

# **Operating System Midterm**

Edited 0  
25/1/2018

## Computer and OS System Overview

- Computer
  - computer ประกอบด้วย 3 layer
    - Application Programs
      - เป็น layer ที่ program ที่เราเขียนทำงานอยู่
    - System Programs
      - มี 2 layer
        - Compiler, Interpreter
          - สำหรับแปลจาก high level language ไปเป็น binary language
        - Operating System
          - ประสานงานระหว่าง App กับ hardware
      - Hardware
        - มี 3 layer
          - Machine Language
          - Microarchitecture
          - Physical device
  - Operating System
    - หน้าที่
      - มี service ให้เรียกใช้งาน (เป็นเหมือน function โปรแกรมในภาษาจะไปเรียกมาใช้งานอีกที)
      - เป็นโล่ป้องกัน user กับ hardware
      - จัดสรรทรัพยากรให้กับ process ต่างๆ (CPU, memory, I/O)
      - ควบคุมการ execute program
    - Basic functionality (starting from 0) (p.4)
      - การทำงานจะเริ่มที่ state start
      - จากนั้นใน state fetch จะเป็นการนำ instruction(คำสั่ง) มาจาก memory
      - จากนั้นใน state execute จะเป็นการแปลและทำงานตามคำสั่ง
      - เมื่อทำงานเสร็จจะจบที่ state halt
    - Closer to reality view
  - Basic Element of a Computer System
    - CPU
      - Processor
      - Register
        - เป็นหน่วยเก็บข้อมูลที่เล็กที่สุด, เข้าถึงได้เร็วที่สุด (เพราะอยู่กับ processor เลย ขนาด memory ยังต้องมาทาง bus)
        - มี 2 ชนิด
          - User visible
            - เข้าถึงได้ทาง machine language
            - การใช้ register แทน memory สามารถช่วยให้โปรแกรมทำงานได้เร็วขึ้น
            - แยกเป็นอีก 2 ชนิดย่อย

- Data, Address
- Control and status
  - Processor ใช้เพื่อควบคุมการ execute
  - OS ใช้ในการควบคุมการ execute program
- Basic C&S register
  - PC (program counter)
    - ใช้เก็บตำแหน่งของ instruction ที่จะถูก fetch
    - ตอนเริ่มทำงานจะมีคนเอาค่ามาให้ pc เพื่อให้รู้ว่าเริ่มทำงานที่คำสั่งไหน (ถ้าจำไม่ผิดจะชื่อ bootstrap)
  - IR (Instruction Register)
    - ใช้เก็บคำสั่งที่ไป fetch มา
  - PSW (program status word)
    - ใช้เพื่อเก็บสถานะต่างๆ เช่น
      - Condition code
      - Interrupt enable/disable (เก็บสถานะว่า รับ interrupt ได้มั้ย)
      - Supervisor, user mode (สิทธิในการเข้าถึง resource)
- I/O module
- Main memory
  - volatile
- System bus
  - เป็นเส้นทางผ่านที่เทอใช้ไปหาคนอื่น
  - ใช้ในการสื่อสารระหว่าง CPU, I/O, memory
- Instruction cycle
  - เมื่อ processor ไป fetch instruction มาจาก memory
    - PC จะเก็บตำแหน่งที่จะไป fetch มา
    - ไป fetch มาเสร็จก็เอา instruction ไปใส่ไว้ใน IR
    - พอ fetch แล้ว PC ก็เพิ่มขึ้น
  - Type of Instruction
    - process-memory
      - คำสั่งในการเข้าถึงข้อมูลบน memory เช่น อ่านค่าตัวแปร
    - process-I/O
      - คำสั่งในการเข้าถึง I/O เช่น พิมพ์, รับ input
    - Data process
      - คำสั่งคำนวณค่าต่างๆ
    - Control
      - คำสั่งที่ใช้ควบคุม flow เช่น if, loop, jump
  - \* CPU จะยังทำงานอยู่แม้เกิด interrupt เพราะว่า cpu ก็ต้องทำงานเพื่อรับ interrupt นั้น

