[redacted] self study

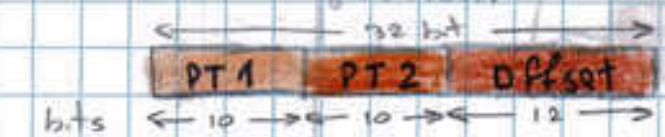
Pages Tables for large Memories

- Instead of using page-table-in-memory, we can use TLB to speed up virtual address to physical address translation

What alternative to page-table-in memory we have to speed up a virtual address to physical address translation

Multilevel Pages Table

- we have a 32-bit virtual Address that is partitioned into: a 10-bit PT1 field
  a 10-bit PT2 field
  and 12-bit offset field
  → since offset is 12-bit, pages are 4KB then there are 2^{20} pages



What is the virtual address of a multi-level page table

- Avoid keeping all the pages tables in memory all the time, in particular those that are not needed should not be kept around

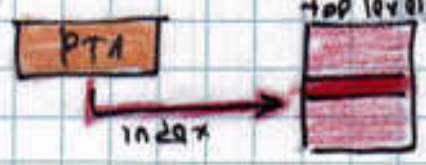
What should we avoid with pages table?

- In a two-level page works in this way

How does a two-level page works

A. on the left we have the top-level page table with 1024 entries, corresponding to the 10-bit PT1 field

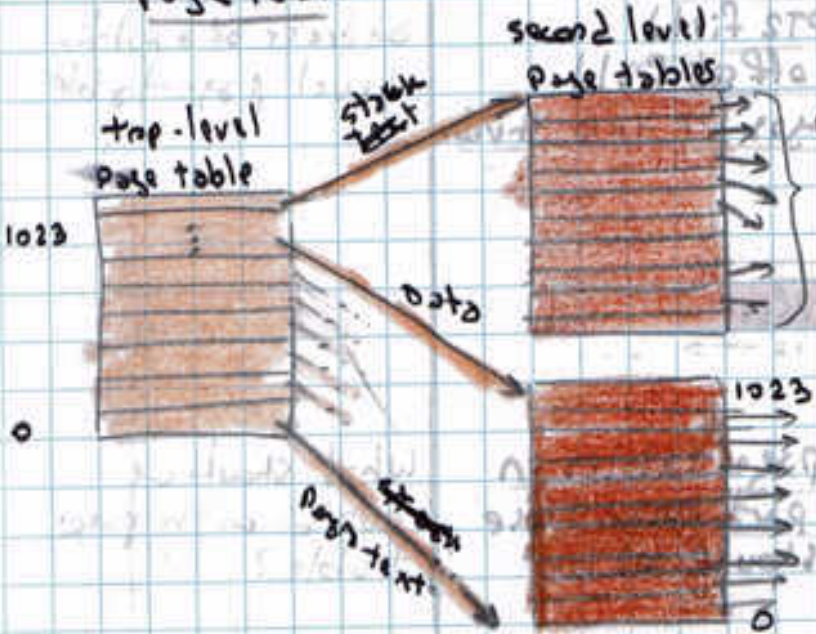
- when a virtual address is presented to the MMU
  ① Extract it the PT1 field and uses this value as an index into the top level page table



Multilevel Page Table (Continue)

- Each of these 1024 entries represent 4MB because the entired 4GB virtual address space has been chopped into chunks of 4096 bytes.

B. The top entry located by indexing into top level page table yields the address or the page frame number of a second-level page table



- Entry 0 of the top-level page table points to the page table for the program text.
- Entry 1 points to the page table for the data
- Entry 1024 points to the page table for the stack
- The PT2 field is now used as an index into the selected second-level page table to find the page frame number for the page itself.

[redacted] self study

- Additional level give more flexibility, but it is doubtful that additional complexity is worth it beyond three levels.

