) หมายถึงการรับค่าตำแหน่งทางตรรกะเป็นค่าหน้า เลี้ยวหาเลขของเฟรมในหน่วยความจำหลัก ออกแบบ เทคนิคตารางหน้า(Page table) มักมีขนาดใหญ่ และต้องทำด้วยความเร็วสูง หากสรุปแล้วตารางหน้า ก็คือตารางที่เก็บอาร์เรย์ของรีจิสเตอร์นั่นเอง

2. บัฟเฟอร์คันนาที่อยู่ (TLB : Translation lookaside buffer) หากครั้งที่เรียกใช้น่วยความจำเสมือน ย่อมเรียกใช้หน่วยความจำหน้า 2 ครั้ง คือ การอ่านตารางหน้า และอ่านข้อมูลจริงจากหน่วยความจำหลัก ดังนั้นการทำงานแบบนี้จะใช้เวลาถึง 2 เท่า จึงมีการใช้ cache memory ซึ่งทำงานน้ำที่เก็บตารางหน้าที่เคยถูกเรียกใช้ หรือเรียกว่า บัฟเฟอร์คันนาที่อยู่ (TLB : Translation Lookaside Buffer) ถ้าพบใน TLB จะเรียก TLB hit ถ้าไม่พบ เรียก TLB miss และเข้าไปอ่านตารางหน้า เมื่ออ่านแล้วก็จะนำมายัง TLB สำหรับโอกาสที่จะถูกเรียกในอนาคต

การแบ่งเป็นชิ้น (Segmentation)

การแบ่งหน้าจะแบ่งให้มีขนาดเท่ากัน แต่การแบ่งเป็นชิ้นจะแบ่งโปรแกรมออกเป็นล่วง ๆ ไม่เท่ากัน และมีการใช้ตำแหน่งทางตรรกะ อ้างอิงตำแหน่งจริงเท่ากัน และมีการใช้ข้อมูล 2 ส่วนคือ เลขที่ชิ้น และระยะเริ่มต้นของชิ้น (Offset) สำหรับผลของการแบ่งชิ้นทำให้เกิดชิ้นล่วงไม่เท่ากัน (Dynamic partitioning) ซึ่งลดปัญหาการลัญเลี้ยฟื้นที่ (Internal fragmentation)

- การนำร่องการแบ่งเป็นชิ้นมาใช้ในหน่วยความจำเสมือน
- การรวมร่องการแบ่งเป็นหน้ากับการแบ่งเป็นชิ้นเข้าด้วยกัน

รูปแบบการจัดหน่วยความจำเสมือน

1. Paged Memory Allocation (โนลด์เด้นหน่วยความจำหลัก)

2. Demand Paging (โนลด์เฉพาะที่ต้องใช้เข้าหน่วยความจำหลัก)

3. Segmented Memory Allocation (แต่ละชิ้นไม่จำเป็นต้องเท่ากัน)

4. Segmented/Demand Paged memory Allocation (แบ่งเป็นชิ้นแล้วค่อยนำไปแบ่งเป็นเพจ)

Page frame คือ การแบ่งงานที่รองรับขนาดข้อมูลที่เท่า ๆ กันในแต่ละกรอบ โดย Page ที่ถูกวางแผนไว้จะเป็นต้องเรียกซ้ำติดกัน 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 เป็นต่อไปในแต่ละชิ้น เมื่องานทำเสร็จแล้วก็จะมี External Fragmentation เกิดขึ้น ทำให้เกิด Overhead ในการเลือกพื้นที่สำหรับของ Segment สำหรับงานใหม่

Virtual Memory

Virtual Memory คือ หน่วยความจำที่ถูกสร้างขึ้นมาในกรณีที่ RAM ไม่พอใช้ ซึ่งโดยส่วนมากมักเอาพื้นที่ในฮาร์ดดิสก์บางส่วนมาใช้แทน เนื่องจากหน่วยความจำของระบบมีจำกัดและมีราคาสูง เราก็จึงเรียกว่าความจำเสมือน การใช้น่วยความจำเสมือนจะทำให้สามารถทำงานกับโปรแกรมขนาดใหญ่มาก ๆ ได้ โดยไม่เสียพื้นที่ เรื่องหน่วยความจำไม่เพียงพอ ระบบการทำงานของหน่วยความจำเสมือนจะใช้วิธีแบ่งโปรแกรมออกเป็นล่วง ๆ และคอมพิวเตอร์จะทำการลับ (Swap) ส่วนโปรแกรมที่ซึ่งไม่ได้ใช้งานไปยังหน่วยเก็บข้อมูลสำรอง และทำการลับกลับมาในหน่วยความจำหลักเมื่อจำเป็นต้องใช้งาน ส่วนวิธีการใช้ประโยชน์จาก Virtual Memory ให้ได้ผลดีที่สุดจะต้องหาวิธีที่จะทำให้พื้นที่ที่กำหนดไว้ในเป็น Swap File ซึ่งเป็นพื้นที่ที่อยู่บนฮาร์ดดิสก์ ทำให้การอ่านเขียนข้อมูลได้เร็วขึ้น

Motivations for Virtual Memory

1. ให้ DRAM ทางกายภาพเป็นแค่สำหรับดิสก์

- พื้นที่ที่อยู่ของกระบวนการสามารถเก็บขนาดหน่วยความจำหลักทางกายภาพได้
- ผู้รวมของพื้นที่ที่อยู่ของกระบวนการหลายเกินทางกายภาพหน่วยความจำ

2. จัดการหน่วยความจำลดความซับซ้อน

- มีวิธีที่อยู่หลายกระบวนการในหน่วยความจำ แต่ละกระบวนการที่มีอยู่ของตัวเองเท่านั้น "งาน" รหัสและข้อมูลที่เป็นจริงในความทรงจำจัดสรรหน่วยความจำมากขึ้นในการดำเนินการตามที่จำเป็น

3. Provide Protection

- หนึ่งกระบวนการที่ไม่สามารถถูกเกี่ยวกับคนอื่น เพราะพวจษาทำงานอยู่ในพื้นที่ที่แตกต่างกัน
- กระบวนการผู้ใช้ไม่สามารถเข้าถึงข้อมูลได้รับการยกเว้นส่วนต่าง ๆ ของช่องว่างอยู่มีสิทธิ์ที่แตกต่างกัน

Possibility of Thrashing

1. เพื่อรับกระบวนการมากที่สุดเท่าที่เป็นไปได้เพียงไม่กี่หน้าของแต่ละชิ้นตอนจะถูกเก็บไว้ในหน่วยความจำ

