CS2106 Notes. Part I.

Date No.

Lec 1: Types of OS.

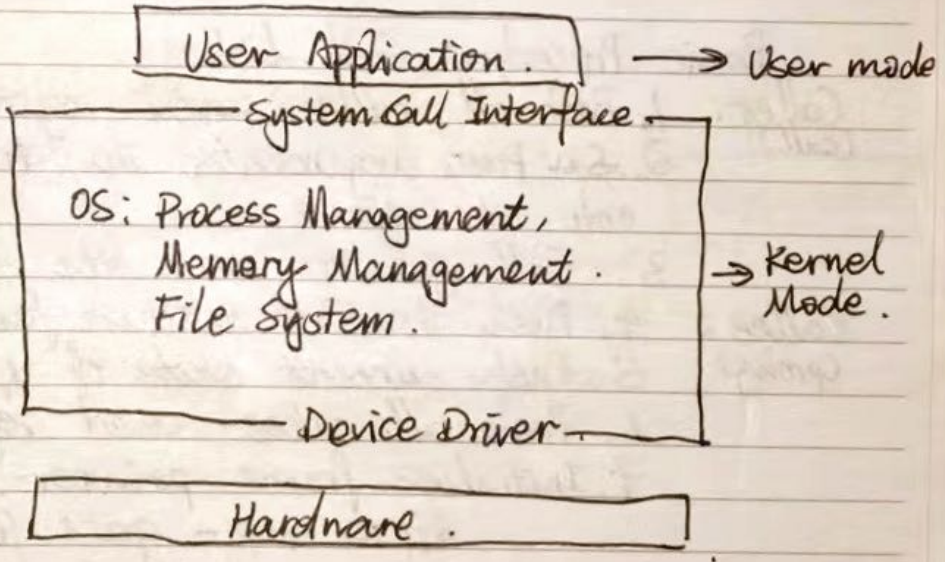
- Embedded OS: For specific devices.
- Real-Time OS: Soft real-time vs Hard real-time.
- Distributed OS: Loosely - vs Tightly coupled.
- Modern PC OS: Multitasking, multi-user, variety of hardware

Overview of Architecture.

User Mode { User applications: Dota, Firefox etc } Software
{ User inteface: GUI, CLI etc. }
Kernel Mode { Operating System: system calls. } Hardware.
Hardware. } Hardware.

Types of Kernel: (OS structures).

Monolithic: A stand-alone kernel that provides a large, integrated user interface.



Microkernel: Very small and clean, and only provides some essential and fundemental services

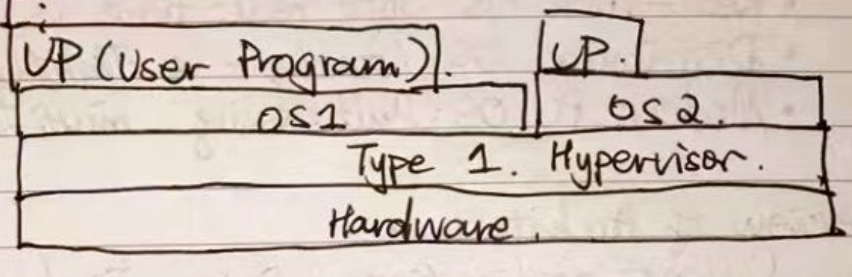
Layered Systems: Turn monolithic structures into layered hierarchies.

Client-Server Model: Variation of microkernel. Client processes make use of server processes.

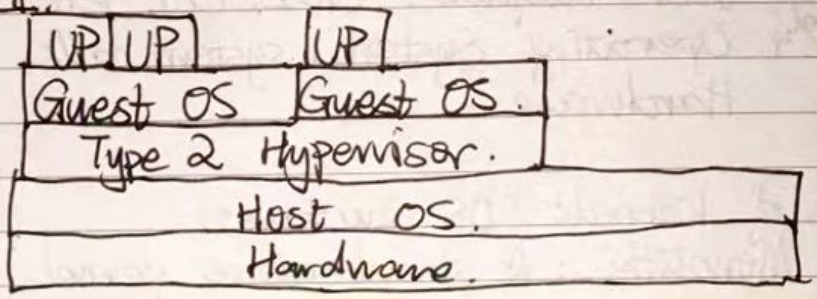
Date No.

