บทบาทหน้าที่ OS

- ผู้ชี้ขาด
  - ทรัพยากร, แยก, การสื่อสาร
- จำลอง
  - ให้ sw มองเห็น hw เป็นแบบเดียวกันหมด
- ประสาน
  - ทำให้ user, app, sys, hw เชื่อมต่อกัน

การประเมิน OS

- ความน่าเชื่อถือ, พร้อมใช้งาน
- security
  - isolation, hacking
- portability
  - AVM, API, HAL
- performance
  - ทำนายได้, ยุติธรรม
- adoption*
  - ความนิยมของผู้พัฒนา software

OS คืออะไร

- เซ็ตของ sw
- sw ที่ช่วยซ่อนความซับซ้อน

kernel

- ♡ ของ OS
- จัดการทรัพยากร
- เป็นสะพานระหว่าง app ⇔ hw

goal of os

- ป้องกัน kernel, process
  - → ห้ามแทรก kernel
  - → ห้ามยุ่ง
  - run หลายโปรแกรม
- สาเหตุที่ป้องกัน
  - relaiability
  - security
  - fairness
- ปกป้องยังไง
  - sw → การจำกัด process, sys call
  - hw → memory addr tran, dual mode operation

dual-mode operation

- kernel mode → ทำได้ทุกอย่าง
- user mode → ทำได้บางอย่าง

hw support

- privilleged instruction → ตอน decode เช็ค flag
- การจำกัดการเข้าถึง mem
- timer → แบ่งกันใช้งาน

vm

- system vm
- process vm → app มองเห็น hw_{os} เหมือนกัน → app เขียนบน os ไหน (port) รันได้บน os อื่นทุกตัวจริง → ห้ามยุ่งกัน
  - instance of program ที่มีสิทธิ์ขาดเดี่ยว

↳ ประกอบด้วย : thread, address space
↳ ที่เก็บสิทธิ์ {mem, I/sys call}}

↳ เก็บไว้ที่ user "process", ใน kernel "pcb"
↳ เก็บพฤติกรรมโปรแกรม

การเปลี่ยน mode

↳ user → kernel
- ↳ interrupt
- ↳ exception
- ↳ sys call

{...}

↳ kernel → user
- ↳ after creat process
- ↳ return from {...}
- ↳ process/context switch
- ↳ up call

การเปลี่ยน mode แบบปลอดภัย

↱ ห้าม kernel แตก

↳ จัดการพฤติกรรมแปลกๆ ของ user ได้ → ดูชุดค่าส่ง parameter, pointer ☐

↳ ห้าม app detect interrupt

interrupt ให้ปลอดภัย

↳ interrupt vector → ชี้ไปที่ interrup rountine → func handle

↳ kernel interrupt stack → เก็บข้อมูล

↳ interrupt masking → ดูว่า interrupt ไหนเกิดซ้อนได้

↳ ชุดคำสั่ง atomic → ที่เดียว → กันเหตุการณ์แทรกระหว่างทาง

↳ จะโปรแกรมที่โดน interrupt ต้องทำงานได้เหมือนเดิม

การเกิด int

↳ ระหว่างเกิด
- ↳ มีการดู interrupt rountine โดย hw → ชี้ไปที่ handle fun
- ↳ ① save reg u
  - ② pc → handle func k

↳ หลังจากเกิด
- ↳ ③ pc → ของเก่างาน (แรก) → func abt save k
  - ④ save other reg k

↳ จบ
- ↳ ⑤ restore k
  - ⑥ switch to U

hw support การดู interrupt

↳ ① ↱ ① ↳ ① ไม่มีใครรู้

↳ interrupt masking

↳ level of interrupt

kernel sys call handler

- ↳ vector → เหมือนใน interrupt
- ↳ ดูค่า
- ↳ copy ค่า
- ↳ validate ⤴ — copy for security
- ↳ return value

shell

- ↳ แปลง command user → syscall

window CreateProcess 1 api

(มีลักษณะ)

- ↳ สร้าง pcb → สร้าง addr space → load program เข้า mem → เอา argument เข้าไปโปรแกรม → เริ่มสร้าง hw context {pc, other env} → แจ้ง scheduler ว่าเปิด

unix fork, exec, wait, signal 4 api

fork ①, exec ③, wait ②, signal ④

③ → ให้ลูก, run

④ ลูก → ② แม่

① สร้างได้ if memory

③ สร้างได้ program ใหม่

② รอแต่ไม่รอ → ③ เกิดไวกว่า ②

- ↳ สร้าง pcb → สร้าง addr space (copy จากแม่) → ก็อป execution context จากแม่ → แจ้งว่าพร้อม

unix I/O

- ↳ มีรูปแบบเดียวทุก I/O
- ↳ open, byte-oriented, kernel buffer-read&write, close
  - ↳ ทำได้ทุกอย่าง 1 ต่อเสร็จ
    - ↳ อาจไม่ปลอดภัยกับการ interrupt แทรก

OS structure

- ↳ mono → ทุกอย่างใน kernel → ใหญ่เร็ว (↑ ปรับยาก)
- micro → เล็ก → ช้า → ปรับง่าย

Interprocess Communication

- ↳ Producer-Consumer → ทำงาน one way } both working
- ↳ Server-Client → 2 way }
- ↳ Peer Server → 2 way → 1 worker enough