2. แต่หน่วยความจำอาจจะเต็มเมื่อระบบปฏิบัติการนำหน้าหนึ่งในมันต้องลับชิ้นหนึ่งของมา

3. ระบบปฏิบัติการไม่ต้องลับออกหน้าของกระบวนการก่อนที่หน้าเป็นสิ่งจำเป็น

4. ถ้ามันทำอย่างนี้บ่อยเกินไปนี่นำไปสู่ thrashing: ประมวลผลใช้เวลาล่วงไปสู่ลับหน้าเร็วของตนคงข้างกับการทำเนินการคำแนะนำของผู้ใช้

Process Execution(การดำเนินการกระบวนการ)

1. ระบบปฏิบัติการที่จะนำเข้ามาในหน่วยความจำเพียงไม่กี่หน้าของโปรแกรม (รวมถึงจุดเริ่มต้นของมัน)

2. รายการตารางแต่ละหน้าจะมีบิตปัจจุบัน (P bit) ที่ถูกตั้งค่าเฉพาะในการนี้ที่หน้าที่เกี่ยวข้องอยู่ในหน่วยความจำหลัก

Roles of the Operating System

Referee (กรรมการ)

- Resource allocation among user application (การจัดสรรทรัพยากระหว่างแอปพลิเคชันของผู้ใช้)
- Isolation of different user,application from each other (การแยกผู้ใช้ที่แตกต่างกัน แอปพลิเคชันออกจากกัน)
- Communication between user,application (การติดต่อสื่อสารระหว่างผู้ใช้และแอปพลิเคชัน)

Illusionist (ภาพลวงตา)

- Each application appears to have entire machine to itself (แต่ละแอปพลิเคชันดูเหมือนจะมีทั้งเครื่องในด้านเอง)
- Infinite processors, memory, reliable storage, network (โปรเซสเซอร์ที่ไม่มีที่สิ้นสุด, หน่วยความจำ, ที่เก็บข้อมูลที่เชื่อถือได้, เครือข่าย)

Glue (ประสาน)

- provide libraries, user interfaces, widgets (จัดหาไลบรารี สำหรับติดต่อผู้ใช้/วิศวกรรม)

What is an Operating System (ระบบปฏิบัติการคืออะไร)

- A set of software that manage computer's resource for it's users and their application (ชุดซอฟต์แวร์ที่จัดการทรัพยากรของคอมพิวเตอร์ สำหรับผู้ใช้และแอปพลิเคชัน)
- May visible or invisible to the user (ผู้ใช้อาจมองเห็นหรือมองไม่เห็น)
- 2 major kinds
 - General purpose os ทั่วไป
 - Specific purpose os

- User would stand in line to use the computer (ผู้ใช้จะยืนคิ้ว隊เพื่อใช้คอมพิวเตอร์)
- Batch System
 - Keep cpu by having a queue of jobs (เก็บ cpu โดยมีคิวงาน)
 - Os would load next job while current one runs (Os จะโหลดงานถัดไปในขณะที่งานปัจจุบันทำงานอยู่)
 - User would submit job ,and wait (User ได้คิ้วและรอ)

Time-Sharing Operating Systems: (ระบบปฏิบัติการแบ่งเวลา)

- Multiple users on computer at some time (บางครั้งอาจจะมีผู้ใช้หลายคนบนคอมพิวเตอร์)
- Interactive performance try to complete everyone's tasks quick (พยายามทำให้ทุก process เสร็จย่างเร็วที่สุด)
- As computer became cheaper more important to optimize for user time not computer time (เมื่อคอมพิวเตอร์มีราคาถูกลง สิ่งสำคัญคือการปรับให้เหมาะสมสำหรับเวลาของผู้ใช้ ไม่ใช่เวลาของคอมพิวเตอร์)



Operating System Evaluation (การประเมินระบบปฏิบัติการ)

- Reliability and Availability (ทำงานตามวัตถุประสงค์ และพร้อมใช้งานตลอดเวลา)
- Security (ความปลอดภัย)
- Privacy (แบ่งแยกฟลีดของแต่ละ ex.ไฟล์ของ application ได้โดยไม่แทรกผ่านๆกัน)
- Portability (สามารถใช้ได้หลาย Hardware)
- Performance (ประสิทธิภาพ)
 - Overhead (ใช้ Overhead น้อย)
 - Throughput (จำนวนงานที่ทำได้ Per time unit)
 - Latency (ความเร็วในการตอบสนอง Response)
 - Fairness (ความ fair ของการใช้งานแต่ละ Program)
 - Predictability (ความสามารถในการคาดการณ์ประสิทธิภาพ)

Design Tradeoffs

- Must balance between the 5
 - Preserve legacy = Portability ต้อง, Reliable ต้อง, Secure ต้อง
 - Break on Abstraction = Performance ต้อง, Portability ต้อง, Reliability ต้อง

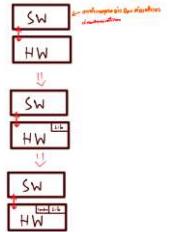
* First Computer 1940s

Early Operating Systems (ระบบปฏิบัติการแรก)

- Had complete Control of Hardware (มีการควบคุมฮาร์ดแวร์อย่างสมบูรณ์)
- Os was runtime library (Os เป็นไลบรารีรันไทม์)

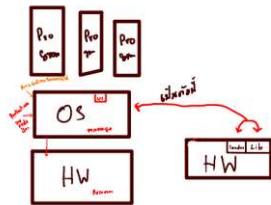
The Kernel Abstraction

- Single task system



การทำงานทุกอย่าง Dev ต้องทำเอง

- Multitasking system



Prog เข้าสู่ HW โดยตรงไม่ได้ Protection HW ถูกปกป้อง
การเปลี่ยนแปลงจากระบบแบบ Single Task ไปเป็นระบบแบบ Multi Task สิ่งที่จะต้องมีการปรับแต่ง ใน Os

Os ต้องควบคุมลำดับการทำงานของโปรแกรม

Os จะตรวจสอบการทำงานของโปรแกรม

What does an Os do (Os ทำอะไร)

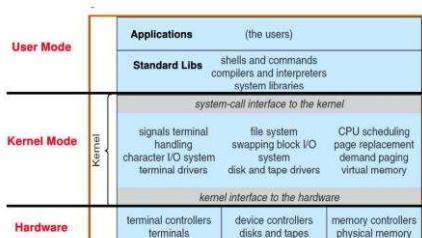
- Hiding Complexity (ซ่อนความซับซ้อน) process จำลอง Com เพื่อให้ Program เข้าสู่ Process ขั้นตอนที่ Os หน้าที่ Kernel