Hyperviser: The monitor/integrater that coordinates different OSes running on the same hardware.

Type I:



Type II:



Basic Procedure Linkage.

Caller: (call)
1. Push all caller saved registers to the stack.
2. ~~Sav~~ Pass arguments in $a0 to $a3. Push extras onto the stack.
3. Call jal to save the return address.

Callee: (prolog)
4. Push $ra. (if another function call needed).
5. Push current state of $fp.
6. Push all callee-saved registers.
7. Initialise frame pointer:
   $fp := $sp - space_for_local_variables.
   At this stage, all local variables are also initialised.
8. $sp := $fp.

(Body).
9. Use any registers (Local var are referred as disp($fp))
10. * Recursively call other functions.

(Epilog).
11. Put return values into $v0 - $v1. $sp := $fp - space.
12. Pop callee saved registers in the correct order.
13. Restore $ra (if pushed).
14. 13.a Restore $fp.

Date No.

14. Perform jr $ra.

Caller (Regaining Control): 15. Pop all caller saved registers back.

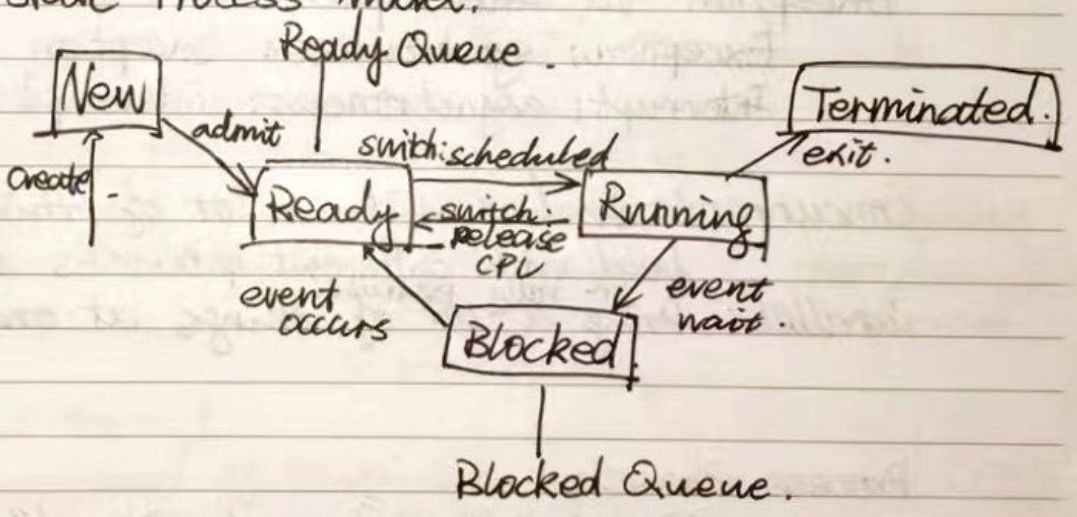
Process.

Process Abstraction
Process Scheduling
Inter-process communication.
Light-weight Process - Thread.

Contexts of a Process:

- Memory Context: Code (.text), Data, Heap, Stack.
- Hardware Context: Registers, PC, Stack Pointer, Stack Frame Pointer.
- OS Context: Process properties (PID etc), Resources Used.

Generic 5-State Process Model:



Orphan Process: A process' parent terminates without waiting for it. It is running.

Zombie Process: A process' parent terminates without waiting for it. It terminates but does not get cleaned up. It remains in the process table.

Date No.

Process Table:
Each entry in the table is a Process Control Block (PCB). A PCB has all info about the 3 contexts of a process.