## - Interrupt

- คือการขัดจังหวะ, การที่ต้องหยุดทำงาน มีได้หลายสาเหตุ เช่น โปรแกรมผิดพลาด, หมดเวลาให้ resource, รอ I/O ทำงานเสร็จ
- Interrupt handler คือโปรแกรมที่ไว้สำหรับจัดการว่าเมื่อเกิด interrupt แล้วต้องทำอะไรต่อ โดยจะมี list ของ interrupt code เพื่อเก็บว่าถ้าเกิด code นี้จะทำอะไร
- อะไรไม่รู้ ฟังใจไว้ มี 3 อย่าง
  - Polling  
คือการไป check ว่าค่าที่เราต้องการ (อาจมาจาก I/O) มาหรือยัง ซึ่งก็ต้อง interrupt เพื่อไป check
  - Interrupt driven
  - DMA  
คือการส่งงานที่ทำอยู่ไปให้คนอื่นช่วย พอเค้าทำงานเสร็จแล้ว เค้าก็จะมา interrupt เพื่อบอกเรา
- Interrupt cycle
  - Process สามารถ disable interrupt เพื่อปฏิเสธ interrupt ไม่ยอมให้ตัวเองหยุดทำงานได้ และก็ สามารถ enable interrupt ได้เช่นกัน
  - เมื่อ process ทำงานจะเช็คว่ามี interrupt หรือไม่ ถ้าไม่มีก็ fetch instruction ต่อไป
  - ถ้าเกิด interrupt ก็จะดูว่าได้ enable อยู่รึเปล่า ถ้า enable ก็ไปเรียก handler มารับผิดชอบชีวิตเราต่อ
  - ถ้า disable ก็ไม่แคร์อะไร ทำงานต่อไป

## - OS Overview

- เป้าหมายในชีวิตของ OS
  - Provide service ให้กับ system program
  - Shield ไม่ให้ program กับ hardware มาเจอกัน
  - จะคอยจัดการ resource ให้
  - ควบคุมการ execute program
  - เพิ่มความสะดวกให้กับ program เพราะเรื่องยากๆ os จะทำให้
  - เพิ่มประสิทธิภาพในการทำงาน
  - สามารถพัฒนาต่อได้ โดย os จะตอบสนองต่อ hardware ใหม่ๆ
- โปรแกรมเรียกชื่อว่าพีใหญ่
  - แท้จริงแล้ว os ก็เป็นแค่ program หนึ่ง ซึ่งยังต้องใช้ cpu, memory เหมือนโปรแกรมอื่นๆในการทำงาน
  - แต่ os นั้นเกิดมาเพื่อจัดการดูแลโปรแกรมอื่นๆ
- OS kernel
  - คือส่วนที่เป็นหัวใจของ os จะทำงานอยู่ตลอดเวลา (แสดงว่าจะอยู่ใน main memory ตลอดเวลาด้วย)

----- เบื่อแล้ว เค้าค่อยอ่านต่อ -----

## Process description and Control

- What is process ?
  - Process คือ program ที่อยู่ระหว่างการประมวลผล
  - Process จะถูกจัดการชีวิตโดย OS
    - การจัดสรรทรัพยากร
    - จัดตารางเวลาในการเข้า execute
    - การสื่อสารกับ process อื่น
  - Process จะประกอบด้วย instruction หลายนั่น
  - Process จะมีความต้องการในชีวิตนี้อยู่ 3 อย่าง (ในชีทไม่มี ไม่ต้องหาถูกเขียนเอง)
    - ต้องการ cpu
      - ถ้าต้องการมากเป็นพิเศษเรียก cpu-bound process
    - ต้องการ file หรือ i/o
      - ถ้าต้องการมากเป็นพิเศษเรียก i/o-bound process
    - ต้องการ memory
- State Process model
  - Two-state process model (p.3)
    - State not-running
      - Process อยู่ใน queue เพื่อรอให้ processor
      - เมื่อถึง queue จะเข้าไปอยู่ใน state running
    - State running
      - Process ที่กำลังใช้ processor ประมวลผล
      - เมื่อหมดเวลา จะปล่อย processor แล้วไปอยู่ที่ not-run
    - หาก process ต้องรอข้อมูลจาก IO แล้วยังอยู่ใน running จะเปลือง cpu เปล่าๆ
  - Five-state process model (p.4-5) - Blocked
    - State new
      - สร้าง process แล้วแต่ยังไม่มาอยู่ใน memory
      - ถ้า memory เหลือ จะถูก admit และย้ายไปยัง ready
    - State ready
      - Process ที่อยู่ใน memory
      - รอคิวเพื่อใช้ processor
    - State running
      - กำลังใช้งาน processor
      - หาก timeout จะกลับไป ready
      - หากต้องรอ IO จะไปที่ blocked
    - State blocked
      - เข้าคิวคำตอบจาก IO
      - ได้คำตอบแล้วจะกลับไป ready
      - ในความเป็นจริงแล้ว IO ไม่ได้มีแค่อย่างเดียว (อาจเป็นแป้น, จอ, ..) แต่ละ IO ก็จะมี blocked queue เป็นของตัวเอง จะทำงานให้กับ process ที่อยู่ในคิวตามลำดับ
    - การที่ process อยู่ใน state blocked นั้น ความจริงแล้วก็ยังใช้พื้นที่ใน memory ถ้าหาก IO ทำงานช้ามากๆ process ก็จะค้างใน blocked เยอะ