Kernel is the part of the os that running all the time on the computer (เครื่องเนลเป็นส่วนหนึ่งของระบบปฏิบัติการที่ทำงานตลอดเวลาบนคอมพิวเตอร์)

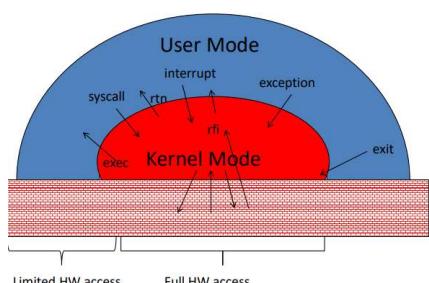
- ชั้นส่วนของ Kernel จะอยู่ในремตลดเวลา
 - Core part of the os (ส่วนหลักของระบบปฏิบัติการ)
 - Manage System Resources (จัดการทรัพยากร)
 - Act's as a bridge between app and HW (ทำหน้าที่เป็นสะพานเชื่อมระหว่างแอปและ HW)

Unix Structure

UNIX Structure



User/Kernel (Privileged) Mode



One of the major goals of Os is (ป้าหมายหลัก)

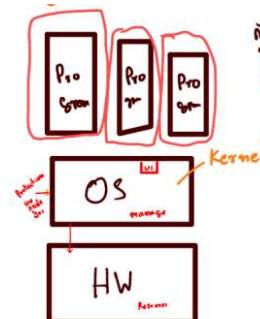
- Protecting Process and the Kernel (การปกป้องกระบวนการและการแล็คเอย์นเล)
- Running Multiple Program
 - Keep them from interfering with OS - kernel (ป้องกันไม่ให้รบกวน OS -kernel)
 - Keep them from interfering with each other (ป้องกันไม่ให้รบกวนซึ่งกันและกัน)

แต่ละตัวจะรัน Process ของตัวเอง

ถ้ามีการทำลาย Protection จะทำลาย Process ที่สักทำไม่ได้ ระบบจะ

หยุดทำงาน

ถ้ามีการล้ำเส้นของ Program จะเกิดการจอฟ้า



Protection Why?

- Protect Process

- Process ไม่ต้อง Process จะไม่ไปยุ่งกันและกัน จะถูก Protect ไว้
- Protect Kernel ป้องกันตัวมันเอง ไม่เข้ามาจึงมีการเพี้ยนเพื่อเปลี่ยนรูปแบบการทำงานของ Os
- สิ่งที่ต้องเพิ่มคือ Protect Hw เพราะตัว Program จะไม่สามารถเข้าสู่ HW โดยตรงได้ Os จะเป็นตัวจัดการให้ทั่วไป Non privilege instruction
- Memory Privilege instruction

การ Protect Process และ kernel ทำให้เกิด Impact อะไรกับระบบบ้าง และต้อง Protect อะไรอีกบ้าง เพื่ออะไร

- Reliability : Buggy Program only hurt Themselves (โปรแกรม Buggy ความไม่ถูกต้องก่อให้ร้ายตัวเองเท่านั้น)
- Security and privacy : Trust Programless (เชื่อถือแบบไม่มีโปรแกรม)
- Fairness : Enforce shares of disk CPU (บังคับใช้ร่วมกันของคิสก์ CPU)

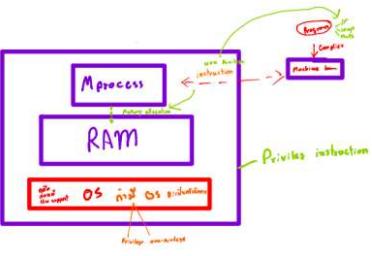
Protection : How (HW/SW)

- 2 main Hw Mechanism

- Memory address Translation
 - Dual mode operation
 - Privilege | Non privilege Mode
 - System calls

- SW

- Process
- System Calls



Dual-Mode Operation

- Kernel Mode
 - Execution with the full privileges of the hardware (ดำเนินการด้วยสิทธิ์เต็มรูปแบบของชาร์ดแวร์)
 - Read/write to any memory, access any I/O device, read/write any disk sector, send/read any packet (อ่าน/เขียนไปยังหน่วยความจำ เข้าถึงอุปกรณ์ I/O ได้จากอ่าน เขียน เช่น เครื่องคอมพิวเตอร์ของคิดส์ ส่ง/อ่านแพ็กเก็ตได้ๆ)
- User Mode
 - Limited privileges (สิทธิพิเศษมีจำนวนจำกัด)
 - Only those granted by the operating system kernel (เฉพาะที่ได้รับจากเครื่องเรนเดอร์ของระบบปฏิบัติการ)
- On the x86, mode stored in EFLAGS register
- On the MIPS, mode in the status register

Hardware Support : Dual-Mode Operation

- Privileged instruction (คำแนะนำพิเศษ)
 - Available to kernel (ใช้ได้กับเครื่องเรนเดอร์)
 - Not available to user code (ไม่สามารถใช้ได้กับรหัสผู้ใช้)
- Limits on memory accesses
 - To prevent user code from overwriting the kernel (เพื่อป้องกันไม่ให้รหัสผู้ใช้เขียนทับเครื่องเรนเดอร์)
- Timer
 - To regain control from a user program in a loop (เพื่อกลับมาควบคุมจากโปรแกรมผู้ใช้ในวงจร)
- Safe way to switch from user mode to kernel mode, and vice versa (วิธีที่ปลอดภัยในการเปลี่ยนจากโหมดผู้ใช้เป็นโหมดเครื่องเรนเดอร์และในทางกลับกัน)

User Mode

- Application Program
 - Running in process (Run program ที่อยู่ใน)

Virtual Machine: VM

- Software emulation of an abstract machine
 - Give programs illusion they own the machine (ให้โปรแกรมเป็นภาพลวงตาว่าพวกเขานั้นเป็นเจ้าของเครื่อง)
 - Make it look like HW has feature you want (ทำให้ดูเหมือนว่า HW มีคุณสมบัติที่คุณต้องการ)
- 2 Type Of VM
 - Process vm
 - Supports the execution of a single program (one of the basic function of the OS) (รองรับการทำงานของโปรแกรมเดียว (หนึ่งในฟังก์ชันพื้นฐานของ OS))
 -

