OS Sys Prog

① Introduction

Operating system = intermediary between users and computer hardwares
Goal: Solving problems easier, convinient to use, efficiency of hardware
- resource utilization
- optimized for usability
- resource locator
- control program

OS structure

- Multiprogramming → can be single user
  - job scheduling
  - context switching
- Time sharing (multitasking) → multi-user
  - interactive
  - response time <1 second
  - swapping
  - virtual memory

Kernel Data Structure
- Singly linked list
- Doubly linked list
- circular linked list
- BST
- Hash map
- Bit map

OS Operation

- hardware interrupt driver
- software error —create→ exception —request→ OS service
- process problems
- Dual-mode (user mode vs kernel mode)
  - hardware
  - mode bit
  - privileged instructions, only executable in kernel mode
  - system call changes mode to kernel and return to user mode
- multi-mode ex VM

Computing Environments

- Traditional – Portal, thin clients, Wireless networks, firewalls
- Mobile – GPS, Gyroscope, Cellular data (Apple iOS, Google Android)
- Distributed – Network
- Client-Server
- P2P
- *Virtualization – OS within OS, Emulation (CPU ต้นทางไม่เหมือนปลายทาง) ไม่นิยมเท่าไหร่
  Virtualization (ภาษาเดียว OS) วิ่ง Guest OS บน Host OS
  ข้อดี: Developing and testing, ทรหดลอง ล้มพังก็คืน, data center
- Cloud computing – Public, Private, Hybrid (SaaS, PaaS, IaaS)
- Real-Time Embedded – เข้มงวดเรื่องเวลา, ถูกต้องในเวลาที่กำหนด ใช้ OS เฉพาะ ex. autonomous car

Open-Source OS

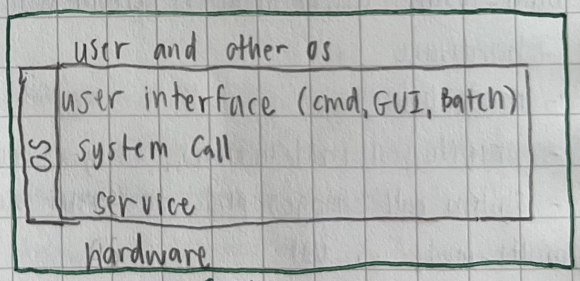
- available in source-code format, not binary closed-source
- Copy left = ฟรีต่อได้ ไม่ลิขสิทธิ์ ex. GNU/Linux, BSD UNIX

② System Structure

OS provide an environment for execution of programs

OS provide **services**

- User interface (Command Line, GUI, Batch)
- Program execution – Load program to memory and run
- I/O operation
- File system manipulation – R/W, create, delete, permission
- Communication
  - Same computer – shared memory
  - Through network – message passing
- Error detection



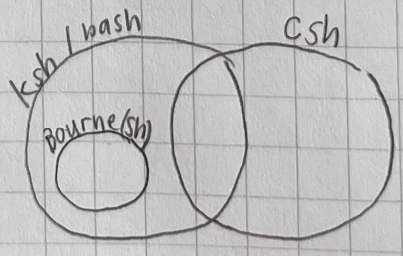
OS View

Functions ensuring operation flow

- Resource allocation – multiple user
- Accounting – User
- Protection and security
  - Protection = all access to system resource is controlled
  - Security = user authentication

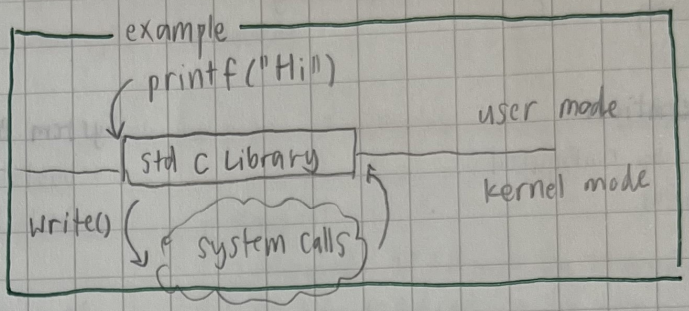
Command Interpreter

- in kernel or systems program
- User type command to shell → execute
  - Some commands built-in
  - Now we can add new features, doesn't require shell modification



System calls

= programming interface to the services provided by the OS
written in a high-level language (C, C++)
- Application Program Interface (API) — program access via this
- อยู่ใน kernel
- portability
- parameter passing



OS Design and Implementation

- not solvable
- LINUX has proven successful
- defining goals and specs
- hardware ส่งผลต่อการเลือกใช้ OS