Ex: ① 32 bit virtual address: 0x00403004
(4,206,596 decimal)
(12,292 bytes into the data)

② virtual address correspond to
- PT1: 1
- PT2: 2
- Offset: 4

- MMU uses PT1 to index into top-level table and obtain entry 1. which correspond to address 4M to 8M

- Then MMU uses PT2 to index into the second level page table just found and extract entry 3 which correspond to addresses 12288 to 16383 within 4M chunk
→ this entry contains the page frame number of the page containing virtual address 0x00403004

  - if page is not in memory, the present/absent Bit in the page entry will be zero, causing page fault

  - if page is in memory, the page frame number taken from the second-level page table is combined with the offset (4) to construct the physical address.
    ⇓
    this address is put on the bus and sent to memory.

~~Inverted Page Tables~~

Multilevel Pages table (continue)

- 32 bit addresses with 2 page table fields
- two-level page tables
- PT to big for MMU
  → Place it in main memory
- How does MMU know where to find PT?
  → Registers (CR2 on Intel)

Inverted Page table

[redacted] self study

Impact of Page Size on Page table

Small page size:

Advantages:
- less internal fragmentation
- Better fit for various data structures, code, sections

(if small page & small data structure, it would fit well)

Disadvantages:
- Program needs more pages and has larger page table.

Notes:

Bigger

~~Concurrency versus Parallelism~~

Concurrency:

OS Involvement with Page table Management

Four times OS deals with with page-tables

1. Process Creation: Create a page table

2. Upon Context switch:

   A. Load MMU context for a new process (ie. load CR3 register w/ base address of page table)

   B. TLB is flushed

3. Page fault time:

   A. Determine the virtual address causing fault (Read CR2 register for faulting address)

   B. swap target page out, bring needed page in.

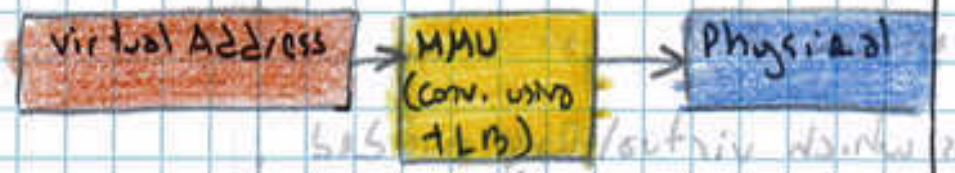
4. Process termination time: release page table and other pages

- the goal is that everytime we bring a page to memory, we have to update the page table
- the table is a mapping between virtual Page and the physical page.

[redacted] self study

- let say in ya program you have the following code: x=x

you need to access memory of x and the memory access of y so



- Page fault is when accessing virtual page address, there is not a page address.
This is not exactly an error and it is handled by the MMU showing the virtual page address, informs the CPU the virtual address producing the viol
→ Page fault is an exception viol, when the CPU tells the OS. Then the OS will handle the CPU and it will bring the page back.

- when a virtual Address page is not located, it may be located in the secondary device = Page fault
In second case: support you try to execute or write code from a page that you don't have permission it will produce a page fault

ex Lets say you do a malloc for a for a 100 MB of space. the OS will waits until the program try to access the memory address. Then the OS will incrementaly add pages.
the first time trying to access it will produce a page fault

Page fault Handling

- When the page fault occurs, the OS has to choose a page to evict (remove) to make room for the incoming page.