System VM

- Supports the execution of an entire OS and its applications (รองรับการทำงานของทั้งระบบปฏิบัติการและแอพพลิเคชัน)

Process VMs

- Goal :
 - Provide an isolation to a program (จัดให้มีการแยกโปรแกรม)
 - Process unable to directly impact other Process (กระบวนการไม่สามารถส่งผลกระทบต่อกระบวนการอื่นโดยตรง)
 - Boundary to the usage of a memory (ขอบเขตของการใช้หน่วยความจำ)
 - Fault isolation (การแยกข้อผิดพลาด)
 - Bugs in program can't crash the computer (ข้อบกพร่องในโปรแกรมไม่สามารถทำให้คอมพิวเตอร์เสียหายได้)
- Portability (Program) : การพกพา (โปรแกรม)
 - Write the program for the os other the Hardware (เขียนโปรแกรมสำหรับระบบปฏิบัติการอื่น ๆ ของชาร์ดแวร์)

Process Abstraction

- Process : an instance of a program , running with limited right (ตัวอย่างของโปรแกรม ทำงานโดยจำกัดสิทธิ์)
 - Thread: a sequence of instructions within a process (V.microprocessor)
 - Potentially many threads per process (for now 1:1)
 - Address space: set of rights of a process
 - Memory that the process can access
 - Other permissions the process has (e.g., which system calls it can make, what files it can access)

Process

- 2 part
 - PCB in kernel
 - Others in user

Process Control Block: PCB

- Kernel represents each process as a process control block (PCB)
 - – Status (running, ready, blocked, ...)
 - – Registers, SP, ... (when not running)
 - – Process ID (PID), User, Executable, Priority, ...
 - – Execution time, ...
 - – Memory space, translation tables, ...
- Kernel Scheduler maintains a data structure containing the PCBs (Kernel Scheduler รักษาโครงสร้างข้อมูลที่ประกอบด้วย PCBs)
- Scheduling algorithm selects the next one to run (อัลกอริทึมการตั้งเวลาเลือกรายการตัดไปเพื่อเรียกใช้)

Main Points

Process concept

- A process is the Os abstraction for executing a (กระบวนการการที่ต้องนามธรรมของระบบปฏิบัติการสำหรับการดำเนินการ)

Dual-Mode operation : user vs kernel

- Kernel-mode : execute with complete privileges
- User-mode : execute with fewer privileges

Safe Control transfer

- How do we switch from one mode to the other

Mode Switch

from user mode to kernel mode

- Interruptions
 - Triggered by time and I/O device
- Exception
 - Triggered by unexpected program behavior (ที่ไม่คาดไวโดยพฤติกรรมของโปรแกรมที่ไม่คาดคิด)
 - Or malicious behavior (หรือพฤติกรรมที่เป็นอันตราย)
- System calls (aka protected procedure call)
 - Request by program for kernel to do some operation on its behalf (คำขอโดยโปรแกรมสำหรับเครื่องเนื่องเพื่อดำเนินการบางอย่างในนามของมัน) เช่น การหารด้วย OS
 - Only limited # of very carefully coded entry point (จุดเข้าที่ของจุดเข้าห้ามอย่างระมัดระวังเท่านั้น)

From kernel mode to user mode

- New process/new thread start (kernel สร้างตัว process)
 - Jump to first instruction in program/thread (ขึ้นไปที่คำสั่งแรกในโปรแกรม/ธread)
- Return from interrupt, exception , system call (กลับจากการขัดจังหวะที่ยกเว้น การเรียกของระบบ) จบ syscall
 - Resume suspended execution (ดำเนินต่อการดำเนินการต่อไป)
- Process / thread context switch
 - Resume some other process
- User-level upcall (Unix signal) ด้านหลังของ syscall
 - Asynchronous notification to user program (การแจ้งเตือนแบบอะซิงไกรันส์ไปยังโปรแกรมผู้ใช้)

Implementing Safe Kernel Mode Transfers (การที่ Safe Kernel Mode Transfers) จะฟื้นค่า state ไว้

- Carefully constructed kernel code pack up the user process state and sets it aside (เก็บค่าสถานะที่สร้างขึ้นอย่างระมัดระวังไว้บรรจุสถานะกระบวนการของผู้ใช้และแยกไว้)
- Must handle weird/buggy/malicious user state (ต้องจัดการกับสถานะผู้ใช้ที่แปลก/บีกี้/ประسنค์ร้าย)
 - Syscall with null pointers
 - Return instruction out of bound (ส่งคืนคำสั่งนอกขอบเขต)
 - User stack pointer out of bound (ตัวชี้สับเปลี่ยนผู้ใช้ที่อยู่นอกขอบเขต)
- Should be impossible for buggy or malicious user program to cause the kernel to corrupt it self (ไม่ควรให้โปรแกรมผู้ใช้ที่เป็นบกก์หรือประسنค์ร้ายทำให้คอร์นอลเดียหายได้เอง)

How do we take interrupts safely

- Interrupt vector
- Kernel interrupt stack
- Interrupt masking
- Atomic transfer of control
 - Single instruction – like to change
 - Program counter
 - Stack pointer
 - Memory protection
 - Kernel/user mode
- Transparent restartable execution

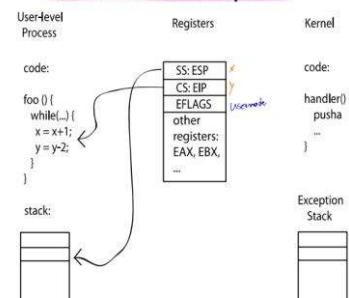
The kernel Stack

- Interrupt handlers want a stack ตัวแปรถูกสร้างใน Stack
- System call handlers want a stack จำเป็นต้องมี Stack
- Can't just use the user stack [why?] security ความปลอดภัย

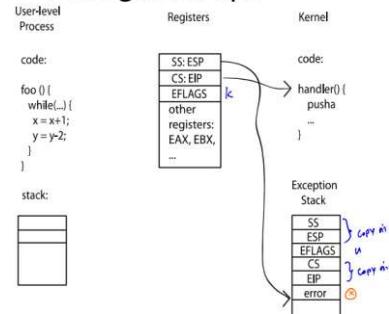
Case Study: x86 Interrupt

- Save current stack pointer
- Save current program counter
- Save current processor status word (condition codes)
- Switch to kernel stack; put SP, PC, PSW on stack
- Switch to kernel mode
- Vector through interrupt table
- Interrupt handler saves registers it might clobber