memory ก็จะเต็ม ทำให้ไม่สามารถ admit process ใหม่ได้

ทำให้ CPU utilization (อัตราการใช้งาน cpu ยิ่งเยอะยิ่งดี เพราะใช้คุ้มค่า) ต่ำ

- การที่มี process อยู่ใน ready queue บ้าง เรียกอาการนี้ว่า cpu busy
- แต่ถ้ามีแต่ process อยู่ใน blocked queue อาการนี้เรียกว่า cpu idle
- Six-state process model (p.6) - Suspend
  - ปรับปรุง State block
    - ถ้าหาก process อยู่ใน state นี้ แล้ว os เห็นว่ามีอยู่ไปก็รัก memory ญ OS จะทำการ คัดลอกข้อมูลของ process จาก memory ไปไว้ใน disk แทน เพื่อเพิ่มพื้นที่ใน memory
  - เพิ่ม State suspend
    - เป็น process ที่ถูกเก็บข้อมูลไว้ใน disk
    - ถ้า process ได้รับข้อมูลจาก IO แล้วจะทำการคัดลอกข้อมูลกลับมาที่ memory และย้ายไปอยู่ที่ ready state
  - การ swap process ไปมาต้องมีค่าใช้จ่าย เพราะต้องใช้ทั้ง cpu และ io
  - นอกจากจะช่วยเพิ่มพื้นที่ใน memory แล้ว ยังสามารถใช้เพื่อป้องกัน deadlock ได้อีกด้วย (น่าจะกรณี deadlock ที่แย่ง memory กัน) แต่ deadlock ก็สามารถเกิดกับ process ที่อยู่ใน running state ได้เหมือนกัน
- Seven-state process model (p.7) - Two Suspend
  - เพิ่ม State Ready/Suspend
    - สำหรับ process ที่ถูก suspend มาจาก running
- OS control structure (p.10, 12)
  - Process Image
    - โครงสร้างของข้อมูล process จะแบ่งเป็น 2 ส่วนหลัก
      - Process Control Block
        - ใช้สำหรับเป็น status ของ execution
        - แบ่งเป็น 3 ส่วน
          - Process Identifier
            - Id ของ process, parent, user, ...
          - Process State Information
            - เก็บว่าอยู่ state ไหน
          - Process Control Information
            - Scheduling information (priority, ..)
            - Process memory table
            - Resource
            - Etc.
      - User Block (ตั้งชื่อเองจำ เห็นในชีทไม่มีชื่อ 555)
        - ใช้สำหรับในการ execute