General System Call Mechanism:
1. User program invokes the library call.
2. Library call places the system call number in a designated location. E.g. Register.
3. Library call executes a special instruction to switch from user to kernel mode. "TRAP"
4. In kernel mode, the appropriate system call handler is determined. "Dispatcher"
5. Execution
6. End of sys call.
7. Library call returns to the user program.

Exception VS. Interrupt.
Exception: synchronous: exception handler
Interrupt: asynchronous: interrupt handler.

Concurrent: Dealing with a lot of things at once ~~parallel~~
Deal with different processes as if they are ~~par~~ parallel.
Parallel: Doing a lot of things at once. or really

Process Phase:
CPU Activity: "Compute-Bound"
IO-Activity: "IO-Bound"

Criteria for Scheduling Algorithms.
Fairness: fair allocation, no starvation.
Utilization: all resources/time should be utilised.

Scheduling Policies:
- Non-preemptive.
- Preemptive.

Processing Environments:
- Batch Processing: No user interaction, non-preemptive scheduling is dominant.
  - First-Come-First-Served (FCFS).
    No starvation. FIFO queue. Problem: longer avg waiting time, Convoy Effect (idling).
  - Shortest Job First (SJF).
    Possible starvation. Guarantees shortest avg waiting time. Need to know process time in advance.
    $Predicted_{n+1} = \alpha Actual_n + (1-\alpha) Predicted_n$.
  - Shortest Remaining Time (SRT).
    A preemptive version of SJF. New incoming tasks may preempt current running task.

  Criteria for Batch Processing:
  - Turnaround Time: Finish − Arrival
  - Throughput: Rate of completion.
  - Makespan: Total time for all jobs.
  - CPU Utilization: % CPU time on task.
- Interactive Environments

  Criteria for Interactive Environments.
  - Response Time: 1st Response by CPU − Arrival.
  - Predictability: Less variation in response time from prediction → more predictability.

  Timer Interrupt.
  Interval of Timer Interrupt: 1ms ~ 10ms (ITI)
  Time Quantum: Multiple of ITI. 1ms - 10ms.
  - Round Robin (RR)
    Given n tasks and time quantum q, response time is bounded by $(n-1)q$.
  - Priority Scheduling.
    Could be preemptive or non-preemptive.
    Possible starvation
    Priority Inversion.

Date No.

- Multi-Level Feedback Queue.
  New job is assigned the highest priority.
  ① Fully utilized TQ : priority decremented.
  ② Gives up / Blocked : priority remained.
- Lottery Scheduling.
  Lottery ticket is randomly chosen. Winner is granted the access. Good control of proportion / likelihood of winning. Mocks priority. Responsive. Distinguishes different resources.

Inter-Process Communication.
- Shared Memory. } general.
- Message Passing. } Not common.
- Semaphores } Unix specific
- Pipe
- Signal

Shared Memory
Create → Attach → Read/Write → Detach → Destroy
Master ↔ Slave.
Pros: Less OS concern + ease of use.
Cons: Synchronization issue + implementation

Message Passing.
Makes use of system calls.
Issues: Naming and synchronization.
- Naming Scheme.
  - Direct Communication:
    Unix Implementation: domain socket.
  - Indirect Communication.
    "Mailbox" / "port"
    Unix Implementation: message queue.

Date No.

- Synchronization Behaviors.
  - Blocking Primitive: Recipient blocked.
  - Non-Blocking Primitive: Recipient can carry on.

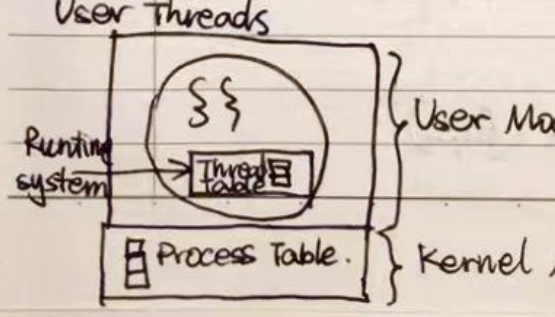
Unix Default Communication Channels: stdin, stdout, stderr

Unix Pipes:
- Circular bounded byte buffer with implicit synchronization.
- Half-duplex / Unidirectional: one end for W and the other for R.
- Full-duplex / Bidirectional: any end W and R.