- user goal : ใช้ง่าย รวดเร็ว
- system goal : ออกแบบง่าย สร้างง่าย ดูแลง่าย

Principle

Policy : What will be done
Mechanism : How to do it

- แยกกันเพื่อ flexibility แยกกันเปลี่ยนได้
- หน้าที่ของ Layer
- ตามหลักสร้างด้วยหลายภาษารวมกัน

Hardware connecting — assembly
Kernel — C
System Program — C, C++, PERL, Python, Shell scripts

high-level = easy
low-level = performance

OS Structure

- **Simple Structure** (MS-DOS)
  - most fuctionality in the least space
  - no modules
  - not separated
- **UNIX**
  - limited by hardware functionality
  - 2 parts — System program and kernel
- **Layer Approach**
  - inner = hardware
    outer = user interface
- **Microkernel** ex. Mach
  - มี kernel เท่าที่จำเป็น → port ง่ายขึ้น, debug ยากขึ้น
  - message passing to user module → เสีย performance

NOW
- **Loadable kernel Module**
  - คล้าย Layer but more flexible
  - OO approach
  - Load module เท่าที่จำเป็นใช้

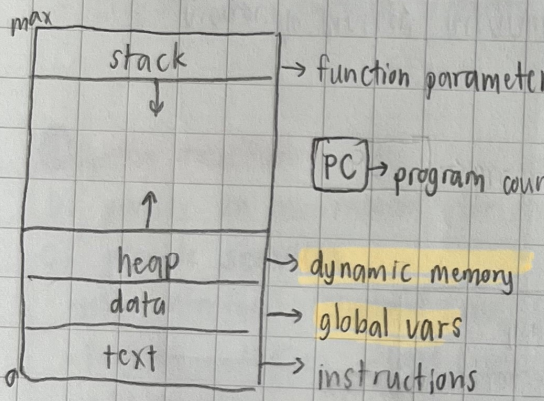
OS Debugging
- OS generate log files (error information) → Protect
- Failure of an application generate core dump (capturing memory)
- OS failure generate crash dump (kernel memory)
- trace listing } performance tuning
- profiling

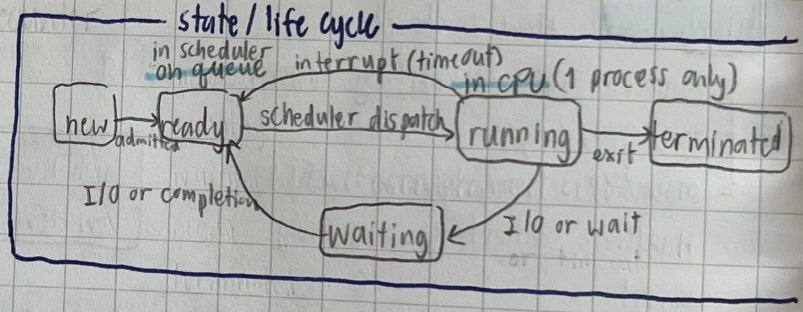
OS Generation
- must be configured
- SYSGEN

System Boot → kernel loading
- bootstrap loader
- boot block

③ Process Concept
- OS executes jobs / tasks / processes

Process = a program in execution , active program (1 program มีได้หลาย process)





Process Control Block (PCB)
- State
- program counter
- CPU registers
- CPU scheduling information (memory allocation)
- Account information
- I/O status information

Thread
- ส่วนย่อยของ Process ที่สามารถทำงานจากหลาย process ขณะเดียวกันได้

Process Scheduling

- Process scheduler in Kernel
- maximize CPU use, quickly switch processes
- Job queue = set of all processes in the system
  Ready queue
  Device queue = set of processes waiting for I/O devices.

Scheduler

- Long-term scheduler (job scheduler) controls the degree of multiprogramming (จำนวนโปรแกรมในระบบ)
- Short-term scheduler (CPU scheduler)
  - which process should be executed next
  - มีแค่อันนี้ก็ได้
  - ต้องเร็ว ส่งผลต่อ performance ของ OS

Process
I/O-bound ใช้ I/O มาก เช่น 3D print
CPU-bound ใช้ CPU มาก เช่น data mining

- Medium-term scheduler
  - ใช้สำหรับการ swap main memory ↔ disk
    disk —swap in→ main memory
  - มีเพราะมี virtual memory

Context Switch

- save the state → load the saved state
- represented in PCB
- overhead
- time dependent on hardware support

Operations on Processes

process creation

- parent create children, forming a tree of processes
- resource sharing options
- execution options
- clone (duplicate) then add new data
  fork() exec()

- แม่ fork ได้ลูก แล้ว wait จนกว่าลูกจะ exit() แล้วแม่ค่อย resume
- init (PID=1) เป็น parent of every process
- PID=0 เป็น child process