1. Hardware (MMU) raises a trap to the OS
2. General registers saved
3. OS determines which virtual page needed
4. OS checks validity of virtual address
5. OS selects victim page; if victim is dirty (victim dirty refers to whenever you write a piece of memory it became dirty) then write to disk
   - this step is normally done out-of-band. ie: before a page-fault occurs, so that page fault can be handled faster
   - Note: if there is a not good algorithm for this step, the machine will become extremely slow.
6. OS schedules to bring new page in from disk

- Dirty bit indicate the pages is dirty.
  - → if the page is dirty then the context of the page have to be saved
  - ~~→ if the page is not dirty~~
  - → if you don't have to write the page down and then you have to bring the other page up. This is overwrite page

Page fault handling (continue)

- Victim Dirty: two ways to free space
  1. Ignore context and overwrite the page bit it can happend that an applic. has some data there, so you have to save it. 2. Data that is not save that you want to save means data here dirty
- Every swap space should have twice the amount of memory ram

7. Pages tables updated to reflect new mappings
8. Faulting instruction backed up to when it began
   (we have saved the execution process but now we are doing it here)
9. Faulting process re-scheduled
10. registers restored
11. Program continues

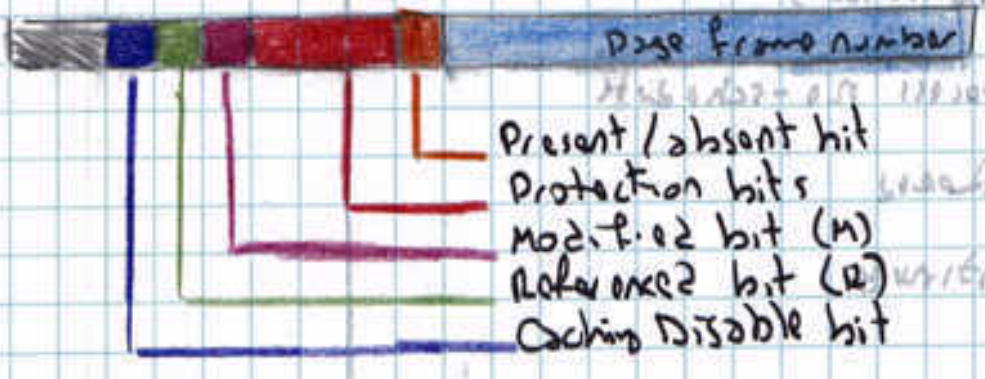
- Any algorithm in this os have to make sure that the # of pages is minized.

Another way to see this:

- When a page fault occurs, the OS chooses a page to evict (remove from memory) in order to make space for the incoming page
  - → If the page to be moved has been modified while in memory, it must be rewritten to the disk to bring the disk copy up to date.
  - → If the page has not been changed (ie. it contains program text), the disk copy is already up to date, so there is no need for rewriting the page to be read in just overwrites the page being evicted.

~~Locking Pages in Memory~~

Not Recently Used Page Replacement Algorithm



- two references are used by the OS:
  1. Reference bit which is set whenever the page is referenced (read or written)
  2. Modified bit which is set whenever is written to (ie. modified)
- These two bits must be updated on every memory reference by hardware.

- the Referenced and Modified bits are used as follows:

  a. when a process is started up, both bits for all pages are set to 0 by the OS

  b. Periodically (by clock interrupt) the Referenced bit is cleared in order to distinguish pages that have not been referenced recently from those who haven't have been referenced

  c. when page faults occurs, the OS inspects all the pages and divides them into 4 categories based on the current values of their Referenced and Modified bits:

	Referenced Bit	Modified Bit
Class 0:	Not Referenced	Not Modified
Class 1:	Not Referenced	Modified
Class 2:	Referenced	Not Modified
Class 3:	Referenced	Modified

  i. class 1 occurs when when a class 3 has its reference bit cleared by a clock interrupt
  ii. Clock interrupt do not clear the modified bit because this information is needed to know whether the page has to be rewritten to disk or not
  iii. Clearing reference bit but not modified bit leads to class 1.