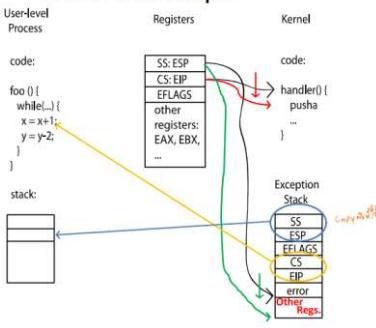
Before Interrupt



During Interrupt



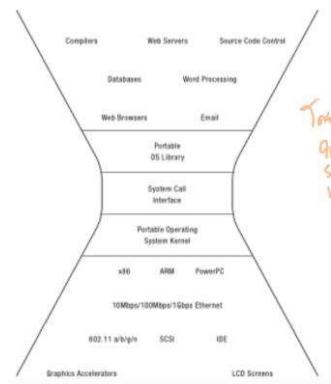
After Interrupt



Today : Four Fundamental Os concepts

- Thread : Execution Context
 - Program Counter , Registers , Execution Flags , Stack
- Address Space (with translation)
 - Program's view of memory is distinct from physical machine(มุมมองของโปรแกรมเกี่ยวกับหน่วยความจำแตกต่างจากเครื่องที่มีอยู่จริง)
- Process : an instance of a running program
 - Address space + One or more Threads
- Dual mode operation / protection
 - Only the "system" can access certain resources (เฉพาะ "ระบบ" เท่านั้นที่สามารถเข้าถึงทรัพยากรบกางอย่างได้)
 - Combined with translation, isolates programs from each other (รวมกับการแปลง, แยกโปรแกรมจากกันและกัน)

The Process and Programming Interface



Shell

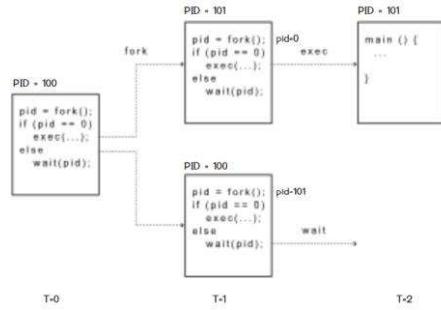
- A shell is a job control system
 - Allows programmer to create and manage a set of programs to do some task (อนุญาตให้โปรแกรมเมอร์สร้างและจัดการชุดของโปรแกรมเพื่อทำงานบางอย่าง)
 - Windows, MacOS, Linux all have shells
- Example : to compile a C program
 - CC -C sourcefile1.c
 - CC -C sourcefile2.c
 - In -o program sourcefile1.o sourcefile2.o

Unix process Management

- Unix fork – system call to create a copy of the current process, and start it running (Copy ตัว Process แล้วเปลี่ยนค่า) No argument

- Unix Exec – system call to change the program being run by the current process (การเรียกกระบวนการเพื่อเปลี่ยนโปรแกรมที่กำลังรันโดยกระบวนการการปั๊กบัน)
- Unix wait – system call to wait for a process to finish (การเรียกกระบวนการเพื่อรอกระบวนการเสร็จ)
- UNIX signal – system call to send a notification to another process(สัญญาณ UNIX – การเรียกของระบบเพื่อส่งการแจ้งเตือนถึงกระบวนการอื่น)

UNIX Process Management



Concurrency

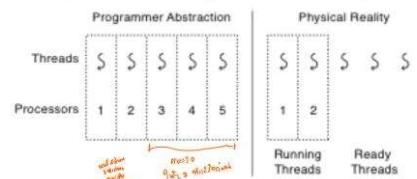
Thread จะใช้งานขึ้นตอนๆ กับ Scheduler

Threads in the Kernel and at User-Level

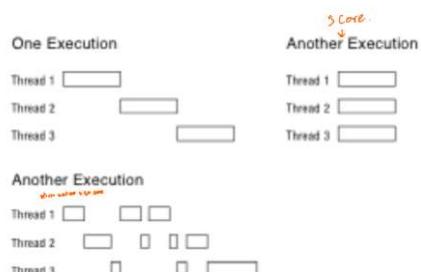
- Multi-threaded kernel ประกอบไปด้วย 1 process
- Multiprocess kernel : process หลายตัวอัน อย่างน้อยทำjob 1 process ต่อ 1 thread
- Multiple multi-threaded user processes : 1 user process ใช้หลาย process

Thread Abstraction

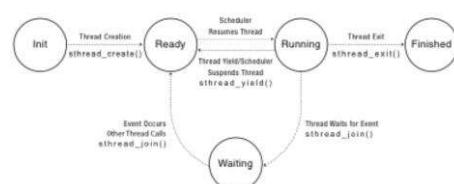
- Infinite number of processors
- Threads execute with variable speed
 - Programs must be designed to work with any schedule



Possible Executions



Thread Lifecycle



Synchronization (ເຄີດມານີ້ເອົ້ນ Race Condition)

Definitions

- Race condition : output of a concurrent program depends on the order of operations between threads (ຜົດລັບຮູ້ອງໄປແກຣມພວ່ນກັນຈິ້ນອຸ້ງກັນຄຳດັບຂອງການດຳເນີນກະຮວ່າງເຫຼືອ)
- Mutual exclusion : only one thread does a particular thing at a time (ເຫຼືອດີຍ່າວ່ານີ້ທີ່ກໍ່າວ່າໃຫ້ມີຕ່ອງດາມທີ່ດີ່ອການກັນໄວ້)
- Lock : prevent someone from doing something (ປຶກກັນໄວ້ໃຫ້ໄກຮ່າວ່າໄວ້)

Scheduling



Definitions

- Task/Job
 - User request: e.g., mouse click, web request, shell command, ...
- Latency/response time
 - How long does a task take to complete?
- Throughput
 - How many tasks can be done per unit of time?
- Overhead
 - How much extra work is done by the scheduler?
- Fairness
 - How equal is the performance received by different users?
- Predictability
 - How consistent is the performance over time?

