**Ch 10 virtual memory**

Demand paging:

* Less i/o needed, no unnecessary i/o
* Less memory needed
* Faster response
* More users

Lazy swapper – never swaps a page into memory unless the page will be needed

Swapper deals with pages is a pager.

Load a invalid page => page fault

Effective access time (EAT) = (1-P)\*memory access + p (page fault overhead + swap out + swap in)

FIFO

More page => Belady’s Anomaly

Optimal: will not be used for longest period

Can’t read the future

LRU: has not been used in the most amount of time

LFU: replace with smallest count

<https://hackmd.io/@Pl-eQT9CQaS0jhExKqL8_w/BkhOSR4jW/https%3A%2F%2Fhackmd.io%2Fs%2FBkosqrX1f?type=book>

Thrashing: process not having enough frame -> high paging activity, paging time is higher than processing time

惡性循環

* Low CPU utilization
* Os increase degree of multiprogramming
* New process take old process frame
* More page fault

Solution: **work-set-model, page-fault frequency**

WSM: os moderate D(total process working-set size), m = available frames

1. D > m :  中止process —> free掉所有他佔用的frame
2. D << m —> 提升degree of multiprogramming

pros: prevent thrashing, increase cpu usage

cons: tracking cost high, need to update working-set, lower efficiency

PFF: 直接測量與控制page fault rate以避免thrashing

* 為每個process建立upper bound與lower bound在一個理想的page fault rate
* 若page fault超過upper bound —>
  + 配額外的frame給這個process
  + 若沒有free frame，立即暫停process，free掉他占用的frame
* 若page fault低於lower bound —>
  + 從該process移除frame（表不需要這麼多frame）
  + launch新的process

Buddy system allocator

Pros: quickly coalesce unused chunks into larger chunk

Cons: fragmentation

Slab allocator:

Pros: no fragmentation, fast memory request satisfaction

**Ch11: mass storage system**

* I/O bus被controller控制
  + Host controller（電腦端）
  + Disk controller（內建到disk drive）

Disk scheduling :

FCFS

SCAN: upward or downward, no starvation

C-SCAN: more uniform waiting time than SCAN

SSTF(shortest-seek-time-first)

Cyclic redundancy check(CRC): uses a hash function to detect multiple-bit errors

Error-correction code(ECC): not only detect, but also correct errors

Low-level formatting

**CH12 I/O system:**

Interact with device:

* Polling: busy-waiting, check device status register regularly
* Interrupt: device notice processor i/o finish

Direct memory access(DMA): used for large data movement,

* Need DMA controller
* Bypasses CPU to transfer data
* OS writes DMA command block into memory: 通常搭配使用memory-mapped I/O和interrupt I/O method

I/O device class: Device class is fairly standard across different OS

* Block I/O
  + Read, write, seek
  + Raw I/O, direct I/O, or file-system access
  + Memory-mapped file access possible
  + DMA
  + Interrupt I/O
* Char-stream I/O
  + Get, put
  + Libraries layered on top allow line editing
  + Pulling I/O
* Memory-mapped file access
* Network sockets
* Clock & timer interfaces
  + Programmable interval timer used for timings, periodic interrupts

Nonblocking & asynchronous I/O

* Blocking
  + Easy to use and understand
  + Insufficient for some needs
  + Use for synchronous communication & I/O
* Nonblocking
  + Implemented vis multi-threading
  + Returns quickly with count of bytes read or written
  + Use for asychronous

Vector I/O: allows one system call to perform multiple I/O operations

* Decrease context switching and system call overhead
* Some versions provide atomicity

Kernel I/O subsystem

* Scheduling
  + Ordering via per-device queue
  + Some OSes try fairness
  + Some implement Quality Of Service (IPQOS)
* Buffering: store data in memory while transferring between devices
  + 當要在I/O devices之間做transfer，將data存到memory（e.g. disk to network card）
  + 調節I/O devices之間的速度
  + 原因
    - devices之間的速度不一樣
    - Devices with different data-transfer sizes
  + Support copy semantics
    - 要做多次的data copy
* Caching: fast memory that holds copies of data
  + Always just a copy
  + Key to performance
  + Sometimes combine with buffering
* Spooling
  + 先load到spooling的空間去，再一次送到printer
  + e.g. printing (cannot accept interleaved files)
* Device reservation
  + System calls for allocation and de-allocation
  + Watch out for deadlock
* Error handling
  + OS can recover from disk read, device unavailable, transient write failures
  + Return an error number or code when I/O request fails
  + System error logs hold problem reports
* I/O protection
  + All I/O instructions defined to be privileged
  + I/O must be performed via system calls
* Power management

Performance

* Device driver code
* Context switches
* Data copy
* Interrupt handling
* Network traffic

How to improve?