Not Recently Used Page Replacement Algorithm (continued)

- NRU (Not Recently Used) algorithm:
  1. removes a page at random from the lowest-numbered nonempty class.
  2. the idea is that it is better to remove a modified page that has not been referenced in at least one clock tick than a clean page that is in heavy use.
  3. NRU is easy to understand, efficient to implement, and provides optimal performance

Locking Pages in Memory

- Sometimes you don't want to kick out pages for the main memory
- Virtual memory and I/O occasionally interact
- Problem ex.
  a. Process P1 issues call for read (read system call) from device into buffer (Directly to the device without intervention of the CPU)
  b. While waiting for I/O, another process P2 start up
  c. P2 has a page fault
  d. Buffer for the process P1 may be chosen to be page out, resulting in DMA (Direct Memory Access) error

Locking Pages in Memory (continue)

- Need to specify some pages as "locked" or "pinned" in memory
  - These pages are exempted from being victim pages
    (if in the main memory, you don't want to send it from memory)
- If there are too many processes when you start the computer then the computer will be busy and not responding. This means the computer is swapping a lot of pages in and out of the system.

Page Replacement Algorithms

- Page fault forces choice:
  - which page must be removed
  - make room for incoming page
- Modified page must first be saved
  - unmodified just overwritten
- Better not to choose an often used page
  - will probably need to be brought back in soon
- Goal of page replacement algorithm is to minimize the number of page faults incurred by the system

Page Replacement Algorithms

1. Optimal page Replacement (OPR)
2. Not Recently used (NRU)
3. First In First Out (FIFO)
4. Second chance
5. ~~Bloca~~ Least Recently used (LRU)
6. Not Frequently used (NFU)
7. Aging
8. Working set
9. Clock
10. WS Clock

• CPU scheduling Algorithm: to minimize the shortest algorithm first.

1. Optimal Page Replacement Algorith (OPR)

A. Replace page that's needed at the farthest point in the future
   i. "Optimal" means that ~~there are~~ no other algorithm can give fewer page faults than optimal Page Replacement (OPR)
   ii. Optimal page Replacement (OPR) is optimal but impossible
   iii. It can serve as a useful tool of measurement (baseline) to measure the performance of other algorithms.

1. Optimal Page Replacement Algorithm (OPA) continued

B. Why is it optimal?

i. OPT says to remove the page with the highest label.

ii. Pushing page not use farthest away so when there is another page to be pushed away but closer, the ~~lower~~ removes the same pushes the page fault that will fetch soon as far ~~into~~ the future as possible

iii. What if there's another algorithm which does not pick the page needed farthest point in the future and can still yield fewer number of page fault than OPT?

C. Other Algorithms try to approximate OPA

i. Estimate the pages not needed for a long time by recording the sequence of pages used in the past.

2. Not Recently Used Page Replacement (NRU)

A. Each page has a Reference bit and a modified bit which are used to estimate timespan

i. These bits are set when a page is referenced, modified, or clock interruption

~~category~~ Classes	~~reference bit modified bit~~	mod. fied bit	status
Class 0	Not referenced	Not modified	~~leave~~ over wrote
Class 1	Not Referenced	modified	~~unwanted~~ so ~~unused~~
Class 2	referenced	Not modified	Recently access, writing
Class 3	Referenced	modified	used often ~~and~~ dirty either

. How you can have no reference but not modified? After the Direct Memory Access (DMA) is over the page would be dirty but not referenced.

Not Recently Used Page Replacement (NRU) Algorithm (cont: NU)

- What happen when sometime causes a page to go to category 2?
  Periodically this algorithm will scan, select, and then reset all references bits back to zero.
  i. this means that there is some pages that were referenced and some that were moved from category 2 to category 1.

  ii. Category 2 means page is dirty but not referenced

B. NRU periodically scan all memory access.

Same
- i. If there is low memory, then it would pick a page from the lower random set of accessed pages
- ii. If system is under memory pressure, NRU removes page at random for the lowest numbered non-empty class
- iii Reset all referenced bits
  I. This implies that pages moved from 3→1 and from 4→2

C. Before the next NRU scan, some pages may be referenced/modified again
  i. Pages moved between 1→{3,4} and 2→4.

- Maintaining time span requires additional space and process time, so every time this page is accessed or written you have to read the current timespan and update it. this is very expensive to maintain so we try to estimate timespan.