Process Termination

- process deallocated
- abort() = แม่สั่งฆ่าลูก
- ถ้าแม่ตาย ลูกก็จะตายด้วย
- Orphan = แม่ตายก่อนลูก แล้วลูกไม่รู้ว่าตัวเองต้องตาย ย้ายไปเป็นลูกของ init แทน (PID=1)
- Zombie = ลูกตาย แล้วแม่ไม่รู้ (เพราะไม่ return exit ไปให้แม่ แล้วแม่ไม่รับ / ไม่ได้ return) "kill ไม่ได้" (อยากkill ก็ kill parent process แทน)

Interprocess communication (IPC)

- shared memory and message passing
- cooperating processes
  - Producer-Consumer problem ex. เครื่องปรับ
    - unbounded-buffer
      bounded-buffer

Message Passing

- without resorting to shared variables
- send and receive through communication link
- direct communication / indirect communication (mailbox)

Synchronization

(Sync) Blocking ส่งไม่ได้จนกว่า receiver จะได้รับ
(ASync) Non-blocking ส่งเสร็จแล้วส่งต่อเลย

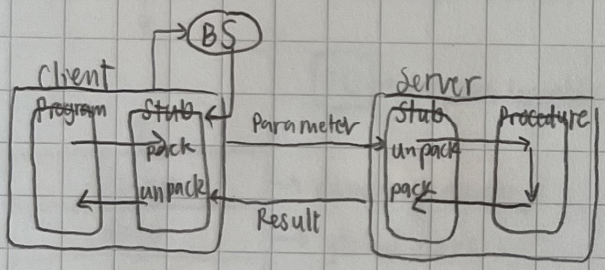
* **Rendezvous** = both are blocking (Deadlock)

Buffering

Zero capacity Sender wait for receiver
Bounded capacity Sender wait if link full
Unbounded capacity

Communication

Remote Procedure Call



Binding Server ผู้ดูแลทั้งหมด service server (คล้าย DNS)
Stub ทำหน้าที่ pack/unpack (parameter passing) ข้อมูลแบบ
Parameter marshalling = pack parameter for network compatability

- big endian and little endian problem
- more failure than local

Pipe ใช้สื่อสารในเครื่องเดียวกันเป็น

Ordinary pipe

- producer-consumer problem
- parent-child
- unidirectional

Named pipe

- bidirectional
- no parent-child needed

④ Multithreaded Programming

- Process creation is heavy-weight, thread creation is light-weight

Client $\xrightarrow{\text{request}}$ Server $\xrightarrow{\text{create}}$ thread
↺ resume listening for client requests

- Benefit

Responsiveness	- Continue execution if part of process is blocked
Resource sharing	- easier than shared memory or message passing
Economy	- Cheaper than process
Scalability	- multiprocessor

Multicore Programming

- Dividing activities, Balance, Data splitting, Data dependency, Testing and Debugging
- Parallelism vs Concurrency
  (Data, Task)
- need hardware threads support
- Threads are Code, Data, File sharing
  Registers and Stack separating
- User threads and Kernel threads
  - user-level
  - library
  - Support by Kernel
  - general purpose OS

Multithreading Models

Many-to-one	One-to-One	Many-to-Many
- One thread blocking cause all to block (because of one kernel thread)	- creating user thread creates a kernel thread - Number of threads per process sometimes restricted due to overhead	- allows the OS to create a sufficient number of kernel threads → thread pool

Thread Libraries

- API for creating and managing threads
- 2 ways of implementing : user space library / Kernel-level library
  - pthread : specifying behavior

Implicit Threading — manage by compiler at runtime library

- Thread pool = สร้าง thread ตามจำนวน → ถ้าถึงจำนวน thread นี้, schedule ใหม่
- OpenMP
- GCD
- Other (Java, TBB)

Threading Issues – Choose one

fork()	–	"new process with n threads" vs "1 new thread"
exec()	–	all thread exec vs 1 thread
Signal handling	–	interrupt → ignore interrupt (ignore ctrl+C) by user-defined
Thread cancelling	┐	thread interrupt vs process interrupt (general)
Thread-local Storage	┘	terminate before return
Scheduler Activations		

⑤ Process Scheduling

Basic Concepts

* Process ประกอบด้วย → CPU intensive program / I/O intensive program → CPU burst distribution
  - CPU Burst → calculation, execution
  - I/O Burst → read from file, write to file

  ทำงานสลับกันไปมา ระหว่าง I/O Burst (multi programming)
  → พยายามเอางานอื่นมาทำระหว่างรอ I/O, เข้า ready queue เพื่อ maximize CPU utilization
  → Scheduler by OS
* CPU Scheduler