- แบ่งเป็น 3 ส่วน
  - User stack
    - ใช้เก็บพวก call function
    - เก็บพวก return address, parameter, local variable
  - Private User Address space
    - เก็บพวก data, program
  - Shared User Address space
- OS Table (p.11)
  - OS จะมีตารางที่เก็บ address ของข้อมูลต่างๆเพื่อใช้ในการทำงาน โดยมี 4 ตาราง
    - Memory table
      - เก็บข้อมูลการจัดสรร memory ให้กับ process
      - เก็บข้อมูลการจัดสรร secondary memory ให้กับ process
      - ปกป้อง attribute สำหรับการเข้าถึง shared memory
      - ข้อมูลสำหรับการทำ virtual memory
    - I/O table
      - เก็บข้อมูลว่า I/O วางอยู่หรือให้ใครไปแล้ว
      - เก็บสถานะของการทำงานของ I/O
      - เก็บตำแหน่งใน memory สำหรับการย้ายข้อมูลไป/มาจาก IO
    - File table
      - เก็บว่ามี file อะไรอยู่บ้าง
      - เก็บตำแหน่งใน secondary memory
      - เก็บสถานะปัจจุบัน, attribute
      - บางครั้งหน้าที่ส่วนนี้ก็เป็นของ File management system
    - Primary Process table
      - ภายในเก็บ process control block ของแต่ละ process (ส่วนแรกของพาที่แล้ว ↑↑)
      - ภายในจะเก็บตำแหน่งของ process image ด้วย (เก็บตรงไหนวะ)
- Process Creation
  - Process จะถูกสร้างขึ้นมาจากไหนบ้าง
    - Execute user program
    - Os สร้าง process ขึ้นมาเพื่อ provide service
    - Process สร้าง process ลูกขึ้นมาเพื่อทำงาน
  - OS จะต้องทำอะไรบ้างเมื่อ process เกิดขึ้นมา
    - กำหนด process ID
    - จัดสรร memory ให้
    - กำหนดค่าเริ่มต้นให้กับ process control block
    - กำหนดค่า linkaged (น่าจะเป็นพวก เอา process นี้ไปใส่คิวหรือตรงไหนก็ได้ที่เกี่ยวข้องกับระบบ)
- Process switching
  - Process จะ switch(สลับ cpu กันใช้) เมื่อไหร่บ้าง

- Clock interrupt หมดเวลา(ใช้cpu)แล้ว เสร็จต้องไป
  - I/O interrupt อยากใช้ I/O ก็ออกไปจาก cpu สิ
  - Memory fault หาข้อมูลใน memory ไม่เจอต้องไปหา disk ปลอ่ย cpu สิ
  - Fork child สร้าง process ลูกมา ก็เอา cpu ให้ลูกไป
  - Trap เกิด error
- 
- Context switch (p.15)
    - การ switch process ทำโดย os ซึ่งจะทำให้เร็วมากๆ
    - ขั้นตอนการ switching มีดังนี้ (สมมติ switch จาก P1 ไป P2)
      - Save register (PC, ..) ไปไว้ใน PCB1 (Process Control Block)
      - ย้าย PCB1 ไปใส่ไว้ใน queue (ready, blocked แล้วแต่สถานการณ์)
      - เลือก process มา (แล้วแต่ schedule algorithm), สมมติเลือก P2
      - Update PCB2
      - Update memory-management
      - เอาค่าของ register ที่เก็บไว้ใน PCB2 ออกมาให้ register
  - Interprocess Communication
    - การทำงานของ process ที่อยู่ในระบบ จะมีอยู่ 2 แบบ
      - Independent ไม่ยุ่งเกี่ยวกัน
      - Cooperation ทำงานด้วยกัน
    - Cooperation เพื่ออะไร
      - เพื่อแลกเปลี่ยนข้อมูล
      - เพื่อเพิ่มความเร็วในการประมวลผล
      - เพื่อความสะดวกและเป็นระบบ
    - การสื่อสารระหว่าง process
      - เรียกว่า interprocess communication (IPC)
      - IPC จะมี 2 แบบ
        - Shared memory
        - Message passing