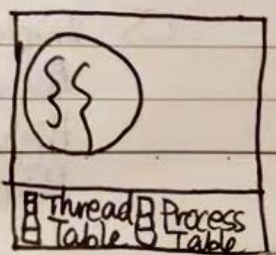
Unix Signal:
- An asynchronous notification regarding an event sent to a process/thread.
  E.g. kill, Interrupt, stop, Continue

Threads:
- Shared Info:
  Memory Context: Text, Data, Heap.
  OS Context: pid, files etc.
- Unique Info.
  Memory Context: Stack.
  OS Context: thread id (identification).
  Hardware Context: Registers, especially PC.

Types of Threads:
- User Threads = kernel is not aware of them
- Kernel Threads: Kernel can schedule for them as if they are processes.



Kernel Threads:



* Thread Benefits:
  - Economy: less resource for context switching.
  - Resource sharing;
  - Responsiveness
  - Scalability: takes advantage of multi-cores.

Date No.

Hybrid Thread Model.
Binds user threads to kernel threads.
Offers great flexibility.

Simultaneous Multi-Threading. (SMT).
Supplies multiple sets of registers for several threads to run in parallel on the same core.

Race Conditions.
Execution outcome depends on the order in which the shared resource is accessed/modified.

Critical Section (CS):
At any point of time, only one process can execute in the CS.

Properties of Correct CS Implementation.
- Mutual Exclusion
- Progress: No live locks. All processes eventually get what they need.
- Bounded Wait: To not make other processes starve.
- Independence: Irrelevant processes can skip the CS. Processes not executing in CS do not block others.

Symptoms of Incorrect Synchronization
Deadlock: No progress
Livelock: Processes keep changing states to avoid deadlock but make no progress.

CS Implementations:
Assembly Level Implementations (ALI)
High Level Language (HLL)
High Level Abstraction.

Date No.

Test-and-Set

Atomic Instruction → Either done or fail. No intermediate states.

Busy waiting is not recommended: ① Waste of system resources & does not really listen to changes all the time.

Possible Causes of Improper Implementation of Synchronization:

① Busy waiting (spin lock)
② Non-atomic instructions make it possible for processes to be preempted halfway.
③ Disabled interrupt → unable to recover
④ Deadlock → not atomic enough.

Peterson's Algorithm:

Make use of turn:
Key idea: Even if both processes want the same resource, turn is either 0 or 1, so one process can proceed. (guaranteed by atomic write op to turn
Cons: Busy waiting, low level, not general (only 2 p).

Tips for Spin Lock (Busy Waiting)

Long wait expected → pay the overhead and block
Low competition → do busy waiting.

Semaphore

S ← an integer, initially non-negative.

- Wait (S): // or down() or P()
  If S≤0, blocks
  S--
- Signal (S): // or Up() or V().
  S++
  Wakes up a sleeping process if any. // regardless // of value // of S.

Date No.

Invariant:

$S_{current} = S_{initial} + \#Signal - \#wait$.

General Semaphore (Counting Semaphore): $S \geq 0$

Binary Semaphore: $S \in \{0, 1\}$.

Incorrect use of semaphore: two semaphores can lead to deadlocks.

Conditional Variable:

Allows a task to wait for a certain event to happen. Can broadcast — wakes up all waiting tasks related to monitor.

Classic Synchronization Problems.

Producer & Consumer

Reader & Writer

Dining Philosophers.

Producer & Consumer.

- Busy Waiting: count is in CS → only one is working at any point of time.
- Blocking Version: Uses 3 semaphores: one to protect count, the other 2 to notify the partner. → still, only one is working at any point of time.