  Short-term scheduler (and Long-term scheduler) → คุมเรื่อง batch job
  - คุมกลุ่มกับ timeout และ interrupt ระดับ millisecond
  - Switch between (เมื่อ CPU ว่าง)
    1 running → waiting = I/O
    2 running → ready = Timeout
    3 waiting → ready
    4 terminates

  1,4 = nonpreemptive
  2,3 = preemptive = แย่งใช้ CPU ได้จาก Priority
* Dispatcher
  - CPU control
  - Context switching
  - user mode ↔ kernel mode
  - dispatch latency = from stop process A to start process B

Scheduling Criteria

CPU utilization	(max)	% busy
Throughput	(max)	# process per time unit
Turnaround Time	(min)	start process A → end process A
Waiting time	(min)	wait in ready queue
Response Time	(min)	start process A → first response from process A

Scheduling Algorithm

1) First Come, First Served (FCFS)
   - มองเข้ามาก่อนอยู่ใน ready queue
   - อาจเกิด convoy effect (short process behind long process)
2) Shortest-Job-First (SJF)
   - เอาจาก history CPU burst-time — suboptimal (approximately)
   - Burst time น้อยสุดทำก่อน
3) Shortest-remaining-time-First
4) Priority Scheduling
   - consider preemptive vs nonpreemtive
   - Problem: starvation — low priority may never execute
     Solution: Aging — increase priority of waiting time
5) Round Robin (RR)
   - time quantum (q) (quota) = time out in 10-100 ms — ใช้ในระบบ timesharing
   - concurrently work
   - q large → FIFO
     q small → overhead of context switching
   - q ที่ดี ต้องเหมาะสมกับ burst time (เท่ากัน)
6) Multilevel Queue
   - แยกคิว

Thread Scheduling

- เมื่อ Process ถูกเลือกมาทำใน CPU แล้ว (user) thread ในนั้นจะทำอันไหนก่อน

Multiple-Processor Scheduling

- Each processor has own private queue
- Processor Affinity — Keep thread run on the same processor (cache benefit)
- Load balancing
  - Counteracts the benefit of processor affinity

⑥ Process Synchronization

- Producer and Consumer
- buffer and counter

Race Condition

- มี process ที่ access and manipulate the same data แล้วทำในคำสั่งกัน
- inconsistency

→ Critical section = one process in critical section at a time

```
do {
    entry section
        critical section
    exit section
        reminder section
} while (true);
```

Solution

1. Mutual exclusion – wait for current running process (forever)
2. Progress (ความคืบหน้า) – if no one in section, enter the section
3. Bounded waiting – มีขอบเขตในการรอ

OS: preemptive or non-preemtive
- free of race condition in kernel mode

Synchronization Hardware – ideas of locking

- Atomic hardware instruction = non-interruptible
  ex. test memory word and set value, swap contents of two memory words
- Uniprocessors Could disable interrupt, running code without preemption

Mutex Locks

acquire() : ขอเข้า critical section และจะไม่ return จนกว่าจะได้ lock
no return แล้ว ก็ทำงานที่ใน critical section

release() : คืน lock

- busy waiting → ระหว่างรอ ต้องมาคอยวน check ว่า lock ว่างยัง → spin lock
  กินทรัพยากร

Semaphore

- ใช้ integer นับ มีได้ 2 operations → wait() and signal()
  ↳ check if S<=0 and S--;
  ↳ S++; ∨release
- Counting Semaphore
  Binary Semaphore → mutex lock

Implementation

block – เอาของไปต่อใน waiting queue
wakeup – ย้ายของจาก waiting queue ไป ready queue

Deadlock and Starvation

Deadlock – ต่างคนต่างรอกันและกันตลอดไป
Starvation – ต้องรออยู่ใน waiting queue นานมาก
Priority Inversion – ของที่ priority ต่ำไม่ยอมปล่อย Lock ที่ของที่ priority สูงกว่า
→ แก้ด้วย Priority-inheritance protocol : แก้ priority ที่สูงขึ้นชั่วคราว จะได้รับทำงานรับของกลับไป

Classical Problem

- Bounded-buffer problem
- Readers and Writers Problem – ยอมให้มี reader หลายตัวได้ แต่ writer ควรมีตัวเดียว
- Dining-Philosophers Problem

  Solution: อย่ามีคนใช้เกินกว่าที่กับทรัพยากร

  ถ้าจะหยิบตะเกียบก็หยิบ 2 อันพร้อมกัน ถ้าหยิบไม่ได้ก็วางยังไงอันเดียว

  หรือมีคนหยิบซ้ายบางขวาบ้าง (asymmetric)

Problems with Semaphore

- ปล่อยก่อนขอ
- ขอแล้วไม่ปล่อย ปล่อยโดยไม่ขอ } leads to deadlock and starvation
- ขอแล้วขออันเดิมซ้ำ

MIDTERM-END