* Reduce number of context switches
* Reduce data copy
* Reduce interrupts by using large transfers, smart controllers, polling
* Use DMA
* Use smarter hardware devices
* Balance CPU, memory, bus, and I/O performance for highest throughput
* Move use-mode processes / daemons to kernel threads

**CH13 File-system interface**

**CH14 File-system implementation**

Boot control block: contain info needed by system to boot OS from that volume

Volume control block: contain volume detail

File control block: contains many details about the file

In memory structure:

* in-memory partition table
  + 每個被掛載的partition的資訊
* in-memory directory structure
  + 最近被存取的directories的資訊
* system-wide open-file table
  + opened file’s FCB copy
* per-process open-file table
  + pointer（file handler/descriptor）指到system-wide open-file table對應的entry

Directory implementation:

* Linear list
  + Time consuming
  + Linear search time
* Hash table
  + O(2)
  + Entries are fixed sized

Contiguous allocation: each file occupies set of contiguous blocks

* Simple
* Best performance in most cases
* Problems:
  + External fragmentation: solution-> compaction
  + Knowing file size

Extent-based systems

* 這個方法用extent分配disk blocks
* 許多新的FS使用modified contiguous allocation scheme
* 如果用完了給他的extent，再給他另一個extent，用linked list串起來
* 若跳開，所以需要讀前一個extent的最尾的extent block中的extent pointer（放pointer的位置由designer決定）
  + 加總要access 2次
* 缺點
  + 隨機存取costly
  + 可能會發生internal & external fragmentation

Linked allocation: each file a linked list of blocks

* 每個file是個blocks的linked list
  + 每個block包含一個pointer指到下一個block
    - data部份：block size – pointer size
* 建立新檔案時並不需同時就宣告檔案大小
* 問題
  + 只有對於sequential-access files有好處
    - 隨機存取需要traversing整個link list
    - 每次的存取link list都是一次的disk I/O
      * 因為link pointer被存在data block
  + pointer所需要的空間（4 / 512 = 0.78%），空間存指標，而不是用於存放資料
    - 解法 - unit = cluster of blocks  
      –> internal fragmentation
  + Reliability
    - 只要遺失一個link就會損毀者個file

File-allocation table (FAT)

Indexed allocation: each file has its own index block of pointers to its data blocks

Pros:

* Direct and random access are effective
* No external fragmentation
* No allocation problem

Cons:

* Index blocks need space
* When files are too big index block are not enough how big should a index block be?

Free space management

Free space list: track available blocks/clusters

* Linked list
  + 所有的free blocks link在一起（same as linked allocation）
  + 讓第一個free block pointer在disk和快取中的特別位置
  + Traversing list會是沒有效率的
  + 沒有需要traversing; 所有的link-pointers放在table（FAT）
* Grouping
  + 與linked-index allocation一樣
  + 將n free blocks的address放在1st block
  + 前（n-1） pointers是free blocks
  + 最後一個pointers是另一個grouping block
* Counting
  + 與[contiguous allocation](https://hackmd.io/s/rkZE0cv7f" \l "contiguous-allocation" \t "_blank)一樣
  + keep first free block的address和contiguous free blocks的數量
* Bit vector
  + East, effective
  + Hardware support
  + Need cache

**CH15 File system internals**

**CH18 virtual machine**

Host: underlying hardware components

Virtual machine manager

Guest

Type 0 hypervisors: hardware based

Type 1 hypervisors: OS-like

Type 1 hypervisors: general purpose OS

Type 2 hypervisors: applications

* Can’t take advantage of some HW features
* Also benefit because require no changes to host OS

Implementation:

Paravirtualization

Programming-environmental virtualization

Emulators

Benefits & features

VMs protected from each other

Freeze or suspend a running VM

Great for OS research better system development efficiency

Run multiple, different OSes on a single machine

Templating

Live migration

Cloud computing

Binary translation

Nested page table: VMM maintain per guest NPTs to represent guest’s page-table state

Emulation: run one OS on a different OS 遊戲機