Readers & Writers:

- Simple Version: Uses 2 semaphores: one to protect the access to the buffer, the other to tell there is room for writing.

Date No.

Dining Philosophers.

- Possible Deadlock: all philosophers wait for the RHS.
- Possible Livelock: all philosophers try to offer chopsticks to others upon mutex
- Low utilization: only one can eat at any point of time if a naive mutex is used.
- Tanenbaum Solution:

  Mutex to protect access to state[N].

  An array of semaphores to block respective philosopher upon failure to secure both chopsticks → signal before wait, → signal succeeds if both chopsticks are at hand, so wait() succeeds immediately. Otherwise, blocked by wait() → Left/Right philosopher signals it by calling putChpStcks().
- Limited Eater:

  Intensionally leaves one seat empty → deadlocks are impossible.

  Use a semaphore seats (initially 4) to control the seats, use an array of semaphores to control chopsticks.

CS2106 Notes Part II

Date No.

Memory Usage
- Transient Data (temp variables, local variables)
- Persistent Data (dynamically allocated ...)

Manage Memory:
- Allocate
- Manage
- Protect
- System calls.
- Internal Use.

Memory Abstraction → Base + Limit Registers.

Base: the starting addr of a process

Limit: the boundary beyond which a process cannot access. Also an absolute addr.

∴ All memory references are compiled as an offset.

Handling Full Memory.
- Remove terminated processes
- Move some memory to a secondary storage.

Memory Partitioning.
- Fixed-Size Partition.
  The partitions do not change dynamically.
- Variable-Size Partition.
  Splitting and merging required.

Fixed-Size Partition.
Pros: Easy to allocate. No need to choose.
Cons: Internal fragmentation. Limited size for large processes.

Date No.

Dynamic Partition / Variable-size Partition.

Pros: Flexible and no internal fragmentation.

Cons: External fragmentation.

Slower to choose.

High info maintained.

Allocation Algorithms.

- First-Fit

  The lowest runtime.

- Best-Fit

  If all processes come together, the waste is low. -est

- Worst-Fit.

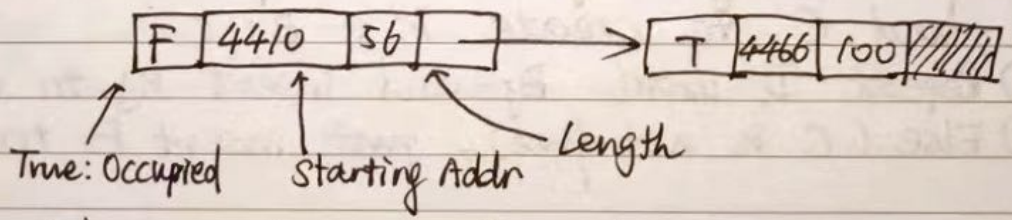
After a hole is found by one of these algo, split this hole to a just-fit partion and a new hole.

Merging & Compaction:

- Merging: Merge adjacent holes.
- Compaction: Move partions around to create a consolidated hole.

  Very time consuming.

Linked Lists for Partion Info.

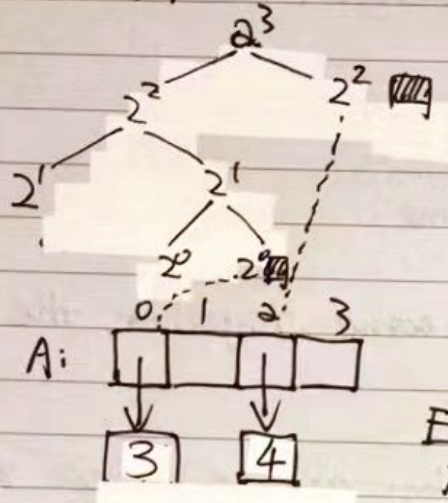


Thus, context switching does not overturn the mem.

Date No.

Buddy System.