More Definitions

- Workload
 - Set of tasks for system to perform
- Preemptive scheduler
 - If we can take resources away from a running task
- Work-conserving
 - Resource is used whenever there is a task to run
- Scheduling algorithm
 - takes a workload as input
 - decides which tasks to do first
 - Performance metric (throughput, latency) as output
 - Only preemptive, work-conserving schedulers to be considered

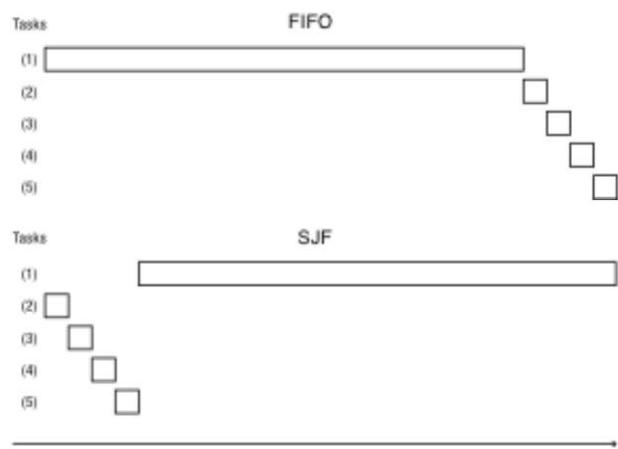
First In First Out (FIFO)

- Schedule tasks in the order they arrive
 - Continue running them until they complete or give up the processor
- Example: memcached
 - Facebook cache of friend lists, ...
- On what workloads is FIFO particularly bad?

Shortest Job First (SJF)

- Always do the task that has the shortest remaining amount of work to do
 - Often called Shortest Remaining Time First (SRTF)
- Suppose we have five tasks arrive one right after each other, but the first one is much longer than the others
 - Which completes first in FIFO? Next?
 - Which completes first in SJF? Next?

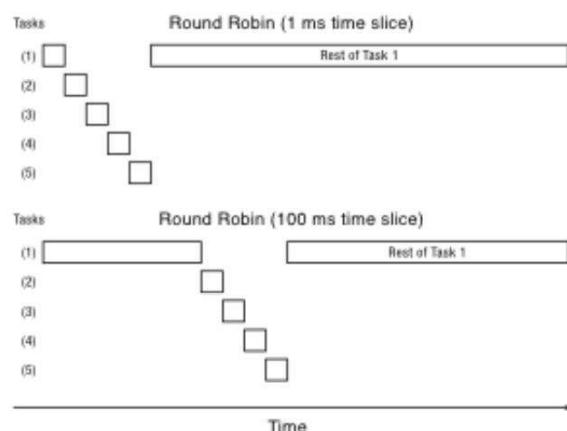
FIFO vs. SJF



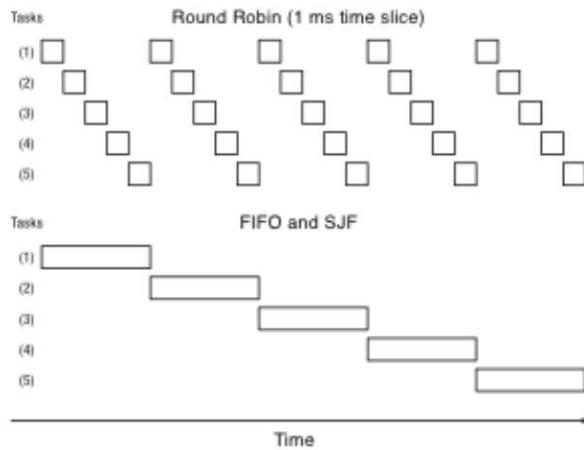
Round Robin

- Each task gets resource for a fixed period of time (time quantum)
 - If task doesn't complete, it goes back in line
- Need to pick a time quantum
 - What if time quantum is too long?
 - Infinite?
 - What if time quantum is too short?
 - One instruction?

Round Robin



Round Robin vs. FIFO



Round Robin = Fairness?

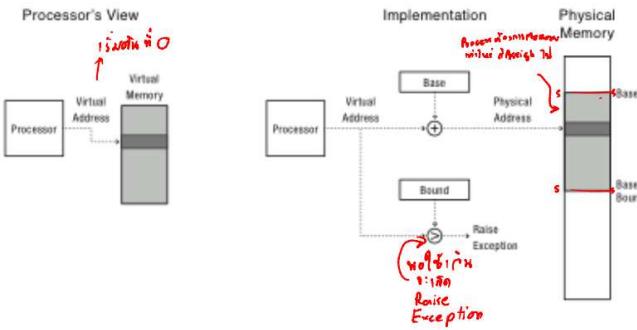
- Is Round Robin always fair?
- What is fair?
 - FIFO?
 - Equal share of the CPU?
 - What if some tasks don't need their full share?
 - Minimize worst case divergence?
 - Time task would take if no one else was running
 - Time task takes under scheduling algorithm

Address Translation Goal

Address Translation Goals

- Memory protection
- Memory sharing
 - Shared libraries, interprocess communication
- Sparse addresses
 - Multiple regions of dynamic allocation (heaps/stacks)
- Efficiency
 - Memory placement
 - Runtime lookup
 - Compact translation tables

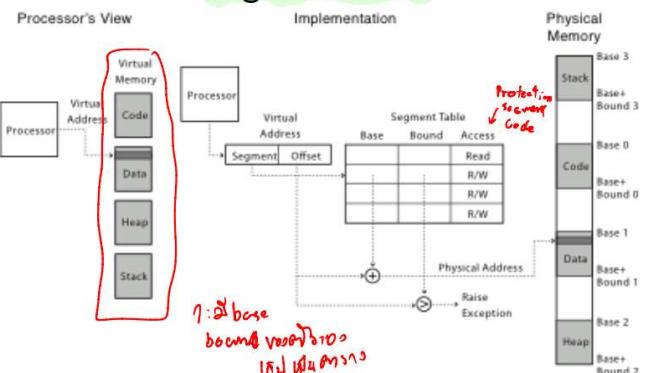
Virtually Addressed Base and Bounds



Segmentation

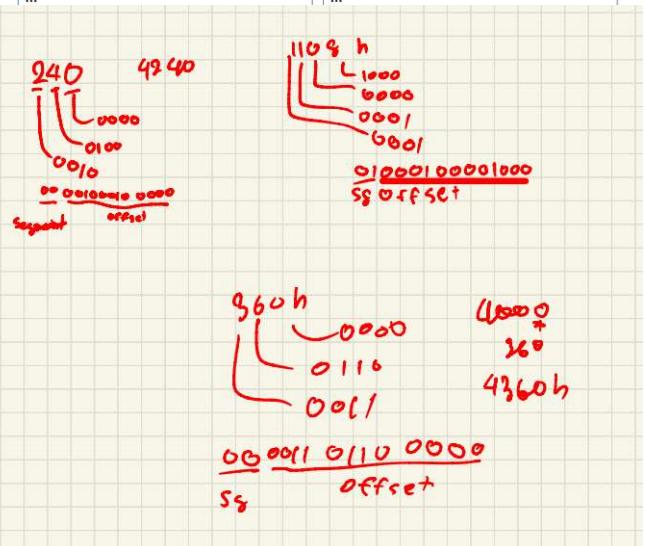
- Segment is a contiguous region of **virtual memory**
- Each process has a segment table (in hardware)
 - Entry in table = segment
- Segment can be located anywhere in physical memory
 - Each segment has: start, length, access permission
- Processes can share segments
 - Same start, length, same/different access permissions