## Threads

- Process and Thread
  - Thread เป็นเหมือนส่วนการทำงานย่อยของ process
  - Process สามารถอยู่ได้โดยไม่มี thread (มองเสมือนว่ามีแค่ main thread)
  - แต่ thread ไม่สามารถอยู่ได้ หากไม่มี process
  - Process เก็บอะไรบ้าง
    - Virtual address ของ Process image
    - Global variable, file, child process
  - Thread เก็บอะไรบ้าง
    - Execution state (แต่ละ thread ก็จะมี state เป็นของตัวเอง)
    - ที่เก็บข้อมูลเมื่อไม่ได้ทำงาน
    - Execution stack
    - Local variable
  - การ suspend process ไปไว้ที่ disk ก็เท่ากับว่าเป็นการ suspend thread ทั้งหมดของ process ไปไว้ที่ disk ด้วย
  - การฆ่า process ก็เท่ากับว่าได้ฆ่า threads ทั้งหมดใน process ไปด้วย
- Multithreading
  - OS ต่างๆ สามารถรองรับระบบ multithread ได้ต่างกัน
    - 1 process 1 thread
      - MS-DOS
    - 1 process หลาย thread
    - หลาย processes 1 thread
      - unix
    - หลาย processes หลาย threads
      - Windows, solaris, linux, mach
- ประโยชน์ของ thread
  - ใช้เวลาในการสร้างและตายน้อยกว่า process
  - ใช้เวลาในการ switch ระหว่าง thread ใน process เดียวกันน้อย
  - เนื่องจาก thread ที่อยู่ใน process เดียวกัน ใช้ memory ร่วมกัน เลยสามารถติดต่อกันได้เองโดยไม่ต้องพึ่ง kernel
  - สามารถทำให้ process ใช้งาน I/O และ processor พร้อมกันได้
  - สามารถทำให้ process ใช้งาน processor หลายอันพร้อมกันได้
  - สามารถทำงาน background พร้อมกับงาน foreground ได้
  - สามารถทำงานแบบ asynchronous ได้
  - เพิ่มความเร็วในการ execute
  - Modular program structure
- Thread state
  - Spawn (วางไข่)
  - Block
  - Unblock
  - Finish

- Thread Level
  - User-level threads (p.17)
    - การจัดการ thread
      - Process จัดการ thread เอง ว่าจะให้ cpu กับ thread ไหน
      - Kernel มองไม่เห็น threads
      - ใน kernel มีเพียง process table
      - ในแต่ละ process จะมี threads table เป็นของตัวเอง
    - Runtime system (thread library) จะมีหน้าที่รับผิดชอบดูแลจัดการ thread
    - สามารถปรับ thread scheduling algorithm ได้เอง
    - สามารถใช้งานบน os ไหนก็ได้
    - ถ้าเกิดมี thread หนึ่งเรียก system call แล้วถูก block ก็จะทำให้ process นั้นถูก os block ไปด้วย thread อื่นก็ไม่สามารถทำงานต่อได้
  - Kernel-level threads (p.21)
    - การจัดการ threads
      - Kernel จะเป็นคนถือข้อมูลทั้งหมดของ process และ thread
      - Kernel จะมีทั้ง process table และ thread table
      - Kernel จะจัดการเองว่าจะให้ cpu กับ thread ไหน
    - แก้ข้อเสียเรื่อง block process ของ user-level threads
    - แต่มีข้อเสียในการที่ os ต้องจัดการ threads เองทั้งหมด ทั้งการ create และ terminate
  - Hybrid
    - รวมข้อดีของทั้ง 2 แบบ
    - การจัดการ threads
      - สร้าง thread ใน user space (user-level)
      - Schedule thread ด้วย application (user-level)
      - Kernel ยังมองเห็น thread ทำให้การ block เฉพาะ thread ยังสามารถทำได้ (kernel-level)

## Uniprocessor Scheduling

- Level of scheduling (p.5)
  - Long-term scheduling
    - ใช้สำหรับตัดสินใจว่าจะให้ process ไหน admit เข้ามา
    - เป็นผู้ควบคุม degree of multiprogramming (น่าจะหมายถึงจำนวนของ process ที่ทำงานได้พร้อมกัน)
  - Medium-term scheduling
    - ใช้สำหรับตัดสินใจว่าจะให้ process ไหนออกจาก state new แล้วเข้าไปยัง state ไหนต่อ (ready, ready suspend)
    - ใช้ตัดสินใจว่าจะให้ใครออกจาก suspend
  - Short-term scheduling
    - เรียกอีกอย่างว่า dispatcher
    - ทำงานบ่อยมาก
    - จะถูกใช้งานเมื่อ
      - Clock interrupt
      - I/O interrupt
      - Os call
      - Signal
    - หลักการ, กฎเกณฑ์
      - User-oriented
        - Turnaround time
          - ผลรวมของเวลาที่ใช้ประมวลผลและเวลาที่รอคอยทั้งหมด
        - Response time
          - อะไรเร็ว
        - Deadline
          - อะไรเนี่ย
        - Predictability
          - อะไรไม่รู้
      - System-oriented
        - Throughput
          - ไม่อ่านแล้วว
        - Processor utilization
          - เปอร์เซ็นที่ processor busy ในหนึ่งหน่วยเวลา
        - Fairness
          - ตกลงหัวข้อนี้มันคืออะไร
        - Enforcing priority
          - ให้อ่านอะไรอยู่เนี่ย
        - Balancing resource
          - หิวข้าวแล้ว
      - Performance-related
        -
  - Process Scheduling Algorithm