Keep an array A[0..k] for a memory space of size 2^K and each element A[i] is a linked list that keeps track of free blocks of size 2^i.



Each free block is indicated by its starting addr only.

Allocation for N

① Find the smallest S such that $2^S \geq N$.
② If A[S] is not empty, allocate it.
③ Else for S+1 to K, find the smallest R s.t. A[R] is not empty.
④ Repeatedly split A[R] to A[S], allocate it.
⑤ Now A[S] to A[R-1] ~~bar~~ each has exactly one free block.

Deallocation of a block B.

① If a buddy C of B is also free, remove B and C to create $B' \leftarrow B \cup C$.
② Repeat ① until Bj. and insert Bj to the list.
③ Else (C is not free), just insert B to the list.

Identification of a buddy block.

B and C are buddies of size 2^s iff for their addr:
① The rightmost s-th (0-based) bits are ~~comp~~ complement.

② The leading bits up to the s-th bit are all the same.
③ The trailing bits must be 0.

Paging

Physical Memory: physical frames, size fixed by the hardware.
Logical Memory: logical pages, from the perspective of a process.
∴ Logical memory remain contiguous, but physical frames may be disjoint.

Physical Address = Frame # × Frame Size + Offset
= fmap (Page #) × Frame Size + Offset.

Design Decisions:

① Frame size = Page size.
② Size = 2^k for some $k \in \mathbb{N}$. (usually 4KB).
∴ Logical: Page # → [000...0][00000...0]
m k
↓ ↓
Physical: Frame # → [101...1][00000...0]
m k

Page Tables.

In RAM kernel space, part of PCB.

Paging – Comments.

Removes external fragmentation
Internal fragmentation persists.
Pros: Flexibility & Easy Translation.
Cons: Two mem accesses: 1. Look up page table
2. Locate the actual item.

Fact: All processes share the kernel space, but each process holds a private user space

Fact: Upon resumption of a process, there can be many TLB misses.

Date No.

Translation Look-Aside Buffer.

- A hardware support for paging.
- Acts as a cache.
- If TLB-Hit, there is no need to access the page table (which is in mem)
- Part of the hardware context, flushed every context switch

Calculation of Mem Access Time.

① Suppose TLB is not updated, page table is not updated:

$Time = p(t+m) + (1-p)(t+2m)$.

p: hit rate.

t: TLB access time.

m: mem access time.

Page Protection.

- Access Right Bit.
  Readable, writable, executable etc.
- Valid Bit.
  Out-of-range pages are not valid.

Page Sharing.

- Diff pages in diff page tables point to the same frame.
- Shared Code Page.
  C Lib, system calls. etc.
- Copy-on-write.
  Both parent and child cannot write before doing a deep copy.

Regions of a typical C programs:
Library, stack, heap, user code, global variables

Date No.

Segmentation

- Separat regions into multiple segments. (Can be diff)
- Now each segment has a name and a limit.
- ~~Physical~~ Memory reference = name + offset.
- May be disjoint in physical mem
- Each segment is mapped to a contiguous physical space
- Logical addr = <SegID, offset>

Look up <Base, Limit>. offset ≤ Limit.
↳ A relative limit.

Pros:
- Independent contiguous phy mem
  ⇒ Grow/Shrink independently.
  ⇒ Protected/shared independently.

Cons
- Requires variable-size contiguous space.
- External fragmentation.

Paging VS. Segmentations

- Paging solves the problem of disjoint physical mem
- Segmentation solves the problem of growing/shrinking
- ∴ In combination, each segment has multiple pages, not a contiguous region.

Each segment is essentially a page table.

Given that max seg size = 4.

Segment	Page Limit	Page Table
text	2	
data	2	

Page #	Frame	Valid
0	7	T
1	4	T
2	—	F
3	—	F

Page #	Frame	Valid
0	6	T
1	—	F
2	—	F
3		

Warning: in general, max size of a seg ≠ page limit of a seg.

Date No.

Virtual Memory

Use a "Present" bit in the page table to distinguish.