Segmentation



	Segment start	length	Physical Memory
2 bit segment #	code	0x4000	0x700
12 bit offset	data	0	0x500
	heap	-	-
	stack	0x2000	0x1000

main: 240	store #1108, r2	x: 108	a b c \0
244	store pc+8, r31	...	
248	jump 360	main: 4240	store #1108, r2
24c		4244	store pc+8, r31
...		4248	jump 360
strlen: 360	loadbyte (r2), r3	424c	
...	
420	jump (r31)	strlen: 4360	loadbyte (r2), r3
...		...	
x: 1108	a b c \0	4420	jump (r31)
...		...	



Visual Memory

Memory Hierarchy

Level vs choice

Cache	Hit Cost	Size
1st level cache/first level TLB	1 ns	64 KB
2nd level cache/second level TLB	4 ns	256 KB
3rd level cache	12 ns	2 MB
Memory (DRAM)	100 ns	10 GB
Data center memory (DRAM)	100 μ s	100 TB
Local non-volatile memory	100 μ s	100 GB
Local disk	10 ms	1 TB
Data center disk	10 ms	100 PB
Remote data center disk	200 ms	1 XB

Virtual or Physical Dirty/Use Bits

- Most machines keep dirty/use bits in the page table entry
- Physical page is
 - Modified if *any* page table entry that points to it is modified
 - Recently used if *any* page table entry that points to it is recently used
- On MIPS, simpler to keep dirty/use bits in the core map
 - Core map: map of physical page frames

FIFO in Action

Reference	A	B	C	D	E	A	B	C	D	E
1	A				E			D		C
2		B			A			E		D
3		C			B			A		E
4		D			C			B		A

Worst case for FIFO is if program strides through memory that is larger than the cache

MIN, LRU, LFU → no eviction

- MIN
 - Replace the cache entry that will not be used for the longest time into the future
 - Optimality proof based on exchange: if evict an entry used sooner, that will trigger an earlier cache miss
- Least Recently Used (LRU)
 - Replace the cache entry that has not been used for the longest time in the past
 - Approximation of MIN
- Least Frequently Used (LFU)
 - Replace the cache entry used the least often (in the recent past)

LRU/MIN for Sequential Scan

Reference	LRU									
	A	B	C	D	E	A	B	C	D	E
1	A				E			D		C
2		B				A			E	D
3			C				B		A	E
4			D				C		B	A

Reference	LRU									
	A	B	A	C	B	D	A	D	E	D
1	A				+			+		+
2		B				+				+
3			C				E			+
4			D			+	+			C

Reference	FIFO									
	A	B	A	C	B	D	A	E	A	B
1	A				+			+	E	
2		B				+				A
3			C							B
4			D			+	+			C

Reference	MIN									
	A	B	A	C	B	D	A	E	A	B
1	A				+			+		+
2		B				+				+
3			C				E			+
4			D			+	+			C

Belady's Anomaly

FIFO (3 slots)										
Reference	A	B	C	D	A	B	E	A	B	C
1	A			D			E			+
2		B			A			+		C
3			C			B		+		D

FIFO (4 slots)										
Reference	A	B	C	D	A	B	E	A	B	C
1	A				+		E			D
2		B				+		A		E
3			C					B		
4			D						C	

Possibility of Thrashing

- เพื่อร้องรับกระบวนการภารกิจที่สุดเท่าที่เป็นไปได้เพียงไม่กี่หน้าของแต่ละขั้นตอนจะถูกเก็บไว้ในหน่วยความจำ
- แต่หน่วยความจำอาจจำเต็มเมื่อระบบปฏิบัติการนำหน้าหนึ่งในมันต้องลับซึ้งหนึ่งออกมา
- ระบบปฏิบัติการไม่ต้องลับออกหน้าของกระบวนการภารกิจที่หน้าเป็นสิ่งจำเป็น
- ถ้ามันทำอย่างนี้บ่อยเกินไปนั่นนำไปสู่ thrashing: ประมาณผลให้เวลาส่วนใหญ่ลับหน้าเว็บของตนค่อนข้างกว่าการดำเนินการคำแนะนำของผู้ใช้

Process Execution(การดำเนินการกระบวนการ)

- ระบบปฏิบัติการที่จะนำเข้ามาในหน่วยความจำเพียงไม่กี่หน้าของโปรแกรม (รวมถึงจุดเริ่มต้นของมัน)
- รายการตารางแต่ละหน้าจะมีบิตปัจจุบัน (P bit) ที่ถูกตั้งค่าเฉพาะในกรณีที่หน้าที่เกี่ยวข้องอยู่ในหน่วยความจำหลัก

Multitasking system

Program เท่านั้น Process เป็นคอมพิวเตอร์ เป็นสเปกเดียวกันหมด

Activity #1

การเปลี่ยนแปลงจากระบบแบบ single task ไปเป็นระบบแบบ Multitask...สิ่งที่จะต้องมีการปรับแต่งหรือเพิ่มเติมเข้ามาใน OS ได้แก่

- การจัดการระบบ หรือ จัดสันทรัพยากรให้เป็นไปตามขั้นตอน

What does an OS do...

Hiding Complexity

Variety of HW

- E.g. different CPU, amount of RAM, I/O devices

Kernel is the part of the OS that runs all the time on the computer

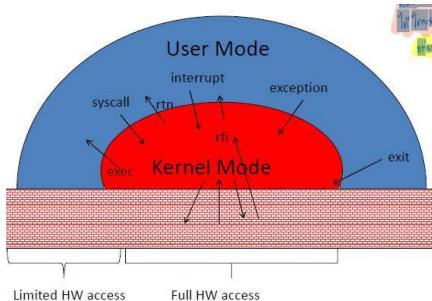
- Core part of the OS
- Manages system resources
- Acts as a bridge between apps and HW

* ส่วนที่เป็น kernel จะอยู่ใน Ram ตลอดเวลา อุ่นอย่างถาวร

* Kernel เป็นตัวกลางที่จะผ่านไป Program

User/Kernel (Privileged) Mode

ยอมให้โปรแกรมติดต่อโดยตรงโดยที่ไม่ผ่าน kernel แต่จะ Error ได้เจ้ายมาก



หนึ่งในเป้าหมายหลักของ OS คือ...