- Decision mode
  - Nonpreemptive
    - ไม่มีการเรียก cpu คืนจาก process
    - เมื่อ process ต้องรอ I/O หรือใช้งานจนเสร็จ จะปล่อย cpu เอง
  - Preemptive
    - Os สามารถเรียกคืน cpu จาก process ได้
- Related value
  - Arrived time
    - เวลาที่ process มาถึง queue
  - Service time (cpu-burst time)
    - เวลาที่ process ต้องการใช้ cpu
    - การจะหาค่า **service time** นั้นต้องมีค่าใช้จายด้วย
  - Waiting time
    - เวลาที่ process ต้องรออยู่ใน queue จนกว่าจะทำงานเสร็จ
  - Turnaround time
    - เวลารวมที่ process ใ้รอและใช้ cpu
- First-Come-First-Served (FCFS)
  - พิจารณาจาก arrived time
  - ใครมาก่อนก็ให้ไปเลย
  - Nonpreemptive
  - ถ้า **cpu-bound process** มาก่อน จะทำให้ **io-bound process** ต้องรอนานมาก
  - แต่ก็มีข้อดีตรงที่ **context switch** ไม่ต้องเกิดขึ้นบ่อย
- Round-Robin
  - พิจารณาจาก arrived time
  - กำหนดระยะเวลาที่แต่ละ process จะใช้ cpu ได้ เรียกว่า quantum
  - Preemptive
- Shortest Process Next
  - พิจารณา service time
  - Nonpreemptive
  - อาจเกิด **starvation** ได้กับ **cpu-bound process**
- Shortest Remaining Time
  - พิจารณา service time ที่เหลืออยู่
  - Preemptive
- Multilevel Feedback Queue

## Concurrency : Mutual Exclusion and Synchronization

- Critical Region
  - คือส่วนของ code ที่ทำการเข้าถึง shared resource
- Mutual Exclusion
  - เป็นเหตุการณ์การ schedule แบบหนึ่ง
  - เงื่อนไขที่จะทำให้เกิด mutual exclusion มีดังนี้
    - เมื่อมี process เข้าไปใน critical region บน resource ใดแล้ว ห้ามไม่ให้มี process อื่น เข้าไปใน critical region บน resource นั้นอีก
    - ถ้า resource นั้นไม่มี process ใช้อยู่, process ที่มาถึงแล้วขอใช้ ต้องได้ใช้
    - ถ้า process ตายใน critical region OS ต้องมีวิธีการจัดการ
- วิธีการที่จะช่วยให้เกิด Mutual Exclusion แบ่งออกเป็น 3 แบบ
  - Hardware Support
    - Interrupt disabling
      - เมื่อจะเข้า critical region ให้ทำ disable interrupt และเมื่อออกจาก critical region ค่อยทำการ enable interrupt กลับมา
      - มีข้อเสียคือเชื่อใจ process มากเกินไป ถ้าเกิด process ไม่ยอมคืน cpu ทำไงล่ะ
      - จำกัด cpu ให้กับแค่ process เดียว
      - ไม่สามารถใช้วิธีนี้ได้กับ multiprocessor
    - Special Machine Instruction
      - เป็นคำสั่งพิเศษที่ machine มีให้ เพื่อช่วยให้ทำ ME ง่ายขึ้น
      - มีคุณสมบัติเป็น atomic คือจะไม่ถูก context switch จนกว่าคำสั่งนี้จะทำงานเสร็จ
      - ได้แก่
        - Test and Set
          - testset(resource)
        - Exchange
  - Software Support
    - Busy waiting
      - ถ้าเกิดว่าต้องการเข้า CR แต่ resource ไม่ว่าง ก็จะต้องวนลูปรอเล่นๆ ให้เปลือง cpu หน่อย
    - Strict Alternation
      - การกำหนดลำดับของ process ไว้เป๊ะๆ เช่นเขียนโค้ดในลักษณะที่ว่า p0 ต้องเริ่มก่อน p1 นะ
  - System Support