- Memory Resident
  Pages in physical mem
- Non Memory Resident → Checked by hardware
  On access, a page fault occurs, and the OS (Trap) loads it from the secondary storage to the physical mem, Then, the OS updates the page table

Thrashing:

Too many page faults. The process keeps accessing non-memory residents.

Locality Principles.

- Temporal Locality:
  Mem addr that is referenced is likely to be referenced again.
- Spatial Locality.
  Mem addr that is closed to a referenced addr is also likely to be referenced.

Virtual Mem Summary

- Logical addr space not limited.
- More efficient use of physical mem (store the needed only)
- Better CPU utilisation as there are more processes to choose to run.

Page Table Structure.

- Direct Paging.
- 2-Level Paging

Size of a Page Table Entry is usually 2 bytes.

Date No.

Direct Paging.

- Too large logical space has high overhead. A page table can occupy multiple pages ⇒ posible self-reference ⇒ unreachable page.

2-Level Paging.

- Motivation: Mapping the entire page table is a waste.
- Split full page tables into regions.
- Note: Only a few regions are used. Warning: 2-level paging also have to map all possible virtual addresses, but some smaller tables might not be in mem
- One sub-page table also occupies a whole page (when they are full)

Directory ⇒ Page Tables → Pages

Ignore frame numbers:

Suppose the logical address space is 2^P.
Smaller page tables := 2^M
Size of master directory = $M \cdot 2^M$
Page size := $M \cdot 2^M$
Size of entries in smaller tables = $\frac{M \cdot 2^M}{2^{P-M}} = M \cdot 2^{2M-P}$.

Total overhead = directory table + all smaller tables

Inverted Table.

- Keeps a single mapping of physical frame to <pid, page#>. Page # are not unique among processes. <pid, page#> uniquely identifies a page.

- Pros: Saving space: one table for all processes. Faster allocation/de allocation
- Cons: Slower translation.

clean page VS dirty page.

Date No.

Page Replacement Algorithms.

- Optimum (OPT) / Optimal Page Replacement.
  Ideal. A base for comparison
  The closer to OPT, the better an algo.
  Replace the page that will not be used again for the longest period of time.

- FIFO.
  OS maintains a queue of pages.
  Evict the oldest page.
  No hardware support needed.
  If A is already in the queue, accessing it will not change its place.
  Belady's Anomaly → FIFO does not exploit temporal locality.

- Least Recently Used (LRU)
  Replace the page that has not been used for the longest time.
  No Belady's Anomaly.

- Second-Chance Page Replacement (CLOCK).
  Each PTE has a reference bit.
  1 = Accessed 2 = Not accessed.
  The oldest FIFO page is selected.
  Ref. bit = 0 ⇒ replace
  Ref. bit = 1 ⇒ clear to 0, give another chance, go to the next page.
  → Load time reset as if newly admitted
  Degenerate into FIFO if all ref. bit = 0 or all 1
  If a page is new, the ref. bit is initialised to 0. If it is accessed, then it becomes 1

Date No.

Frame Allocation

:N frames and M processes.

- Equal Allocation: $\frac{N}{M}$.
- Proportional Allocation: $\frac{size_p}{size_{total}} \cdot N$.

Local Replacement

- Pros; #Frames for a process is constant → stable performance for multiple runs.
- Cons: Hinders performance if not enough pages allocated.

Global Replacement:

- Pros: Allows dynamic self-adjustment
- Cons: Badly behaved processes can affect others
  #Frames is diff from run to run

Frame Allocation & Thrashing

- Heavy I/O to bring many pages to RAM.
- For global replacement:
  Cascading Thrashing: a process stealing frames causes others to thrashing.
- For local replacement:
  Thrashing can be limited to one process.
  Single process can hog on I/O (if no enough frames allocated) and downgrade others' performance.

Working Set Model

A combination of both localities.

Δ := Working set window / an interval of time (fixed)

$W(t, \Delta)$ = active pages in the interval at time t.