[redacted] self study

Not Recently Used Page Replacement Algorithm (cont'd)

- How do you know a page was recently accessed the last time?
  referenced is used to access modifying is write

3. First-In-First-Out Algorithm (FIFO)

A. Maintain a linked list of all pages
   i. In order they came into memory

B. Page at the beginning of list replaced

C. Disadvantage: Page in memory the longest may be often used

- Suppose you kick out the last the last used page, and an application want to used it. Then some way to fix FIFO so second chance to live.

A. the OS maintain a list of all pages currently in memory, with the most recent arrival at the tail and the least arrival at the head.

B. on a page fault, the page head is removed and the new page added to the tail of the list

- FIFO in its pure form is rarely used.

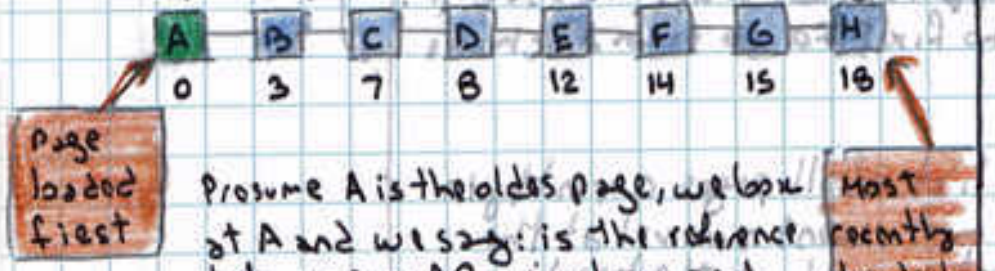
Second Chance Page Replacement Algorithm

- Second chance is a modification to the FIFO algorithm that avoids the problem of throwing out a heavily used page by inspecting the reference bit of the oldest page.

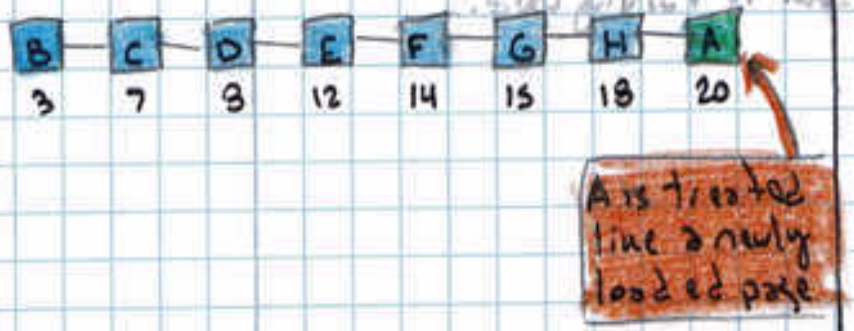
~~Pages are stored in FIFO order~~

A. If reference bit is 0, the page is both old and unused, so it is replaced immediately

B. If reference bit is 1 the bit is cleared, the page is put onto the end of the list of pages, and its load time is updated as though it had arrived immediately. Then the search continue



Presume A is the oldest page, we look at A and we say: is the reference bit to page A? We chose and A will be given a second time so we will move it to the top of the list



[redacted] self study

Second Chance Page Replacement Algorithm (continue)

- When a Page-Fault occurs:
  1. Look at the oldest page in the list
  2. If Referenced bit of the page is set (PTE), then set its Referenced bit to 0 and move the page to the front of the FIFO list; go back to step 1
  3. Select the oldest page with reference bit to 0 as the victim page.
  4. If no page with referenced bit (unlikely), then select the oldest page (FIFO)

- If all of them were referenced, then all of them have same priority, so it continues the regular FIFO terminology

- Why is this algorithm bad?

- Linux have two list FIFO so the page would move from inactive list to active list (better).