- การปกป้องกระบวนการและเครื่องเนล
 - ทำงานหลายโปรแกรม
 - ป้องกันไม่ให้รบกวน OS – kernel
 - ไม่ให้รบกวนซึ่งกันและกัน

หยุดยั้ง Program นั้นให้ได้ Kill Process

Activity #2: Protection: WHY?

การ protect process และ kernel ทำให้เกิด impact อะไรกับระบบบ้าง และยังต้อง protect อะไรอีกบ้าง เพื่ออะไร

- Protect Kernel ไม่ให้ User ติดตอกับ Hardware โดยตรงโดยที่ไม่ผ่าน kernel เพื่อป้องกันการ Error ที่เกิดขึ้น
- Protect Process เพื่อไม่ให้เกิดปัญหาจนฟ้า
- Reliability: buggy programs only hurt themselves
- Security and privacy: trust programs less
- Fairness: enforce shares of disk, CPU

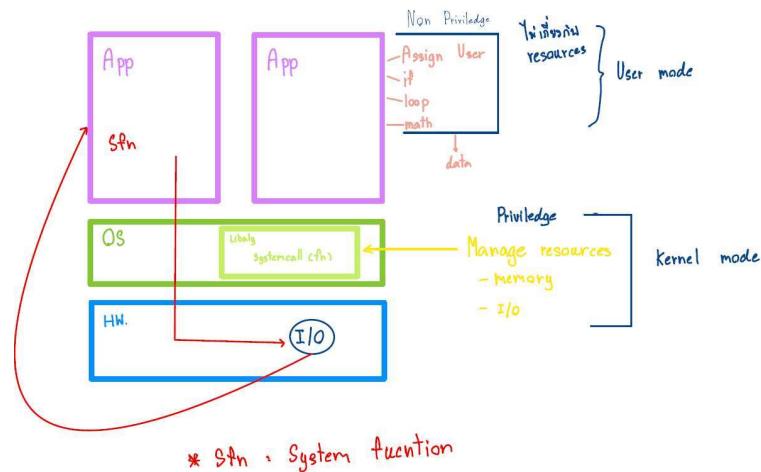
Protection: How? (HW/SW)

- ตรวจสอบ Resources ของตัว Hardware ว่าใช้อะไรบ้าง มีการจัดสันทรัพยากรได้หรือไม่ เช็คการทับซ้อนการทำงานของ Hardware

- ตรวจสอบ Process การทำงานว่ามีการทำงานที่ซับซ้อนหรือไม่ เพื่อให้ Computer ทำงานได้ถูกต้องมีประสิทธิภาพมากที่สุด

- ในระหว่างอยู่ใน kernel mode จะทำงานได้แบบไม่ มีข้อจำกัดใช้ Instruction ที่เป็น Privilege ได้

- ใน User mode



* SFn : System function

Hardware Support:

Dual-Mode Operation

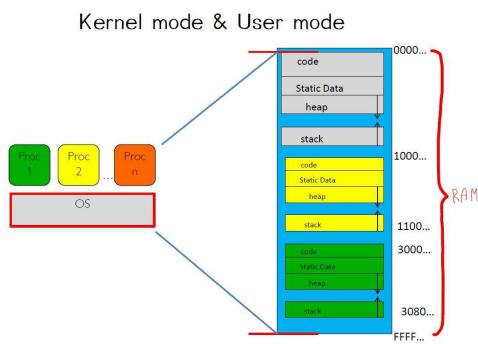
- On the x86, mode stored in EFLAGS register
 - On the MIPS, mode in the status register
 - Privileged instructions
 - Available to kernel
 - Not available to user code
 - Limits on memory accesses
- user ไปเป็น Kernel > เกิดจากการเรียกใช้ System call
- To prevent user code from overwriting the kernel
 - Timer ****Interrupt Routine จะอยู่ใน OS ***Kernel จะถูกปลูกขึ้นมาทำงาน
 - To regain control from a user program in a loop
 - Safe way to switch from user mode to kernel mode, and vice versa

Process VMs **ส่วนหนึ่งของระบบปฏิบัติการ

- GOAL:

- Provide an isolation to a program
- โปรแกรมแต่ละโปรแกรมจะไม่รบกัน
- Portability (Program)

Program สามารถเอาไป Run บน Computer ต่างสเปกได้



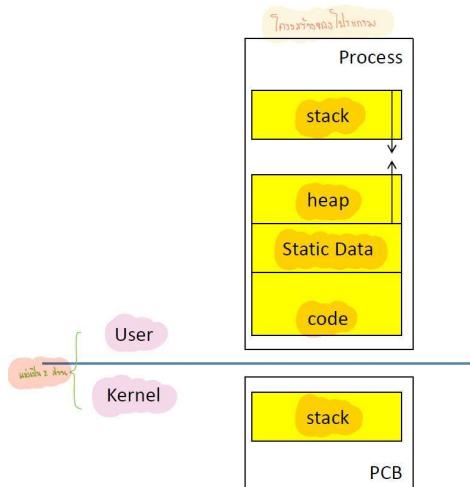
Process Abstraction

Process ต้องมีสองด้านที่เหมือน Thread(เป็น Virtual Microprocessor), Address space

Process

2 parts

- PCB in kernel
- Others in user



Process Control Block: PCB

จัดการให้เป็นปัจจุบัน(Update)

- Status (running, ready, blocked, ...)
- Registers, SP, ... (when not running)
- Process ID (PID), User, Executable, Priority, ...
- Execution time, ...
- Memory space, translation tables, ...

Implementing safe Kernel Mode Transfers

- เมื่อ Process State ว่า Instruction จะเริ่มต้นที่ register เวลาแน่นอนแล้ว
เก็บไว้

- ต้อง handle weird/buggy/malicious user state ก็คือเช็คว่า

- System calls with null pointers ได้ไหม
- Return instruction out of bounds กลับออกมากจาก function ได้ไหม
- User stack pointer out of bounds

- ต้องไม่ให้ bug ทำให้ Kernel ทำงานผิดปกติ หรือล้มเหลว

- User ต้องไม่รู้ว่าการ Interrupt เกิดขึ้น